User's Manual

RX族C/C++编译程序、汇编程序、优化连接编辑程序

编译程序包 用户手册

瑞萨单片机开发环境系统

瑞萨电子
www.renesas.com

Rev.1.00 2011.03

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

前言

本手册叙述了“RX 族 C/C++ 编译程序、汇编程序、优化连接编辑程序”的使用方法。
本产品是将 C 语言、C++ 语言以及汇编语言记述的源程序转换成 RX 族的目标程序和装入模块的软件系统。
在使用前，请务必仔细阅读并理解用户手册。

■ 记载上的注意事项

在本手册的说明中使用的符号表示以下的含义：

< >	表示指定此符号括起来的内容。
[ ]	表示能省略的项目。
…	表示至少指定 1 次前一项目。
△	表示至少 1 个空格。
|	表示能选择用“|”分隔的项目。

本手册的内容对应在 IBM PC*1 及其兼容机上运行的 Microsoft® Windows® 2000、 Windows® XP 或者 Windows Vista®*2。

【注】 *1 IBM PC 是美国 International Business Machines Corporation 的注册商标。
*2 Microsoft®、 Windows® 是美国 Microsoft Corporation 在美国及其他国家的注册商标或者商标。
※ 在本手册的说明中使用的其他公司名称、产品名称和系统名称等是各公司的注册商标或者商标。

目　录

1. 概要

1.1 编译程序的构成

RX 族 C/C++ 编译程序的构成如下所示。

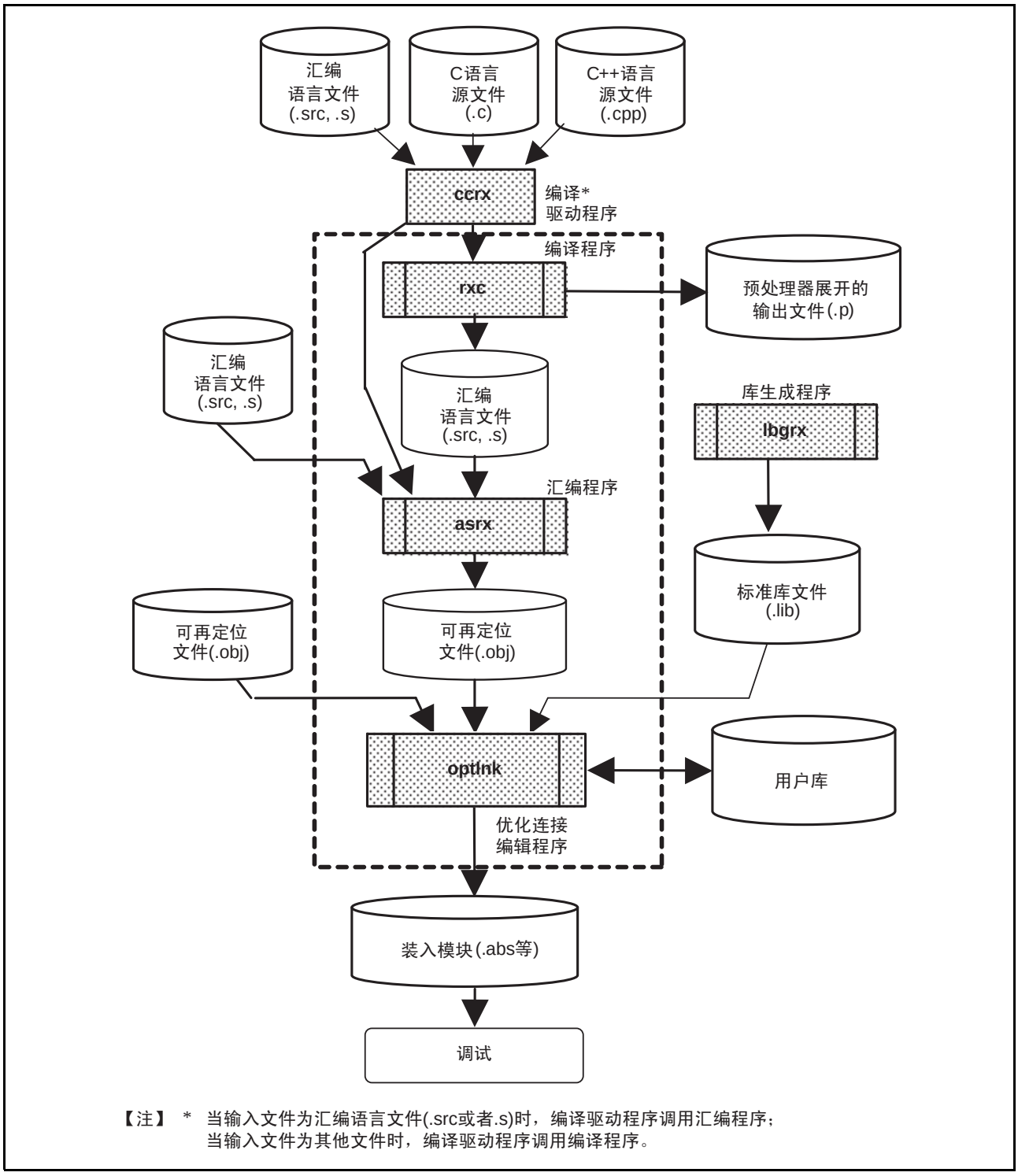


图 1.1 编译程序的构成

1.1.1 编译驱动程序的输入

给编译驱动程序输入的文件是由 ASCII 字符和 Shift-JIS 字符 （根据选项，能更改为 EUC、 Latin1 或者 UTF-8）构成并且是用 ANSI 标准 C 语言（C89/C99（可变长数组除外））、ANSI 标准 C++ 语言、EC++ 语言记述的源文件 （.c、 .cpp）和汇编语言文件 （.src、 .s）。

1.1.2 编译驱动程序的输出

输出预处理器展开的输出文件 （.p）、汇编语言文件 （.src、 .s）、可再定位文件和装入模块。

1.1.3 ccrx

ccrx 是编译驱动程序的执行文件。

ccrx 能通过指定选项，一次性进行编译到连接的处理。能在 ccrx 的启动选项 “-asmcmd”、“-lnkcmd”、“-asmopt”和 “-lnkopt”后面接着指定汇编程序 asrx 和优化连接编辑程序 optlnk 的选项。

1.1.4 asrx

asrx 是汇编程序的执行文件。

将汇编语言文件 （.src、 .s）转换为可再定位文件。

1.1.5 optlnk

optlnk 是优化连接编辑程序的执行文件。

将多个可再定位文件 （.obj）和库文件 （.lib）转换为装入模块文件 （.abs 等）或者库文件 （.lib）。

1.1.6 lbgrx

lbgrx 是库生成程序的执行文件。

根据用户指定的选项，生成标准库文件 （.lib）。

1.2 选项的指定规则

以下说明 ccrx 能利用的启动命令。

在使用这些命令前，请参照 “6. 环境变量”，确认是否设定了所需的环境变量。

1.2.1 编译程序 （ccrx）

ccrx 是编译驱动程序的启动命令。

能通过启动此命令进行编译、汇编和连接。

在输入文件的扩展名为 “.s”、“.src”、“.S” 或者 “.SRC” 时，编译程序将该文件解释为汇编语言文件（.src, .s）并且启动汇编程序。

将其他扩展名的文件作为 C/C++ 语言的源文件 （.c、 .cpp）进行编译。

【命令记述格式】

```
ccrx [ △ < 选项 > … ][ △ < 文件名 >[ △ < 选项 > …] …]
   < 选项 >：-< 选项 >[=< 子选项 >[=< 子选项 >]][,…]
```

1.2.2 汇编程序 （asrx）

asrx 是汇编程序的启动命令。

【命令记述格式】

```
asrx [ △ < 选项 > … ][ △ < 文件名 >[ △ < 选项 > …] …]
   < 选项 >：-< 选项 >[=< 子选项 >][,…]
```

1.2.3 优化连接编辑程序 （optlnk）

optlnk 是优化连接编辑程序的启动命令。

不仅进行连接处理，还包括以下功能：

- 在结合可再定位文件时进行优化。
- 建立并且编辑库文件。
- 转换为 Motorola S 格式文件、 Intel HEX 格式文件和二进制文件。

【命令记述格式】

```
optlnk [ △ < 选项 > … ][ △ < 文件名 >[ △ < 选项 > …] …]
   < 选项 >：-< 选项 >[=< 子选项 >][,…]
```

1.2.4 库生成程序 （lbgrx）

lbgrx 是库生成程序的启动命令。

【命令记述格式】

```
lbgrx [ △ < 选项 > … ]
   < 选项 >：-< 选项 >[=< 子选项 >][,…]
```

1.3 命令的记述例子

1.3.1 用 1 个命令进行编译、汇编和连接的情况

用 1 个命令执行以下的全部步骤：

- 通过ccrx对C/C++语言的源文件 （tp1.c和tp2.c）进行编译。
- 在编译后，通过asrx进行汇编。
- 在汇编后，通过optlnk进行连接，建立绝对文件 （tp.abs）。

【命令记述】

```
ccrx -cpu=rx600 -output=abs=tp.abs tp1.c tp2.c
```

【备注】

- 如果将output选项的输出格式改为 “-output=sty”，就将连接后的文件生成为Motorola S格式文件。
- 不保留在绝对文件生成过程中产生的中间文件 （汇编语言文件和可再定位文件），而只生成output选项所指定的格式文件。
- 对于ccrx，如果只想给汇编程序或者优化连接编辑程序指示有效的汇编选项或者连接选项，就必须使用-asmcmd选项、 -lnkcmd选项、 -asmopt选项或者-lnkopt选项。
- 从地址0开始分配连接对象的目标，但是不保证段的排列顺序。要指时分配地址或者段的分配顺序时，必须使用-lnkcmd选项或者-lnkopt选项给优化连接编辑程序指示选项。

1.3.2 用 1 个命令进行编译和汇编的情况

用 1 个命令执行以下步骤，并且用其他命令启动连接程序，建立 tp.abs。

- 通过ccrx对C/C++语言的源文件 （tp1.c和tp2.c）进行编译。
- 在编译后，通过asrx进行汇编，建立可再定位文件 （tp1.obj和tp2.obj）。

【命令记述】

```
ccrx -cpu=rx600 -output=obj tp1.c tp2.c
optlnk -form=abs -output=tp.abs -subcommand=cmd.sub tp1.obj tp2.obj
```

【备注】

- 如果对ccrx指示 “-output=obj”选项， ccrx就生成可再定位文件。
- 如果要更改可再定位文件名，就需要将C/C++语言的源文件逐一输入到ccrx。
- 如果将optlnk的form选项改为 “-form=sty”，就将连接后的文件生成为Motorola S格式文件。

1.3.3 用不同命令分别进行编译、汇编和连接的情况

分别用 1 个命令执行以下的各步骤：

- 通过ccrx对C/C++语言的源文件 （tp1.c和tp2.c）进行编译，建立汇编语言文件 (tp1.src和tp2.src)。
- 通过asrx对汇编语言文件 （tp1.src和tp2.src）进行汇编，生成可再定位文件(tp1.obj和tp2.obj)。
- 通过optlnk连接可再定位文件 （tp1.obj和tp2.obj），建立绝对文件 （tp.abs）。

【命令记述】

```
ccrx -cpu=rx600 -output=src tp1.c tp2.c
asrx tp1.src tp2.src
optlnk -form=abs -output=tp.abs -subcommand=cmd.sub tp1.obj tp2.obj
```

【备注】

- 如果对ccrx指示 “-output=src”选项， ccrx就生成汇编语言文件。

1.3.4 用 1 个命令进行汇编和连接的情况

用 1 个命令执行以下的全部步骤：

- 通过asrx对汇编语言文件 （tp1.src和tp2.src）进行汇编。
- 在汇编后，通过optlnk进行连接，建立绝对文件 （tp.abs）。

【命令记述】

```
ccrx -cpu=rx600 -output=abs=tp.abs tp1.src tp2.src
```

【备注】

- 从地址0开始分配连接对象的目标，但是不保证段的排列顺序。要指示分配地址或者段的分配顺序时，必须使用-lnkcmd选项或者-lnkopt选项给优化连接编辑程序指示选项。

1.3.5 用不同命令进行汇编和连接的情况

分别用 1 个命令执行以下的各步骤：

- 对汇编语言文件 （tp1.src和tp2.src）进行汇编，生成可再定位文件 （tp1.obj和tp2.obj）。
- 通过optlnk连接可再定位文件 （tp1.obj和tp2.obj），建立绝对文件 （tp.abs）。

【命令记述 1】

```
ccrx -cpu=rx600 -output=obj tp1.src tp2.src
optlnk -form=abs -output=tp.abs -subcommand=cmd.sub tp1.obj tp2.obj
```

【命令记述 2】

```
asrx -cpu=rx600 tp1.src tp2.src
optlnk -form=abs -output=tp.abs -subcommand=cmd.sub tp1.obj tp2.obj
```

2.　C/C++ 编译程序的选项

2.1　源选项

表 2.1　源选项一览表

No.	选项	对话菜单	内容
1	lang = { c 	cpp 	ecpp 	c99 }	C/C++ <Source> [Show entries for:] [Source file] [Language :] [C :] [C (C89)] [C99] [C++ :] [C++] [EC++]	作为 C （C89）语言源文件进行编译。 作为 C++ 语言源文件进行编译。 作为 EC++ 语言源文件进行编译。 作为 C （C99）语言源文件进行编译。
2	include = < 路径名 >[,…]	C/C++ <Source> [Show entries for :] [Include file directories]	指定 include 文件的路径名。
3	preinclude = < 文件名 >[,…]	C/C++ <Source> [Show entries for :] [Preinclude files]	在每个编译的起始位置包含指定的文件。
4	define = <sub>[,…] <sub>:< 宏名 >[=< 字符串 >]	C/C++ <Source> [Show entries for :] [Defines]	将 < 字符串 > 定义为 < 宏名 >。
5	undefine = <sub>[,…] <sub>:< 宏名 >	C/C++ <Source> [Show entries for :] [Undefines]	使 < 宏名 > 的预定义宏无效。
6	message nomessage[=< 错误号 > 　　[-< 错误号 >][,…]]	C/C++ <Source> [Show entries for :] [Messages] [Repressed information level messages]	信息级消息的输出有效。 信息级消息的输出无效。
7	change_message =<sub>[,…] <sub>:<level> [=<n>[-m],[…]] <level>:{Information	warning	error }	C/C++ <Other> [User defined options :]	更改编译程序输出的消息级。
8	file_inline_path=< 路径名 >[,…]	C/C++ <Source> [Show entries for :] [File inline path]	指定文件之间 inline 展开文件的路径名。
9	comment = { nest 	nonest }	C/C++ <Source> [Show entries for:] [Source file] [Allow comment nest]	允许注释 （/**/）的嵌套。 禁止注释 （/**/）的嵌套。
10	check={ nc	ch38}	C/C++ <Source> [Show entries for:] [Source file] [Interchangeability check :] [None] [NC compiler] [H8 compiler]	检查和既存程序的兼容性。

lang

格　式　　lang = { c | cpp | ecpp | c99 }

说　明　　指定源文件的语言。
当指定 lang=c 选项时，作为 C （C89）语言源文件进行编译。
当指定 lang=cpp 选项时，作为 C++ 语言源文件进行编译。
当指定 lang=ecpp 选项时，作为 Embedded C++ 语言源文件进行编译。
当指定 lang=c99 选项时，作为 C （C99）语言源文件进行编译。
在省略此选项的情况下，当扩展名为 cpp、 cc 或者 cp 时，作为 C++ 语言源文件进行编译；否则作为 C （C89）语言源文件进行编译。但是在扩展名为 src 或者 s 时，与此选项的指定无关，作为汇编语言文件进行处理。

备　注　　Embedded C++ 语言规格不支持 catch、 const_cast、 dynamic_cast、 explicit、 mutable、 namespace、 reinterpret_cast、 static_cast、 template、 throw、 try、 typeid、 typename、 using、 多重继承和虚基类。如果记述这些内容，就输出错误信息。在使用 EC++ 库时，必须指定 lang=ecpp 选项。

include

格　式　　include = < 路径名 >[,…]

说　明　　指定 include 文件的路径名。
当有多个路径名时，能用逗号 （,）分开指定。
按照 include 选项指定的文件夹、环境变量 INC_RX 指定的文件夹、环境变量 BIN_RX 指定的文件夹的顺序，检索系统 include 文件。
按照编译对象源文件的文件夹、 include 选项指定的文件夹、环境变量 INC_RX 指定的文件夹、环境变量 BIN_RX 指定的文件夹的顺序，检索用户 include 文件。

备　注　　如果多次指定此选项，指定的全部路径名都有效。

preinclude

格　式　　preinclude = < 文件名 >[,…]

说　明　　将指定的文件内容取到每个编译的起始位置。当有多个文件名时，能用逗号 （,）分开指定。
当有多个 include 选项指定的文件夹时，从左边指定的文件夹开始按顺序进行检索。

备　注　　如果多次指定此选项，指定的全部文件都为取入对象。

define

格　式　　define = <sub>[,…]
<sub> : < 宏名 > [= < 字符串 >]

说　明　　和源文件中记述的 #define 有相同的效果。
能通过记述 < 宏名 >=< 字符串 >，将 < 字符串 > 定义为宏名。
如果给子选项单独指定 < 宏名 >，就假设为该宏名已被定义。能在 < 字符串 > 中记述名字或者整数型常数。

备　注　　如果在源文件中已经通过 #define 定义了此选项指定的宏名，就优先 #define。
如果多次指定此选项，指定的全部宏名都有效。

undefine

格　式　　undefine = <sub >[,…]
<sub> : < 宏名 >

说　明　　使＜宏名＞的预定义宏无效。
当有多个宏名时，能用逗号 （,）分开指定。

备　注　　有关能指定的预定义宏，请参照 “6.2 预定义宏”。
如果多次指定此选项，指定的全部宏名都为未定义宏。

message,nomessage

格　式　　message
nomessage [= < 错误号 > [- < 错误号 >][,…]

说　明　　指定是否输出信息级消息。
当指定 message 选项时，输出信息级消息。
当指定 nomessage 选项时，抑制信息级消息的输出。如果通过子选项指定错误号，就只抑制所指定的信息级消息的输出。当有多个错误号时，能用逗号 （,）分开指定。
能通过 < 错误号 >-< 错误号 > 这样的连字符 （-）指定要抑制的错误号范围。
在省略此选项时解释为 nomessage。

备　注　　不能通过指定此选项来控制汇编程序和优化连接编辑程序的信息输出，而能通过 lnkcmd 选项指定优化连接编辑程序的 message 选项和 nomessage 选项，控制优化连接编辑程序的信息输出。
如果多次指定 nomessage 选项，就抑制所指定的全部错误号。

change_message

格　式　　change_message = <sub>[,…]
<sub> : < 错误级 >[=< 错误号 >[- < 错误号 >][,…]]
< 错误级 > : { information | warning | error }

说　明　　能更改信息和警告的消息级。
当有多个错误号时，能用逗号 （,）分开指定。

例　　change_message=information= 错误号
只将警告级的指定错误号更改为信息级。

change_message=warning= 错误号
只将信息级的指定错误号更改为警告级。

change_message=error= 错误号
只将信息级和警告级的指定错误号更改为错误级。

change_message=information
将全部警告信息更改为信息级。

change_message=warning
将全部信息级消息更改为警告级。

change_message=error
将全部信息消息和警告消息更改为错误级。

备　注　　能通过指定 nomessage 选项，抑制已更改为信息级消息的输出。
不能通过指定此选项来控制汇编程序和优化连接编辑程序的信息输出，而能通过 lnkcmd 选项指定优化连接编辑程序的 message 选项和 nomessage 选项，控制优化连接编辑程序的信息输出。
如果多次指定此选项，指定的全部错误号都有效。
错误信息不是此选项的信息级控制对象。

file_inline_path

格　式　　file_inline_path = < 路径名 >[,…]

说　明　　指定文件之间 inline 展开对象的文件路径名。
当有多个路径名时，能用逗号（,）分开指定。按照 file_inline_path 选项指定的文件夹、当前文件夹的顺序，检索文件之间 inline 展开的对象文件。

备　注　　如果多次指定此选项，指定的全部路径名都有效。

comment

格　式　　comment = { nest | nonest }

说　明　　当指定 comment=nest 时，能记述嵌套的注释。
当指定 comment=nonest 时，如果记述嵌套的注释，就发生错误。
在省略此选项时解释为 comment=nonest。

例　　/* This is an example of /* nested*/ comment*/
↑
[1]
当指定 comment=nest 时，全部解释为注释；当指定 comment=nonest 时，解释为在 [1] 处注释结束。

check

格　式　　check = { nc | ch38 }

说　明　　在将针对 R8C 族和 M16C 族的 C 编译程序以及 H8 族、 H8S 族和 H8SX 族的 C/C++ 编译程序而编制的 C/C++ 语言源文件流用到本编译程序时，能检查到影响兼容性的选项指定以及源记述。

当指定 check=nc 时，对以下情况输出信息：

- 选项：signed_char、 signed_bitfield、 bit_order=left、 endian=big、 dbl_size=4
- inline、 enum 型、 #pragma BITADDRESS、 #pragma ROM、 #pragma PARAMETER、 __asm()
- 在没有指定 -int_to_short 时，将 signed short 范围外的常数赋值到 int 型或者 signed int 型，或者将 unsigned short 范围外的常数赋值到 int 型或者 unsigned int 型。
- 将 signed short 和 unsigned short 范围外的常数赋值到 long 型或者 long long 型。
- signed short 范围外的常数和 int 型、 short 型、 char 型 （带符号 char 型除外）的比较式

当指定 check=ch38 时，对以下情况输出信息：

- 选项：unsigned_char、 unsigned_bitfield、 bit_order=right、 endian=little、 dbl_size=4
- __asm、 #pragma unpack
- 和大于 sigined long 常数的比较式
- 在没有指定 -int_to_short 时，将 signed short 范围外的常数赋值到 int 型或者 signed int 型，或者将 unsigned short 范围外的常数赋值到 int 型或者 unsigned int 型。
- 将 signed short 和 unsigned short 范围外的常数赋值到 long 型或者 long long 型。
- signed short 范围外的常数和 int 型、 short 型、 char 型 （带符号的 char 型除外）的比较式

备　注　　当 dbl_size=4 有效时， R8C 族和 M16C 族的 C 编译程序以及 H8 族、 H8S 族和 H8SX 族的 C/C++ 编译程序和浮点相关的转换 / 库函数的计算结果有可能不同。在 dbl_size=4 有效的情况下，本编译程序将 double 型和 long double 型设定为 32 位，但是各种 R8C 族和 M16C 族的 C 编译程序 （fdouble_32）以及 H8 族、 H8S 族和 H8SX 族的 C/C++ 编译程序 （double=float）只将 double 型设定为 32 位。

选项：与语言规格中未规定的安装有关的内容因各种编译程序而不同，必须通过输出的信息确认选项的选择。

扩展规格：此规格有可能影响程序的运行，必须通过输出的信息确认扩展规格的记述。

对于结构体和位域成员的分配，不通过此选项输出信息。在声明了结构体和位域成员的分配时，请参照 “9.1.2 数据的内部表示”。

在 R8C 族和 M16C 族的 C 编译程序 （不指定 fextend_to_int）的情况下，因为不通过条件式进行一般整数的扩展而生成评估代码，所以本编译程序生成的代码和运行有可能不同。

2.2 目标选项

表 2.2 目标选项一览表

No.	选项	对话菜单	内容
1	output = {prep | src | obj | abs | hex | sty} [= 文件名]	C/C++ <Object> [Output file type :] [Machine code] [Assembly source code] [Preprocessed source file]	指定输出文件的格式。 输出预处理器展开后的源文件。 输出汇编语言文件。 输出可再定位文件。 输出绝对文件。 输出 Intel HEX 格式文件。 输出 Motorola S 格式文件。
2	noline	C/C++ <Object> [Output file type :] [Suppress #line in preprocessed source file]	在预处理器展开时，抑制 #line 的输出。
3	debug nodebug	C/C++ <Object> [Generate debug information]	输出调试信息。 不输出调试信息。
4	section = <sub>[,…] <sub>: {P = < 段名 > | C = < 段名 > | D = < 段名 > | B = < 段名 > | W = < 段名 >}	C/C++ <Object> [Details] [Section :] [Program section (P)] [Const section (C)] [Data section (D)] [Uninitialized data section (B)] [Switch table section (W)]	更改段名。 程序区的段名 常数区的段名 初始化数据区的段名 未初始化数据区的段名 switch 语句转移表区的段名
5	stuff nostuff[= { B | D | C | W } [,…]]	C/C++ <Object> [Details] [Disposition of variables :] [Const section (C)] [Data section (D)] [Bss section (B)] [Switch table section (W)]	分配到对应变量调整数的段。 将没有初始值的变量分配到调整数为 4 的段。 将有初始值的变量分配到调整数为 4 的段。 将 const 型变量分配到调整数为 4 的段。 将 switch 语句转移表分配到调整数为 4 的段。

output

格　式　　output = <sub> [=< 文件名 >]
　　<sub> : { prep | src | obj | abs | hex | sty }

说　明　　指定输出文件的格式。
子选项和输出文件一览表如下所示。
在不指定 < 文件名 > 时，建立文件的文件名是在起始位置输入的源文件名后附加了下表中的扩展名。
在省略此选项时解释为 output=obj。

表 2.3　子选项输出格式

子选项	输出格式	省略文件名时的扩展名
prep	预处理器展开后的源文件	C （C89、 C99）语言源文件：p C++ 语言源文件：pp
src	汇编语言文件	src
obj	可再定位文件	obj
abs	绝对文件	abs
hex	Intel HEX 格式文件	hex
sty	Motorola S 格式文件	mot

【注】 可再定位文件是汇编程序的输出文件。
绝对文件、 Intel HEX格式文件和Motorola S格式文件是优化连接编辑程序的输出文件。

备　注　　如果指定了文件夹，就将用于建立所指定格式文件的中间文件建立到该文件夹，否则就建立到当前文件夹。

noline

格　式　　noline

说　明　　在进行预处理器展开时，抑制 #line 的输出。

备　注　　在没有指定 output=prep 选项时，此选项无效。

debug, nodebug

格　式　　debug
nodebug

说　明　　当指定 debug 选项时，输出 C 源级调试时所需的调试信息。当指定优化选项时， debug 选项也有效。
当指定 nodebug 选项时，不输出调试信息。
在省略此选项时解释为 nodebug。

section

格　式　　section = <sub>[,…]
<sub> : { P = < 段名 > |
C = < 段名 > |
D = < 段名 > |
B = < 段名 > |
W = < 段名 > }

说　明　　指定段名。
section=P=< 段名 > 指定程序区的段名。
section=C=< 段名 > 指定常数区的段名。
section=D=< 段名 > 指定初始化数据区的段名。
section=B=< 段名 > 指定未初始化数据区的段名。
section=W=< 段名 > 指定 switch 语句转移表区的段名。

< 段名 > 是不以数字开头的英文字母、数字、下划线 （_）或者 $ 的字符串。
在省略此选项时解释为 section=P=P、 C=C、 D=D、 B=B、 W=W。

备　注　　有关程序和段名的对应关系的详细内容，请参照 “8.1.2 C/C++ 程序的段”。
不能给不同区域的段指定相同的段名。
有关段名长度的翻译界限，请参照 “14. 翻译限制”。

stuff, nostuff

格　式　　stuff
nostuff [= < 段类 >[,…]]
< 段类 > : { B | D | C | W }

说　明　　当指定 stuff 选项时，根据调整数将全部变量分配到调整数为 4、 2 或者 1 的段 （表 2.4）。

表 2.4　指定 stuff 选项时的各变量和输出目标段的关系

变量的种类	变量的调整数	变量所属的段
const 限定型变量	4	C
	2	C_2
	1	C_1
有初始值的变量	4	D
	2	D_2
	1	D_1
无初始值的变量	4	B
	2	B_2
	1	B_1
switch 语句的转移表	4	W
	2	W_2
	1	W_1

当指定 nostuff 选项时，将属于指定 < 段类 > 的变量分配到调整数为 4 的段。如果省略 < 段类 >，对象就是全部段类的变量。
C、 D、 B 是由 section 选项或者 #pragma section 指定的段名， W 是由 section 选项指定的段名。总是按照定义顺序输出各段内的数据。
在省略此选项时解释为 stuff。

例
```
int a;
char b=0;
const short c=0;
struct {
    char x;
    char y;
} ST;
```

＜指定 stuff 选项时＞			＜指定 nostuff 选项时＞		
	.SECTION	C_2,ROMDATA,ALIGN=2		.SECTION	C,ROMDATA,ALIGN=4
	.glb	_c		.glb	_c
_c:			_c:		
	.word	0000H		.word	0000H
	.SECTION	D_1,ROMDATA		.SECTION	D,ROMDATA,ALIGN=4
	.glb	_b		.glb	_b
_b:			_b:		
	.byte	00H		.byte	00H
	.SECTION	B,DATA,ALIGN=4		.SECTION	B,DATA,ALIGN=4
	.glb	_a		.glb	_a
_a:			_a:		
	.blkl	1		.blkl	1
	.SECTION	B_1,DATA,ALIGN=2			
	.glb	_ST		.glb	_ST
_ST			_ST		
	.blkb	2		.blkb	2

2.3 列表选项

表 2.5 列表选项一览表

No.	选项	对话菜单	内容
1	listfile[=< 文件名 >] nolistfile	C/C++ <List> [Generate list file]	输出源列表文件。 不输出源列表文件。
2	show = <sub>[,…] <sub>: {source | conditionals | definitions | expansions }	C/C++ <List> [Contents :]	设定源列表的内容。 输出 C/C++ 源。 在进行条件汇编时输出不满足条件的行。 输出 .DEFINE 置换前的信息。 输出汇编程序的宏记述展开行。

listfile, nolistfile

格　式　　listfile[=< 文件名 >]
nolistfile

说　明　　指定是否输出源列表文件。
当指定 listfile 选项时，输出源列表文件，也能指定＜文件名＞。
当指定 nolistfile 选项时，不输出源列表文件。
如果不指定＜文件名＞，就建立和源文件相同文件名的源列表文件，其扩展名为 “lst”。
在省略此选项时解释为为 nolistfile。

备　注　　不能通过指定此选项来输出连接列表。要输出连接列表时，必须通过 lnkcmd 选项指定优化连接编辑程序的 list 选项。
将编译程序要输出的信息写到源列表。有关源列表文件的格式，请参照 “**7.2 源列表的参照方法**”。

show

格　式　　show = <sub>[,…]
　　<sub> : { source | conditionals | definitions | expansions }

说　明　　设定源列表文件的内容。
子选项和指定内容一览表如下所示。

表 2.6　子选项指定一览表

子选项	内容
source	输出 C/C++ 源。
conditionals	在进行条件汇编时也输出不满足条件的行。
definitions	输出 .DEFINE 置换前的信息。
expansions	输出汇编程序的宏记述展开行。

备　注　　此选项只在指定 listfile 选项时有效。
将编译程序要输出的信息写到源列表。有关源列表文件的格式，请参照“7.2 源列表的参照方法”。

2.4 优化选项

与优化有关的选项有可能因条件而不适用，必须通过输出代码确认该优化是否适用。

表 2.7 优化选项一览表

No.	选项	对话菜单	内容
1	optimize = { 0 | 1 | 2 | max }	C/C++ <Optimize> [Optimize level :]	指定优化级。
2	goptimize	C/C++ <Optimize> [Inter-module optimization]	输出用于模块之间优化的附加信息。
3	speed size	C/C++ <Optimize> [Speed or size :] [Optimize for speed :] [Optimize for size :]	选择优化。 进行执行性能优先的优化。 进行代码长度优先的优化。
4	loop[=< 数值 >]	C/C++ <Optimize> [Details] [Miscellaneous] [Loop expansion :]	进行最大展开数 =< 数值 > 的循环展开。
5	inline[=< 整数 >] noinline	C/C++ <Optimize> [Details] [Inline] [Automatic inline expansion :]	进行自动 inline 展开。 不进行自动 inline 展开。
6	file_inline = < 文件名 >[,…]	C/C++ <Optimize> [Details] [Inline] [Inline file path]	进行文件之间的 inline 展开。
7	case = { ifthen | table | auto }	C/C++ <Optimize> [Details] [Miscellaneous] [Switch statement :]	以 if_then 方式进行展开。 以表跳转方式进行展开。 以编译程序选择的展开方式进行展开。
8	volatile novolatile	C/C++ <Optimize> [Details] [Global variables] [Treat global variables as volatile qualified]	将外部变量 volatile 化。 不将外部变量 volatile 化。
9	const_copy noconst_copy	C/C++ <Optimize> [Details] [Global variables] [Propagate variables which are const qualified :]	进行 const 声明的外部变量的常数传递。 抑制 const 声明的外部变量的常数传递。
10	const_div noconst_div	C/C++ <Optimize> [Details] [Miscellaneous] [Division by constant :]	通过乘法指令串进行常数的乘法运算 （余数运算）。 通过除法指令串进行常数的除法运算 （余数运算）。

No.	选项	对话菜单	内容
11	library = { function | intrinsic }	C/C++ <Optimize> [Details] [Miscellaneous] [Library function :]	调用库函数。 对一部分库函数进行指令展开。
12	scope noscope	C/C++ <Optimize> [Details] [Miscellaneous] [Divide the optimization range :]	分割优化范围。 不分割优化范围。
13	schedule noschedule	C/C++ <Optimize> [Details] [Miscellaneous] [Schedule instructions :]	进行指令的重新排序。 不进行指令的重新排序。
14	map=< 文件名 > smap nomap	C/C++ <Optimize> [Optimization for access to external variables :] [Inter-module] [Inner-module] [None]	进行外部变量存取的优化。 对在编译对象文件中定义的外部变量进行外部变量存取的优化。 抑制外部变量存取的优化。
15	approxdiv	C/C++ <Optimize> [Details] [Miscellaneous] [Approximate a floating-point constant division]	进行浮点常数除法的乘法运算。
16	enable_register	C/C++ <Optimize> [Details] [Miscellaneous] [Enable register declaration]	将指定 register 存储类的变量优先分配到寄存器。
17	simple_float_conv	C/C++ <Optimize> [Details] [Miscellaneous] [Not check the range in conversion between floating point number and integer]	省略浮点型 <-> 无符号整数型的范围检查。
18	fpu nofpu	C/C++ <Optimize> [Details] [Miscellaneous] [Use floating point arithmetic instructions :]	输出使用浮点运算指令的目标。 输出不使用浮点运算指令的目标。

optimize

格　式　　optimize = { 0 | 1 | 2 | max }

说　明　　指定优化级。
当指定 optimize=0 时，不进行优化。因此，能高精度输出调试信息并且便于源级的调试。
当指定 optimize=1 时，对自动变量的寄存器分配、函数出口块的统一以及多个指令的统一（能统一的）等进行部分优化。因此，和指定 optimize=0 时相比，能缩减代码长度。
当指定 optimize=2 时，进行整体优化。但是，进行优化的内容因 size/speed 选项的选择而有些不同。
当指定 optimize=max 时，最大限度地进行优化。例如，将优化的适用范围扩大到最大限度，或者在指定 speed 选项时能进行大规模的循环展开。虽然能期待优化效果，但是有可能带来编译时间的延长以及指定 speed 选项时的代码长度大幅度增加等副作用。
在省略此选项时解释为 optimize=2。

备　注　　在各种优化选项的说明中未记述的默认内容因 optimize 选项和 speed, size 选项的指定值而发生变化。有关默认的详细内容，请参照 speed, size 选项。

goptimize

格　式　　goptimize

说　明　　在输出文件内部生成用于模块之间优化时的附加信息。
指定此选项的文件为连接时模块之间优化的对象。

speed, size

格　式　　speed
size

说　明　　当指定 speed 选项时，进行执行性能优先的优化。
当指定 size 选项时，进行代码长度优先的优化。

备　注　　当指定 speed 选项或者 size 选项时，通过指定 optimize 选项，视为已指定以下选项。如果明确指定以下选项，这些选项就有效。

表 2.8　指定的选项

＜指定 optimize=max 时＞

	循环展开	inline 展开	常数除法的乘法运算	指令的重新排序	const 限定型变量的常数传送	优化范围的分割	外部变量存取的优化
speed	loop=32	inline=250	const_div	schedule	const_copy	noscope	map* nomap*
size	loop=1	inline=0	noconst_div	schedule	const_copy	noscope	map* nomap*

【注】 * 当输入为 C/C++ 源并且输出的指定为 output=abs 或者 mot 时，默认为 map，否则默认为 nomap。

＜指定 optimize=2 时＞

	循环展开	inline 展开	常数除法的乘法运算	指令的重新排序	const 限定型变量的常数传送	优化范围的分割	外部变量存取的优化
speed	loop=2	inline=100	const_div	schedule	const_copy	scope	nomap
size	loop=1	noinline	noconst_div	schedule	const_copy	scope	nomap

＜指定 optimize=0 或者 optimize=1 时＞

	循环展开	inline 展开	常数除法的乘法运算	指令的重新排序	const 限定型变量的常数传送	优化范围的分割	外部变量存取的优化
speed	loop=1	noinline	const_div	noschedule	noconst_copy	scope	nomap
size	loop=1	noinline	noconst_div	noschedule	noconst_copy	scope	nomap

loop

格　式　　loop[=< 数值 >]

说　明　　指定是否进行循环展开的优化。
当指定 loop 选项时，展开循环语句（for、 while、 do-while）。
能用 < 数值 > 指定进行最多几倍的展开。 < 数值 > 能指定 1 ～ 32 的整数。在不指定 < 数值 > 时，为 “2”。
省略此选项时的解释取决于 optimize 选项和 speed, size 选项的指定，详细内容请参照 speed, size 选项。

inline, noinline

格　式　　inline[=< 数值 >]
noinline

说　明　　指定是否进行函数的自动 inline 展开。
当指定 inline 选项时，进行自动 inline 展开。但是，不对指定 #pragma noinline 的函数进行 inline 展开。能用 < 数值 > 指定函数大小最多增加百分之几的 inline 展开。例如，当指定 inline=100 时，进行函数大小最多增加 100%（最多 2 倍）的 inline 展开。
在指定没有数值的 inline 选项时，解释为 inline=100。
当指定 noinline 选项时，不进行自动 inline 展开。
省略此选项时的解释取决于 optimize 选项和 speed, size 选项的指定，详细内容请参照 speed, size 选项。

备　注　　指定 #pragma inline 的函数和带 inline 说明符的函数与选项的指定无关，尝试展开。

file_inline

格　式　　file_inline = < 文件名 >[,…]

说　明　　对于 < 文件名 > 指定的文件，进行跨文件的函数的 inline 展开。
当有多个文件时，能用逗号 （,）分开指定。

例　　<a.c>

```
func(){
   g();
}
```

<b.c>

```
g(){
   h();
}
```

通过指定 ccrx. -inline -file_inline=b.c a.c 进行编译， a.c 中函数 g() 的调用进行以下的展开：

```
func(){
    h();
}
```

备　注　　file_inline 选项只在指定 inline 选项或者 #pragma inline 时有效。
如果在 file_inline 选项指定的多个文件中定义了相同名字的 extern 函数，就无法保证运行（使用任选的 1 个函数定义进行 inline 展开）。
不能省略 < 文件名 > 指定的文件名的扩展名。
不能通过 file_inline 选项指定编译对象文件。
不能给 < 文件名 > 指定通配符 （*、 ?）。
如果多次指定此选项，指定的全部文件都为展开对象。

case

格　式　　case = { ifthen | table | auto }

说　明　　指定 switch 语句的代码展开方式。
当指定 case=ifthen 时，以 if_then 方式展开 switch 语句。 if_then 方式将 switch 语句的判断式的值和 case 标号的值进行比较，如果相同就跳转到 case 标号的语句进行处理，重复 case 标号的次数。对于此方式，目标码长度的增加与 switch 语句包含的 case 标号个数成正比。
当指定 case=table 时，以表格方式展开 switch 语句。表格方式将 case 标号的跳转目标确保到转移表，通过参照 1 次转移表，跳转到和 switch 语句的判断式相同的 case 标号语句。对于此方式，转移表容量的增加与 switch 语句包含的 case 标号个数成正比，但是执行速度总是固定的。将转移表输出到常数区的段。
当指定 case=auto 时，编译程序自动选择 if_then 方式或者表格方式。
在省略此选项时解释为 case=auto。

备　注　　在指定 nostuff 选项时，将在指定 case=table 时建立的转移表输出到 W 段；在没有指定 nostuff 选项时，根据 switch 语句的规模，分别输出到 W、 W_2 或者 W_1 段。

volatile, novolatile

格　式　　volatile
novolatile

说　明　　当指定 volatile 时，将全部外部变量作为 volatile 声明的变量进行处理。因此，外部变量的存取次数和存取顺序与 C/C++ 语言源文件的记述相同。
当指定 novolatile 时，对没有进行 volatile 限定的外部变量进行优化。因此，外部变量的存取次数和存取顺序有可能和 C/C++ 语言源文件的记述不同。
在省略此选项时解释为 novolatile。

const_copy, noconst_copy

格　式　　const_copy
noconst_copy

说　明　　当指定 const_copy 时，也对 const 限定型外部变量进行常数传递。
当指定 noconst_copy 时，抑制 const 限定型外部变量的常数传递。
在省略此选项时，如果指定 optimize=2 或者 optimize=max 选项就解释为 const_copy，否则解释为 noconst_copy。

备　注　　不能通过此选项控制 C++ 语言源文件的 const 限定型变量 （总是进行常数传递）。

const_div, noconst_div

格　式　　const_div
noconst_div

说　明　　当指定 const_div 时，将源文件中的整数型常数的除法和余数运算转换为使用乘法运算的指令串。
当指定 noconst_div 时，将源文件中的整数型常数的除法和余数运算转换为使用除法运算的指令串。
在省略此选项时，如果指定 speed 选项就解释为 const_div ； 如果指定 size 选项就解释为 noconst_div。

备　注　　通过移位运算进行的常数乘法运算以及通过位逻辑进行的余数运算不是 const_div 选项和 noconst_div 选项的控制对象。

library

格　式　　library = { function | intrinsic }

说　明　　当指定 library=function 时，对全部库函数进行函数调用。
当指定 library=intrinsic 时，对 abs()、 fabsf() 和能使用串操作指令的库函数进行指令展开。
在省略此选项时解释为 library=intrinsic。

scope, noscope

格　式　　scope
noscope

说　明　　当指定 scope 时，对于大容量的函数，将优化范围分割为多个范围后进行编译。
当指定 noscope 时，不分割优化范围而进行编译。优化范围的扩大使编译速度变慢，但是在一般情况下能提高目标性能。如果寄存器的数量不足，就可能降低目标性能。此选项会因程序而影响执行性能，所以请在调整性能时试一试。
在省略此选项时，如果指定 optimize=max 选项就解释为 noscope，否则解释为 scope。

schedule, noschedule

格　式　　schedule
noschedule

说　明　　当指定 schedule 时，进行指令的重新排序（考虑到流水线处理）。
当指定 noschedule 时，不进行指令的重新排序。基本上按照 C/C++ 语言源文件记述的顺序进行处理。
在省略此选项时，如果指定 optimize=2 或者 optimize=max 选项就解释为 schedule，否则解释为 noschedule。

map, smap, nomap

格　式　　map[= < 文件名 >]
smap
nomap

说　明　　进行外部变量存取的优化。
当指定 map 选项时，根据优化连接编辑程序生成的外部符号分配信息设定基址，生成以基址相对方式存取外部变量或者静态变量的代码。
当指定 smap 选项时，给在编译对象文件中定义的外部变量或者静态变量设定基址，生成以基址相对方式进行存取的代码。
map 选项的使用方法因 output 选项的指定而不同。
[output=abs 或者 mot 的情况]
只能指定 map （在指定 optimize=max 时不需要）。自动进行 2 次编译和连接，根据外部符号分配信息，生成设定了基址的代码。
[output=obj 或者 src 的情况]
必须先在不指定此选项的情况下对源文件进行一次编译，在通过优化连接编辑程序进行连接时指定 map=< 文件名 >，建立外部符号分配信息文件，然后再次给 ccrx 指定 map=< 文件名 > 进行编译。
当指定 nomap 选项时，不进行外部变量存取的优化。
当通过 map 选项进行外部变量存取的优化时，必须先在不指定此选项的情况下对源文件进行一次编译，在连接时指定 map=< 文件名 >，建立外部符号分配信息文件，然后再次指定 map=< 文件名 > 进行编译。
在省略此选项时，如果指定 optimize=max 选项就解释为 map，否则解释为 nomap。

例　　＜ C 源＞

```
long A,B,C;
void func()
{
      A = 1;
      B = 2;
      C = 3;
}
```

＜输出代码＞
(1) 不进行优化的情况

```
_func:
     MOV.L     #_A,R4
     MOV.L     #1,[R4]
     MOV.L     #_B,R4
     MOV.L     #2,[R4]
     MOV.L     #_C,R4
     MOV.L     #3,[R4]
```

(2) 进行优化的情况

```
_func:
    MOV.L    #_A,R4    ; 将 A 的地址设定为基址。
    MOV.L    #1,[R4]
    MOV.L    #2,4[R4]  ; 将 A 的地址作为基址存取 B。
    MOV.L    #3,8[R4]  ; 将 A 的地址作为基址存取 C。
```

备　注　　如果更改外部变量或者静态变量的定义顺序，就需要重新生成外部符号地址信息文件。如果指定的选项与通过 map 以外的选项进行第 1 次编译时指定的选项不同并且追加了函数内的处理，就无法保证运行。在这些情况下，必须重新生成外部符号地址信息文件。
如果同时指定 map 选项和 smap 选项， map 选项就有效。
如果在程序段之后连续分配数据段，外部变量存取的优化就无效或者可能无法进行充分的优化。
为了进行最大限度的优化，必须在连续分配多个段时，将程序段分配到末尾。
具体例子如下所示：

假设从地址 0x100 开始分配 P，紧接在 P 之后分配 C1 和 C2，从地址 0x400 开始分配 C3。此时，因为在 P 段之后接着分配了 C1 段和 C2 段，所以必须将 P 段分配到 C2 的后面。因为 C3 段不是连续分配关系，所以与 C3 无关。

approxdiv

格　式　　approxdiv

说　明　　将浮点常数的除法运算转换为常数倒数的乘法运算。

备　注　　当指定此选项时，虽然能提高浮点常数的除法运算速度，但是有可能改变运算精度。

enable_register

格　式　　enable_register

说　明　　将指定 register 存储类的变量优先分配到寄存器。

备　注　　在指定 message 选项时，如果不分配到寄存器，就输出以下的信息级消息：
C0102 (I) Register is not allocated to " 变量名 " in " 函数名 "。
但是，如果不将参数分配到寄存器，就不输出此信息。

simple_float_conv

格　式　　simple_float_conv

说　明　　对于无符号的整数型和浮点型之间的型转换，生成不检查转换对象值范围的代码。

例 1　　＜浮点型到整数型的型转换＞

```
unsigned long func(float f)
{
        return ((unsigned long)f);
}
```

不指定选项时：

```
        FCMP    #1325400064,R1  ; 0x4F000000
        BLE     L12
    L11:
        FSUB    #1333788672,R1  ; 0x4F800000
    L12:
        FTOI    R1,R1
        RTS
```

指定选项时：

```
        FTOI    R1,R1
        RTS
```

例 2　　＜整数型到浮点型的型转换＞

```
float func2(unsigned long u)
{
     return ((float)u);
}
```

不指定选项时：

```
         ITOF    R1,R5
        CMP     #-2147483648,R1
        BLTU    L12
    L11:
        FADD    #1333788672,R5  ; 0x4F800000
    L12:
        MOV     R5,R1
        RTS
```

指定选项时：

```
        ITOF    R1,R1
        RTS
```

fpu, nofpu

格　式　　fpu
nofpu

说　明　　选择是否使用浮点运算指令生成浮点运算代码。
当指定 fpu 时，生成使用浮点运算指令的代码。
当指定 nofpu 时，生成不使用浮点运算指令而在运行时调用库程序的代码。
在省略此选项时解释为 fpu。

备　注　　有关浮点运算指令的具体内容，请参照《RX 族 用户手册 软件篇》。

2.5 单片机选项

表 2.9 单片机选项一览表

No.	选项	对话菜单	内容
1	cpu = { rx600 }	CPU [CPU :]	生成用于 RX600 系列的指令代码。
2	endian = { big | little }	CPU [Endian :]	指定数据的字节序。 Big Endian Little Endian
3	round = { zero | nearest }	CPU [Details] [Round to :]	以 round to zero 方式进行舍入。 以 round to nearest 方式进行舍入。
4	denormalize = { off | on }	CPU [Details] [Denormalized number allower as a result :]	将非规格化数作为 0 进行处理。 将非规格化数作为非规格化数进行处理。
5	dbl_size = { 4 | 8 }	CPU [Details] [Precision of double :] [Single precision] [Double precision]	将 double 型和 long double 型作为单精度进行处理。 将 double 型和 long double 型作为双精度进行处理。
6	int_to_short	CPU [Details] [Replace from int with short]	将 int 型转换为 short 型，将 unsigned int 型转换为 unsigned short 型。
7	signed_char unsigned_char	CPU [Details] [Sign of char :]	将 char 型作为 signed char 型进行处理。 将 char 型作为 unsigned char 型进行处理。
8	signed_bitfield unsigned_bitfield	CPU [Details] [Sign of bit field :]	用 signed 解释位域的符号。 用 unsigned 解释位域的符号。
9	auto_enum	CPU [Details] [enum size is made the smallest]	自动选择枚举型大小。
10	bit_order = { left | right }	CPU [Details] [Bit field order :] [Upper_bit] [Lower_bit]	从左开始按顺序分配位域成员。 从右开始按顺序分配位域成员。
11	pack unpack	CPU [Details] [Pack struct, union and class]	进行结构体成员的调整数为 1 的数据调整。

No.	选项	对话菜单	内容
12	exception noexception	CPU [Details] [Use try, throw and catch of C++]	将异常处理功能设定为有效。 将异常处理功能设定为无效。
13	rtti= { on | off}	CPU [Details] [Use dynamic_cast and typeid of C++]	将 dynamic_cast、 typeid 设定为有效。 将 dynamic_cast、 typeid 设定为无效。
14	fint_register = { 0 | 1 | 2 | 3 | 4 }	CPU [Fast interrupt vector register :] [None] [R13] [R12,R13] [R11,R12,R13] [R10,R11,R12,R13]	指定只用于高速中断函数的通用寄存器。 没有高速中断专用的寄存器。 将 R13 用作高速中断专用寄存器。 将 R13 ～ R12 用作高速中断专用寄存器。 将 R13 ～ R11 用作高速中断专用寄存器。 将 R13 ～ R10 用作高速中断专用寄存器。
15	branch = { 16 | 24 | 32 }	CPU [Details] [Width of divergence of function :]	保证转移宽度不超过 16bit。 保证转移宽度不超过 24bit。 不限定转移宽度。
16	base = { rom = < 寄存器 > | ram= < 寄存器 > | < 地址值 > = < 寄存器 > }	CPU [Base register :]	指定 ROM 的基址寄存器。 指定 RAM 的基址寄存器。 指定要设定地址值的基址寄存器。
17	patch = { rx610 }	CPU [Changes code generation :]	避免各种 CPU 特有的问题。 不使用 MVTIPL 指令 （面向 RX610 群）。

cpu

格　式　　cpu = { rx600 }

说　明　　指定要生成的指令代码的单片机种类。
当指定 cpu=rx600 时，生成用于 RX600 系列的指令代码。

备　注　　根据今后单片机产品的展开，追加子选项。

endian

格　式　　endian = { big | little }

说　明　　当指定 endian=big 时，数据的字节序为 big endian。
当指定 endian=little 时，数据的字节序为 little endian。
也能用 #pragma endian 扩展名指定数据的字节序。当同时指定选项和 #pragma 扩展名时，优先 #pragma 扩展名的指定。
在省略此选项时解释为 endian=little。

round

格　式　　round = { zero | nearest }

说　明　　选择浮点常数运算的舍入方式。
当指定 round=zero 时，以 round to zero 方式进行舍入。
当指定 round=nearest 时，以 round to nearest 方式进行舍入。
在省略此选项时解释为 round=nearest。

denormalize

格　式　　denormalize = { off | on }

说　明　　指定在对浮点常数记述了非规格化数时的处理。
当指定 denormalize=off 时，将非规格化数作为 0 进行处理。
当指定 denormalize=on 时，将非规格化数作为非规格化数进行处理。
在省略此选项时解释为 denormalize=off。

dbl_size

格　式　　dbl_size = { 4 | 8 }

说　明　　指定 double 型和 long double 型的精度。
当指定 dbl_size=4 时，作为单精度浮点型 （4 字节）进行处理。
当指定 dbl_size=8 时，作为双精度浮点型 （8 字节）进行处理。
在省略此选项时解释为 dbl_size=4。

int_to_short

格　式　　int_to_short

说　明　　在将源文件中的 int 型转换为 short 型并且将 unsigned int 转换为 unsigned short 后进行编译。

备　注　　limits.h 的 INT_MAX、 INT_MIN 和 UINT_MAX 不是此选项的转换对象。
在进行 C++ 和 EC++ 编译时，此选项无效。对有可能从 C++、 EC++ 程序中参照 C 程序的外部名，输出 C1804(W)。

signed_char, unsigned_char

格　式　　signed_char
unsigned_char

说　明　　给没有指定符号的 char 型指定符号。
当指定 signed_char 时，作为 signed char 型进行处理。
当指定 unsigned_char 时，作为 unsigned char 型进行处理。
在省略此选项时解释为 unsigned_char。

备　注　　char 型的位域成员不是此选项的控制对象，必须通过 signed_bitfield 选项和 unsigned_bitfield 选项控制 char 型的位域成员。

signed_bitfield, unsigned_bitfield

格　式　　signed_bitfield
unsigned_bitfield

说　明　　给没有指定符号的位域型指定符号。
当指定 signed_bitfield 时，作为带符号型进行处理。
当指定 unsigned_bitfield 时，作为不带符号型进行处理。
在省略此选项时解释为 unsigned_bitfield。

auto_enum

格　式　　auto_enum

说　明　　将 enum 声明的枚举型数据作为枚举值范围内的最小数据型进行处理。
在省略此选项时，将枚举型大小作为 signed long 型进行处理。
枚举型能取得的值和数据型的关系如下所示。

表 2.10　枚举型能取得的值和数据型的关系

枚举值		选择的数据型
最小值	最大值	
–128	127	signed char
0	255	unsigned char
–32768	32767	signed short
0	65535	unsigned short
上述以外		signed long

bit_order

格　式　　bit_order = { left | right }

说　明　　指定位域成员的排列顺序。
当指定 bit_order=left 时，从高位开始分配成员。
当指定 bit_order=right 时，从低位开始分配成员。
也能用 #pragma bit_order 扩展名指定位域成员的排列顺序。如果同时指定选项和 #pragma，就优先扩展名的指定。
在省略此选项时解释为 bit_order=right。

pack, unpack

格　式　　pack
unpack

说　明　　指定结构体成员和类成员的调整数。
也能用 #pragma pack 扩展名指定结构体成员的调整数。如果同时指定选项和 #pragma，就优先 #pragma 扩展名的指定。结构体成员和类成员的调整数与成员的最大调整数相同。
在省略此选项时解释为 unpack。

备　注　　指定此选项时的结构体成员的调整数如下所示。

表 2.11　指定 pack 选项时的结构体成员和类成员的调整数

成员的数据型	pack	unpack	无指定
(signed) char	1	1	1
(unsigned) short	1	2	2
(unsigned) int*、(unsigned) long、(unsigned) long long、浮点型、指针型	1	4	4

【注】 * 当指定 int_to_short 选项时，和 short 相同。

exception, noexception

格　式　　exception
noexception

说　明　　当指定 exception 选项时，将 C++ 异常处理功能（try、catch、throw）设定为有效。
当指定 noexception 选项时，将 C++ 异常处理功能（try、catch、throw）设定为无效。
当指定 exception 选项时，有可能降低代码性能。
在省略此选项时解释为 noexception。

备　注　　要在文件之间将异常处理功能设定为有效时，必须进行以下指定：
(1) 指定 rtti=on。
(2) 不给优化连接编辑程序指定 noprelink 选项。
只能在 C++ 编译时指定 exception 选项。在没有指定 lang=cpp 并且输入文件的扩展名为“.c”或者“.p”时，不能指定 exception 选项，否则就发生错误。

rtti

格　式　　rtti = { on | off }

说　明　　指定运行时型信息的有效或者无效。
当指定 rtti=on 时，将 dynamic_cast、 typeid 设定为有效。
当指定 rtti=off 时，将 dynamic_cast、 typeid 设定为无效。
在省略此选项时解释为 rtti=off。

备　注　　不能将通过指定此选项而建立的可再定位文件 （.obj）注册到库，也不能通过优化连接编辑程序以可再定位格式 （.rel）输出文件，否则有可能发生符号的双重定义错误或者未定义错误。
只能在 C++ 编译时指定 rtti=on。在没有指定 lang=cpp 并且输入文件的扩展名为 “.c”或者 “.p”时，不能指定 rtti=on，否则就发生错误。

fint_register

格　式　　fint_register = { 0 | 1 | 2 | 3 | 4 }

说　明　　指定只用于高速中断函数 （函数中通过 #pragma interrupt 指定了 （fint）中断规格中的高速中断）的通用寄存器，但是在非高速中断函数中不使用被指定的寄存器。因为此选项指定的通用寄存器在高速中断函数中不需要进行保存和恢复，所以能实现高速中断函数的高速化。但是，因为减少其他函数能用的通用寄存器，所以会降低整个程序的寄存器分配效率。
选项和寄存器的关系如下所示。

表 2.12　选项和寄存器的关系

选项	高速中断专用寄存器
fint_register=0	无
fint_register=1	R13
fint_register=2	R12、 R13
fint_register=3	R11、 R12、 R13
fint_register=4	R10、 R11、 R12、 R13

在省略此选项时解释为 fint_register=0。

备　注　　如果在非高速中断函数中使用此选项指定的寄存器，就无法保证运行。如果此选项指定的对象寄存器已经被 base 选项指定，就发生错误。

branch

格　式　　branch = { 16 | 24 | 32 }

说　明　　指定转移宽度。
当指定 branch =16 时，将转移宽度限为不超过 16bit 进行编译。
当指定 branch =24 时，将转移宽度限为不超过 24bit 进行编译。
当指定 branch =32 时，不限定转移宽度。
在省略此选项时解释为 branch=24。

base

格　式　　base = { rom=< 寄存器 >
| ram=< 寄存器 >
| < 地址值 > = < 寄存器 > }
< 寄存器 > : = { R8 ～ R13 }

说　明　　指定在整个程序中被固定用作基址的通用寄存器。
当指定 base=rom=< 寄存器 A> 时，以指定的寄存器 A 相对方式存取全部 const 变量。但是，ROM 数据的总长度必须在 64KB ～ 256KB*1 以内。
当指定 base=ram=< 寄存器 B> 时，以指定的寄存器 B 相对方式存取全部的初始化变量和未初始化的变量。但是， RAM 数据的总长度必须在 64KB ～ 256KB*1 以内。
当指定 < 地址值 >=< 寄存器 C> 时，以指定的寄存器 C 相对方式存取从地址值开始的 64KB ～ 256KB*1 以内的区域。

【注】 *1　根据存取的变量长度，此值在 64KB 和 256KB 之间发生变化。

备　注　　不能给不同区域指定相同的寄存器。
1 个区域只能指定 1 个寄存器。如果用此选项指定已经被 fint_register 选项指定的寄存器，就发生错误。

patch

格　式　　patch = { rx610 }

说　明　　避免各种 CPU 特有的问题。
如果指定 -patch=rx610，就不将 RX610 群中有问题的 MVTIPL 指令用于生成的代码。如果不指定 -patch=rx610，调用内部函数 set_ipl 的生成代码中就含有 MVTIPL 指令。

2.6 汇编选项和连接选项

表 2.13 汇编选项和连接选项一览表

No.	选项	对话菜单	内容
1	asmcmd=< 文件名＞	—	用子命令文件指定 asrx 的选项。
2	lnkcmd=< 文件名 >	—	用子命令文件指定 optlnk 的选项。
3	asmopt=['']< 汇编程序选项 >['']	—	指定 asrx 的选项。
4	lnkopt=['']< 连接选项 >['']	—	指定 optlnk 的选项。

asmcmd

格　式　　asmcmd = < 文件名 >

说　明　　通过子命令文件指定传递给 asrx 的汇编选项。

例　　如果记述 ccrx-cpu=rx600 -asmcmd=file.sub sample.c，就和以下 2 行命令记述的含义相同：
ccrx -cpu=rx600 -output=src sample.c
asrx -cpu=rx600 -subcommand=file.sub sample.src

备　注　　如果多次指定此选项，指定的全部子命令文件都有效。

lnkcmd

格　式　　lnkcmd = < 文件名 >

说　明　　通过子命令文件指定传递给 optlnk 的连接选项。

例　　如果记述 ccrx -cpu=rx600 -output=abs=tp.abs － lnkcmd=file.sub tp1.c tp2.c，就和以下 3 行命令记述的含义相同：
ccrx -cpu=rx600 -output=src tp1.c tp2.c
asrx -cpu=rx600 tp1.src tp2.src
optlnk -subcommand=file.sub -form=abs -output=tp tp1.obj tp2.obj

备　注　　如果多次指定此选项，指定的全部子命令文件都有效。

asmopt

格　式　　asmopt = ["]< 汇编程序选项 >["]

说　明　　通过字符串指定传递给 asrx 的汇编程序选项。
在指定参数中含有空白字符的选项时，用双引号 （"）括起来指定。

例　　如果记述 ccrx -cpu=rx600 -asmopt="-chkpm" sample.c，就和以下 2 行命令记述的含义相同：
ccrx -cpu=rx600 -output=src sample.c
asrx -cpu=rx600 -chkpm sample.src

备　注　　如果多次指定此选项，指定的全部汇编程序选项都有效。

lnkopt

格　式　　lnkopt = ["]< 连接选项 >["]

说　明　　通过字符串指定传递给 optlnk 的连接选项。
在指定参数中含有空白字符的选项时，用双引号 （"）括起来指定。

例　　如果记述 ccrx -cpu=rx600 -output=abs=tp.abs -lnkopt="-start=P,C,D/100,B/8000" tp1.c tp2.c，就和以下 3 行命令记述的含义相同：
ccrx -cpu=rx600 -output=src tp1.c tp2.c
asrx -cpu=rx600 tp1.src tp2.src
optlnk -start=P,C,D/100,B/8000 -form=abs -output=tp tp1.obj tp2.obj

备　注　　如果多次指定此选项，指定的全部连接选项都有效。

2.7　其他选项

表 2.14　其他选项一览表

No.	选项	对话菜单	内容
1	logo nologo	— (nologo is always valid)	输出版权。 抑制版权的输出。
2	 euc sjis latin1 utf8	C/C++ <Source> [Show entries for:] [Source file] [Input character code:]	指定输入程序的字符码。 EUC 码 SJIS 码 ISO-Latin1 码 UTF-8 码
3	 outcode = { euc | sjis | utf8 }	C/C++ <Object> [Output character code:]	指定输出汇编语言文件的字符码。 EUC 码 SJIS 码 UTF-8 码
4	subcommand = < 文件名 >	—	从 < 文件名 > 指定的文件中取命令选项。

logo, nologo

格　式　　logo
nologo

说　明　　抑制版权的输出。
当指定 logo 选项时，输出版权显示。
当指定 nologo 选项时，抑制版权显示的输出。
在省略此选项时解释为 logo。

euc, sjis, latin1, utf8

格　式　　euc
sjis
latin1
utf8

说　明　　用指定的字符码处理字符串、字符常数和注释内的字符。
选项和字符码的关系如下所示。
在省略此选项时解释为 sjis。

表 2.15　选项和字符码的关系 （euc、 sjis、 latin1、 utf8）

选项	字符码
euc	EUC 码
sjis	SJIS 码
latin1	ISO-Latin1 码
utf8	UTF-8 码

备　注　　utf8 选项只在指定 lang=c99 选项时有效。

outcode

格　式　　outcode = { euc | sjis | utf8 }

说　明　　用指定的字符码输出字符串、字符常数内的字符。
选项和字符码的关系如下所示。
在省略此选项时解释为 outcode=sjis。

表 2.16　选项和字符码的关系 （outcode）

选项	字符码
euc	EUC 码
sjis	SJIS 码
utf8	UTF-8 码

备　注　　utf8 选项只在指定 lang=c99 选项时有效。

subcommand

格　式　　subcommand = < 子命令文件名 >

说　明　　在指定 subcommand 选项时，用子命令文件指定编译程序启动时的编译程序选项。子命令文件中的格式和命令行的格式相同。

备　注　　如果多次指定此选项，指定的全部子命令文件都有效。

3. 库生成程序的选项

3.1 库选项

表 3.1 库生成程序的选项一览表

No.	选项	对话菜单	内容
1	head=<sub>[,…] <sub>:{ all | runtime | ctype | math | mathf | stdarg | stdio | stdlib | string | ios | new | complex | cppstring | c99_complex | fenv | inttypes | wchar | wctype }	Standard Library <Standard Library> [Category:]	指定生成对象的库。 全部库函数和运行库 运行库 ctype.h （C89/C99）和运行库 math.h （C89/C99）和运行库 mathf.h （C89/C99）和运行库 stdarg.h （C89/C99）和运行库 stdio.h （C89/C99）和运行库 stdlib.h （C89/C99）和运行库 string.h （C89/C99）和运行库 ios （EC++）和运行库 new （EC++）和运行库 complex （EC++）和运行库 string （EC++）和运行库 complex.h （C99）和运行库 fenv.h （C99）和运行库 inttypes.h （C99）和运行库 wchar.h （C99）和运行库 wctype.h （C99）和运行库
2	output=< 文件名 >	Standard Library <Object> [Output file path:]	指定输出的库文件名。
3	nofloat	Standard Library <Object> [Simple I/O function]	生成简易的输入 / 输出函数。
4	reent	Standard Library <Object> [Generate reentrant library]	生成可重入库。
5	lang = { c | c99 }	—	生成可重入库。

head

格　式　　head = <sub>[,…]
<sub> : { all
| runtime | ctype | math | mathf | stdarg | stdio | stdlib | string | ios | new
| complex | cppstring | c99_complex | fenv | inttypes | wchar | wctype
}

说　明　　用头文件名指定生成对象。
当指定 head=all 时，将全部头文件名指定为生成对象。
运行库总是为生成对象。
在省略此选项时解释为 head=all。

output

格　式　　output = < 文件名 >

说　明　　指定输出的文件名。
在省略此选项时解释为 output=stdlib.lib。

nofloat

格　式　　nofloat

说　明　　不支持浮点转换 （%f、 %e、 %E、 %g、 %G），生成简易的输入 / 输出函数。
当输入或者输出不需要浮点转换的文件时，能减小 ROM 容量。

对象函数　　fprintf、 fscanf、 printf、 scanf、 sprintf、 sscanf、 vfprintf、 vprintf、 vsprintf

备　注　　对于指定此选项而建立的库，不保证在对象函数中输入或者输出浮点数时的运行。

reent

格　式　　reent

说　明　　生成可重入库。但是， rand 函数和 srand 函数不可重入。

备　注　　在连接可重入库时，必须在程序内包含标准 include 文件前通过 #define 定义 _REENTRANT 的宏名，或者在编译时通过 define 选项定义 _REENTRANT。

lang

格　式　　lang = { c | c99 }

说　明　　选择能使用的 C 语言标准库函数的构成。
如果选择 lang=c，就只由符合 C89 规格的内容构成 C 语言的标准函数，而不包括以 C99 规格扩展的函数。如果选择 lang=c99，就由符合 C89 规格和 C99 规格的内容构成 C 语言的标准函数。
在省略此选项时解释为 lang=c。

备　注　　C++ 库和 EC++ 库标准函数的构成不变。
如果指定 lang=c99，就能使用包括 C99 规格的全部函数，但是和指定 lang=c 时相比，函数的数量更多，因此库的生成有可能需要更多的时间。

3.2 无效的编译程序选项

除“3.1 库选项”以外，还能通过库生成程序指定 C/C++ 编译程序的选项。但是，以下选项无效：

表 3.2 无效选项一览表

No.	无效选项	生成库时选择的选项 （固定）
1	lang	无
2	include	无
3	define	无
4	undefined	无
5	message nomessage	nomessage
6	change_message	无
7	file_inline_path	无
8	comment	无
9	check	无
10	output	output=obj
11	noline	无
12	debug nodebug	nodebug
13	object noobject	无
14	listfile nolistfile show	nolistfile
15	file_inline	无
16	asmcmd	无
17	lnkcmd	无
18	asmopt	无
19	lnkopt	无
20	logo nologo	nologo
21	euc sjis latin1 utf8	无
22	outcode	无
23	subcommand	无

4. 汇编程序的选项

4.1 源选项

表 4.1 源选项一览表

No.	选项	对话菜单	内容
1	include = < 路径名 >[,...]	Assembly <Source> [Show entries for:] [Include file directories]	指定 include 文件的路径名。
2	define = <sub>[,...] <sub>:< 宏名 >= < 字符串 >	Assembly <Source> [Show entries for:] [Defines]	将 < 字符串 > 定义为 < 宏名 >。
3	chkpm	Assembly <Other> [Miscellaneous options:]	检查特权指令。
4	chkfpu	Assembly <Other> [Miscellaneous options:]	检查浮点运算指令。
5	chkdsp	Assembly <Other> [Miscellaneous options:]	检查 DSP 功能指令。

include

格 式 include = < 路径名 >[,…]

说 明 指定 include 文件的路径名。
当有多个路径名时，能用逗号 （,）分开指定。
按照当前文件夹、 include 选项指定的文件夹、环境变量 INC_RXA 指定的文件夹的顺序，检索 include 文件。

例 asrx -include=c:\usr\inc,c:\usr\rxc test.src
将文件夹 c:\usr\inc 和 c:\usr\rxc 作为 include 文件路径进行检索。

define

格 式 define = <sub>[,…]
<sub> : < 宏名 > = < 字符串 >

说 明 将宏名转换为对应的字符串。
（和在源文件的起始位置记述 .DEFINE 指示指令的情况相同）

备 注 如果同时指定 define 选项和 .DEFINE，就优先 .DEFINE。

chkpm

格　式　　chkpm

说　明　　当指定此选项时，如果记述特权指令，就通知警告 A1011。

备　注　　有关特权指令的详细说明，请参照 《RX 族 用户手册 软件篇》。

chkfpu

格　式　　chkfpu

说　明　　当指定此选项时，如果记述浮点运算指令，就通知警告 A1012。

备　注　　有关浮点运算指令的详细说明，请参照 《RX 族 用户手册 软件篇》。

chkdsp

格　式　　chkdsp

说　明　　当指定此选项时，如果记述 DSP 功能指令，就通知警告 A1013。

备　注　　有关 DSP 功能指令的详细说明，请参照 《RX 族 用户手册 软件篇》。

4.2 目标选项

表 4.2 目标选项一览表

No.	选项	对话菜单	内容
1	output=< 输出文件名 >	Assembly <Object> [Output directory]	指定可再定位文件。
2	debug nodebug	Assembly <Object> [Generate debug information]	有调试信息。 无调试信息。
3	goptimize	Assembly <Object> [Inter-module optimization]	输出用于模块之间优化的附加信息。

output

格 式 output = < 输出文件名 >

说 明 指定要输出的可再定位文件名。
当输出文件名没有扩展名时，要输出的可再定位文件名为给输出文件名附加扩展名 “.obj” 后的字符串；当输出文件名有扩展名时，要输出的可再定位文件名为用 “.obj” 替换输出文件名的扩展名后的字符串。
在不指定此选项时，输出和源文件相同文件名的可再定位文件，其扩展名为 “obj”。

debug, nodebug

格 式 debug
nodebug

说 明 当指定 debug 选项时，将调试信息输出到可再定位文件。
当指定 nodebug 选项时，不将调试信息输出到可再定位文件。
在省略此选项时解释为 nodebug。

goptimize

格 式 goptimize

说 明 输出用于模块之间优化的附加信息。
指定此选项的文件为连接时模块之间优化的对象。

4.3 列表选项

表 4.3 列表选项一览表

No.	选项	对话菜单	内容
1	listfile[=< 文件名 >] nolistfile	Assembly <List> [Generate list file]	输出汇编列表文件。 不输出汇编列表文件。
2	show = <sub>[,...] <sub>:{ conditionals | definitions | expansions }	Assembly <List> [Generate list file] [Source program:]	设定汇编列表的输出内容。 在进行条件汇编时也输出不满足条件的行。 输出用 .DEFINE 替换前的信息。 输出宏记述的展开行。

listfile, nolistfile

格　式　　listfile[=< 文件名 >]
nolistfile

说　明　　指定是否输出汇编列表文件。
当指定 listfile 选项时，输出汇编列表文件，也能指定＜文件名＞。
当指定 nolistfile 选项时，不输出汇编列表文件。
能根据 “7.1 文件名的命名方法”指定＜文件名＞。
如果不通过 listfile 选项指定＜文件名＞，就建立和源文件相同文件名的汇编列表文件，其扩展名为 “lst”。
在省略此选项时解释为 nolistfile。

show

格　式　　show = <sub>[,…]
<sub> : { conditionals
| definitions
| expansions }

说　明　　设定汇编程序要输出的列表内容。进行各指定时的输出内容如下所示。

表 4.4 show 选项指定一览表

输出种类	内容
conditionals	在进行条件汇编时也将不满足条件的行输出到汇编列表文件。
definitions	将用 .DEFINE 替换前的信息输出到汇编列表文件。
expansions	将宏记述的展开行输出到汇编列表文件。

4.4 单片机选项

表 4.5 单片机选项一览表

No.	选项	对话菜单	内容
1	cpu = { rx600 }	CPU [CPU:]	生成用于 RX600 系列的可再定位文件。
2	endian = { big | little }	CPU [Endian:]	Big Endian Little Endian
3	fint_register = { 0 | 1 | 2 | 3 | 4 }	CPU [Fast interrupt register:]	指定只用于高速中断的通用寄存器。 没有高速中断专用的寄存器。 将 R13 用作高速中断专用寄存器。 将 R13 ～ R12 用作高速中断专用寄存器。 将 R13 ～ R11 用作高速中断专用寄存器。 将 R13 ～ R10 用作高速中断专用寄存器。
4	base = { rom = < 寄存器 > | ram= < 寄存器 > | < 地址值 > = < 寄存器 > }	CPU [Base register:]	指定 ROM 的基址寄存器。 指定 RAM 的基址寄存器。 指定 SFR 的基址寄存器。
5	patch = { rx610 }	CPU [Changes code generation for the CPU types :]	避免各种 CPU 特有的问题。 不使用 MVTIPL 指令 （面向 RX610 群）。

cpu

格　式　　cpu = { rx600 }

说　明　　指定要建立的可再定位文件的单片机种类。
当指定 cpu=rx600 时，生成用于 RX600 系列的可再定位文件。

备　注　　根据今后单片机产品的展开，追加子选项。

endian

格　式　　endian = { big | little }

说　明　　当指定 endian=big 时，数据的字节序为 BigEndian。
当指定 endian=little 时，数据的字节序为 LittleEndian。
在省略此选项时解释为 endian=little。

fint_register

格　式　　fint_register = { 0 | 1 | 2 | 3 | 4 }

说　明　　将高速中断专用（由编译程序的同名选项指定）的通用寄存器的信息输出到可再定位文件。

备　注　　此选项必须在整个目标中统一指定，否则就无法保证运行。
在汇编语言文件中，不能将指定为高速中断专用的通用寄存器用于高速中断以外的用途，否则就无法保证运行。
如果此选项指定的对象寄存器已经被 base 选项指定，就发生错误。

base

格　式　　base = { | rom=< 寄存器 >
| ram=< 寄存器 >
| < 地址值 > = < 寄存器 > }
< 寄存器 > : = { R8 ～ R13 }

说　明　　将固定用作基址寄存器（由编译程序的同名选项指定）的通用寄存器的信息输出到可再定位文件。

备　注　　此选项必须在整个目标中统一指定，否则就无法保证运行。
不能将此选项指定的通用寄存器用于基址寄存器以外的用途，否则就无法保证运行。
如果对不同区域指定相同的通用寄存器，就发生错误。
如果此选项指定的通用寄存器已经被 fint_register 选项指定，就发生错误。

patch

格　式　　patch = { rx610 }

说　明　　避免各种 CPU 特有的问题。
当指定 -patch=rx610 时，将 RX610 群中有问题的 MVTIPL 指令作为未定义指令进行处理。
不认为 MVTIPL 是指令而输出错误信息 A2113(E)。

4.5 其他选项

表 4.6 其他选项一览表

No.	选项	对话菜单	内容
1	logo nologo	— (nologo is always valid)	输出版权。 抑制版权的输出。
2	subcommand = < 文件名 >	—	从文件输入命令行。
3	euc sjis latin1	—	选择 EUC 码。 选择 SJIS 码。 选择 ISO-Latin1 码。

logo, nologo

格　式　　logo
nologo

说　明　　抑制版权的输出。
当指定 logo 选项时，输出版权的显示。
当指定 nologo 选项时，抑制版权显示的输出。
在省略此选项时解释为 logo。

subcommand

格　式　　subcommand = < 文件名 >

说　明　　在指定 subcommand 选项时，用子命令文件指定汇编程序启动时的汇编程序选项。子命令文件中的格式和命令行的格式相同。

例　　＜子命令文件 opt.sub 的内容＞
-listfile
-debug

＜指定命令行＞
如果指定 (1) 的命令行，汇编程序就解释为 (2)。
(1) asrx -endian=big -subcommand=opt.sub test.src
(2) asrx -endian=big -listfile -debug test.src

euc, sjis, latin1

格　式　　euc
sjis
latin1

说　明　　用指定的字符码处理字符串、字符常数以及注释内的字符。
选项和字符码的关系如下所示。

表 4.7　选项和字符码的关系 （euc、 sjis、 latin1）

选项	字符码
euc	EUC 码
sjis	SJIS 码
latin1	ISO-Latin1 码

5. 优化连接编辑程序的操作方法

5.1 选项的指定规则

5.1.1 命令行的格式

命令行的格式如下：

```
optlnk [ {△<文件名> | △<选项列>}…]
<选项列> : -<选项>[= <子选项>[,…]]
```

5.1.2 子命令文件的格式

子命令文件的格式如下：

```
<选项> {= | △}[<子选项>[,…]] [△&] [;<注释>]
& : 继续行的指定
```

子命令文件格式的详细内容请参照 “5.2.8 子命令文件选项”。

5.2 选项解说

选项和子选项的英文大写字母表示指定缩写型时的字符，下划线表示省略时的解释。

用选项卡名 < 种类名 >[项目]…表示综合开发环境对应的对话菜单。选项的顺序对应综合开发环境的选项卡及其种类。

5.2.1 输入选项

表 5.1　输入种类选项一览表

	项目	选项	对话菜单	指定内容
1	输入文件	Input = <sub> [{,| △ }...] <sub>: < 文件名 > [(< 模块名 >[,...])]	Link/Library <Input> [Show entries for :] [Relocatable files and object files]	指定输入文件 （在命令行指定输入文件时不需要 input）。
2	库文件	LIBrary = < 文件名 >[,…]	Link/Library <Input> [Show entries for :] [Library files]	指定输入库文件。
3	二进制文件	Binary = <sub>[,...] <sub> : < 文件名 > (< 段名 > [:< 调整数 >] [,< 符号名 >])	Link/Library <Input> [Show entries for :] [Binary files]	指定输入二进制文件。
4	符号的定义	DEFine = <sub>[,...] <sub>: < 符号名 > = {< 符号名 > | < 数值 > }	Link/Library <Input> [Show entries for :] [Defines:]	对未定义符号进行强制定义。 定义为和符号名相同的值。 用数值进行定义。
5	执行的起始地址	ENTry = { < 符号名 > | < 地址 > }	Link/Library <Input> [Use entry point :]	指定入口符号。 指定入口地址。
6	预连接程序	NOPRElink	Link/Library <Input> [Prelinker control :]	抑制预连接程序的启动。

输入文件

Input

连接程序 < 输入 >[选项项目 :][可再定位文件 / 目标文件]

格　式　　Input = < 子选项 >[{,| △ }…]
< 子选项 > : < 文件名 >[(< 模块名 >[,…])]

说　明　　指定输入文件。当有多个输入文件时，用逗号 （,）或者空格分开指定。
也能用通配符 （*、 ?）指定，按照字母顺序展开通配符指定的字符串。数字排在英文字母的前面，英文大写字母排在英文小写字母的前面。
能将编译程序和汇编程序输出的目标文件、优化连接编辑程序输出的可再定位文件以及绝对文件指定为输入文件，也能用库名 （< 模块名 >）的格式将库中的模块指定为输入文件。在指定模块名时，不需要扩展名。
在没有给输入文件名指定扩展名的情况下，当没有模块名时假设为 “obj”，当有模块名时假设为 “lib”。

例
```
input=a.obj lib1(e)     ; 输入 a.obj 和 lib1.lib 内的模块 e。
input=c*.obj            ; 输入以 c 开头的扩展名为 obj 的全部文件。
```

备　注　　在指定 form=object 和 extract 时，此选项无效。
在命令行指定输入文件时，不需要 input。

库文件

LIBrary

连接程序 < 输入 >[选项项目 :][库文件]

格　式　　LIBrary = < 文件名 >[,…]

说　明　　指定库文件。当有多个库文件时，用逗号 （,）分开指定。
也能用通配符 （*、 ?）指定，按照字母顺序展开通配符指定的字符串。数字排在英文字母的前面，英文大写字母排在英文小写字母的前面。
在没有给输入文件名指定扩展名时，假设为 “lib”。
当指定 form=library 选项或者 extract 选项时，将库文件作为编辑对象库进行输入，否则就在被指定为输入文件的文件之间进行连接处理，然后检索库文件中的未定义符号。
按照库选项指定的用户库文件 （指定顺序）、库选项指定的系统库文件 （指定顺序）、默认库 （环境变量 HLNK_LIBRARY1、 2、 3）的顺序，检索库文件中的符号。

例
```
library=a.lib,b         ; 输入 a.lib 和 b.lib。
library=c*.lib          ; 输入以 c 开头的扩展名为 lib 的全部文件。
```

二进制文件

Binary

连接程序 < 输入 >[选项项目 :][二进制文件]

格　式　　Binary = < 子选项 >[,…]
< 子选项 > : < 文件名 >(< 段名 >[:< 调整数 >][,< 符号名 >])
< 调整数 > : 1 | 2 | 4 | 8 | 16 | 32 （默认值为 1）

说　明　　指定输入二进制文件。当有多个二进制文件时，用逗号（,）分开指定。
在没有给文件名指定扩展名时，假设为 “bin”。
将输入的二进制数据作为指定的段数据进行分配。通过 start 选项指定段地址，不能省略段。
也能通过指定符号，作为定义符号进行连接。如果是在 C/C++ 程序中参照的变量名，就在程序中的参照名之前附加 “_”。
能给此选项指定的段指定调整数。能指定的调整数值为 2 的乘方，不能指定其他值。
在没有指定调整数时，默认值为 1。

例

```
input=a.obj
start=P,D*/200
binary=b.bin(D1bin),c.bin(D2bin:4,_datab)
form=absolute
```

将 b.bin 作为 D1bin 段，从地址 0x200 开始分配。
将 c.bin 作为 D2bin 段（调整数为 4），分配到 D1bin 之后。
将 c.bin 数据作为定义符号 _datab 进行连接。

备　注　　在指定 form={object | library} 或者 strip 时，此选项无效。
在没有指定输入目标文件时，不能指定此选项。

符号定义

DEFine

连接程序 < 输入 >[选项项目 :][符号定义]

格　式　　DEFine = < 子选项 >[,…]
< 子选项 > : < 符号名 > = {< 符号名 > | < 数值 >}

说　明　　用外部定义符号或者数值对未定义符号进行强制定义。
用 16 进制数指定数值。当开头为 A ～ F 时，先检索符号，如果没有对应的符号，就解释为数值；当开头附加 0 时，总是解释为数值。当符号名为 C/C++ 变量名时，在程序中的定义名之前附加 “_”；当符号名为 C++ 函数名时（main 函数除外），用双引号将程序中含有参数串的定义名括起来指定。但是，在参数为 void 时，用 “函数名 ()” 指定。

例

```
define=_sym1=data       ; 将 _sym1 定义为和外部定义符号 data 相同的值。
define=_sym2=4000       ; 将 _sym2 定义为 0x4000。
```

备　注　　在指定 form={object | relocate | library} 时，此选项无效。

执行的起始地址

ENTry

连接程序 < 输入 >[入口点 :]

格　式　　ENTry = {< 符号名 > | < 地址 >}

说　明　　用外部定义符号或者地址指定执行的起始地址。
用 16 进制数指定地址。当开头为 A ～ F 时，先检索定义符号，如果没有对应的符号，就判断为地址；当开头附加 0 时，总是解释为地址。
当符号名为 C 函数名时，在程序中的定义名之前附加 “_”；当符号名为 C++ 函数名时（main 函数除外），用双引号将程序中含有参数串的定义名括起来指定。但是，在参数为 void 时，用 “函数名 ()” 指定。
如果在编译或者汇编时指定了 entry 符号，就优先此选项的指定。

例

```
entry=_main             ; 将 C/C++ 的 main 函数设定为执行的起始地址。
entry="init()"          ; 将 C++ 的 init 函数设定为执行的起始地址。
entry=100               ; 将 0x100 设定为执行的起始地址。
```

备　注　　在指定 form={object | relocate | library} 或者 strip 时，此选项无效。
在指定未参照符号的删除优化（optimize=symbol_delete）时，必须指定执行的起始地址，否则未参照符号的删除优化就无效。当单片机种类为 RX 族时，如果用此选项指定了地址，就将未参照符号的删除优化设定为无效。

预连接程序

NOPRElink

连接程序 < 输入 >[预连接程序的控制 :]

格　式　　NOPRElink

说　明　　抑制预连接程序的启动。
预连接程序支持 C++ 模板示例的自动生成功能和运行时的型检查功能。如果没有使用 C++ 模板功能和运行时的型检查功能，就必须指定 noprelink 选项，缩短连接时间。

备　注　　在指定 extract 或者 strip 时，此选项无效。

5.2.2 输出选项

表 5.2　输出种类选项一览表

	项目	选项	对话菜单	指定内容
1	输出格式	FOrm = { Absolute | Relocate | Object | Library [= {S | U}] | Hexadecimal | Stype | Binary }	Link/Library <Output> [Type of output file :]	绝对格式 可再定位格式 目标格式 库格式 Intel HEX 格式 Motorola S 格式 二进制格式
2	调试信息	DEBug SDebug NODEBug	Link/Library <Output> [Debug information :]	输出 （输出到文件）。 输出调试信息文件。 不输出。
3	记录长度的统一	REcord = { H16 | H20 | H32 | S1 | S2 | S3 }	Link/Library <Output> [Data record header :]	Intel HEX 记录 Intel 扩展 HEX 记录 Intel 32bitHEX 记录 S1 记录 S2 记录 S3 记录
4	ROM 化支持	ROm = <sub>[,...] <sub> : <ROM 段名 > =<RAM 段名 >	Link/Library <Output> [Show entries for :] [ROM to RAM mapped sections:]	确保 RAM 区，用 RAM 地址将符号重新定位。
5	输出文件	OUtput = <sub>[,…] <sub> : < 文件名 > [=< 输出范围 >] < 输出范围 >: { < 起始地址 > - < 结束地址 > | < 段名 >[:...] }	Link/Library <Output> [Show entries for :] [Output file path/ Messages] or [Divided output files:]	指定输出文件 （能指定范围以及分割输出）。
6	外部符号分配信息文件	MAp [= < 文件名 >]	Link/Library <Output> [Generate external symbol-allocation information file]	指定外部符号分配信息文件的输出 （面向 SuperH 族和 RX 族）。
7	空区域输出的指定	SPace [= {< 数值 > | Random}]	Link/Library <Output> [Specify value filled in unused area] [Output padding data]	指定空区域的输出值。
8	信息消息	Message NOMessage [= <sub>[,...]] <sub> : < 错误号 > [- < 错误号 >]	Link/Library <Output> [Show entries for :] [Output file path/ Messages] [Repressed information level messages:]	输出 不输出 （能指定错误号和范围）
9	没被参照的定义符号的通知	MSg_unused	Link/Library <Output> [Show entries for :] [Notify unused symbol:]	通过输出信息，通知一次也没被参照的定义符号。

	项目	选项	对话菜单	指定内容
10	段内数据的紧凑分配	DAta_stuff	Link/Library <Output> [Show entries for :] [Reduce empty areas of boundary alignment:]	在紧凑每个编译之间的空区域后分配数据 （面向 SuperH 族、 H8 族、 H8S 族和 H8SX 族）。
11	数据记录的字节数指定	BYte_count=< 数值 >	Link/Library <Output> [Length of data record :]	指定数据记录的最大字节数。
12	CRC 运算	CRc = < 子选项 > < 子选项 >: < 输出位置 >=< 计算范围 >[/ < 多项式 >][:< 字节序 >] < 输出位置 >:< 地址 > < 计算范围 >: < 起始地址 >-< 结束地址 >[,...] < 多项式 >：{ CCITT | 16 } < 字节序 >：{BIG|LITTLE}	Link/Library <Output> [Show entries for :] [Generate CRC code]	在连接时进行计算范围内的 CRC （Cyclic Redundancy Check）运算，将计算结果填入到输出位置。
13	段尾的填充	PADDING	Link/Library <Output> [Padding]	根据调整数，将填充输出到段尾。
14	特定向量号地址的设定	VECTN=< 子选项 >[,...] < 子选项 >: < 向量号 >={< 符号 > |< 地址 >}	Link/Library <Output> [Show entries for :] [Vector] [Specific vector :]	给可变向量的特定向量号设定地址 （面向 RX 族）。
15	可变向量的空区域地址的设定	VECT={< 符号 >|< 地址 >}	Link/Library <Output> [Show entries for :] [Vector] [Empty vector :]	给可变向量的空区域设定地址（面向 RX 族）。

输出格式

FOrm

连接程序 < 输出 >[输出格式 :]

格　式　　FOrm = {Absolute | Relocate | Object | Library[={S|U}]
| Hexadecimal | Stype | Binary}

说　明　　指定输出格式。
在省略此选项时解释为 form=absolute。子选项一览表如表 5.3 所示。
RX 族不支持 form=relocate。

表 5.3　form 选项的子选项一览表

	子选项名	内　　容
1	absolute	输出绝对文件。
2	relocate	输出可再定位文件。
3	object	输出目标文件。在通过 extract 选项从库中取出 1 个模块作为目标文件时，使用此子选项。
4	library	输出库文件。 当指定 library=s 时，将输出库文件作为系统库。 当指定 library=u 时，将输出库文件作为用户库。 在省略时解释为 library=u。
5	hexadecimal	输出 Intel HEX 格式文件。 IntelHEX 格式请参照 “16.1.2 Intel HEX 格式的文件”。
6	stype	输出 Motorola S 格式文件。 Motorola S 格式请参照 “16.1.1 Motorola S 格式的文件”。
7	binary	输出二进制文件。

备　注　　输出格式和输入文件及其他选项的关系如表 5.4 所示。

表 5.4　输出格式和输入文件及其他选项的关系

	输出格式	指定选项	能输入的文件格式	能指定的选项 *1
1	Absolute	有 strip	绝对文件	input、 output
		上述以外	目标文件 可再定位文件 二进制文件 库文件	input、 library、 binary、 debug/nodebug、 sdebug、 cpu、 ps_check、 start、 rom、 entry、 output、 map、 hide、 optimize/nooptimize、 samesize、 symbol_forbid、 samecode_forbid、 variable_forbid、 function_forbid、 section_forbid、 absolute_forbid、 profile、 cachesize、 sbr、 compress、 rename、 delete、 define、 fsymbol、 stack、 noprelink、 memory、 msg_unused、 data_stuff、 show=symbol、 reference、 xreference
2	Relocate	有 extract	库文件	library、 output、 show=symbol、 reference
		上述以外	目标文件 可再定位文件 二进制文件 库文件	input、library、debug/nodebug、output、hide、 rename、 delete、 noprelink、 msg_unused、 data_stuff、 show=symbol、 xreference
3	Object	有 extract	库文件	library、 output、 show=symbol
4	Hexadecimal Stype Binary		目标文件 可再定位文件 二进制文件 库文件	input、library、binary、cpu、ps_check、start、 rom、 entry、 output、 map、 space、 optimize/ nooptimize、 samesize、 symbol_forbid、 samecode_forbid、 variable_forbid、 function_forbid、 section_forbid、 absolute_forbid、 profile、 cachesize、 sbr、 rename、 delete、 define、 fsymbol、 stack、 noprelink、 record、 s9*2、 byte_count*3、 memory、 msg_unused、 data_stuff、 show=symbol、 reference、 xreference
			绝对文件	input、 output、 record、 s9*2、 byte_count*3、 show=symbol、 reference、 xreference
5	Library	有 strip	库文件	library、 output、 memory*4、 show=symbol、 section
		有 extract	库文件	library、 output、 show=symbol、 section
		上述以外	目标文件 可再定位文件	input、 library、 output、 hide、 rename、 delete、 replace、 noprelink、 memory*4、 show=symbol、 section

【注】 *1　总是能指定 message/nomessage、 change_message、 logo/nologo、 form、 list 和 subcommand。
*2　只能在输出格式为 form=stype 时指定 s9。
*3　只能在输出格式为 form= hexadecimal 时指定 byte_count。
*4　在指定 hide 时，不能使用。

调试信息

DEBug
SDebug
NODEBug

连接程序 < 输出 >[调试信息 :]

格　式　　DEBug
SDebug
NODEBug

说　明　　指定是否输出 debug 信息。
debug 选项将调试信息输出到输出文件。
sdebug 选项将调试信息输出到 < 输出文件名 >.dbg 文件。
nodebug 选项不输出调试信息。
如果在指定 form=relocate 时指定 sdebug 选项，就解释为 debug 选项。
当通过 output 选项指定多个文件的输出时，如果指定 debug 选项，就解释为 sdebug 选项，输出到 < 起始输出文件名 >.dbg。
在省略此选项时解释为 debug。

备　注　　在指定 form={object | library | hexadecimal | stype | binary}、strip 或者 extract 时，此选项无效。

记录长度的统一

REcord

连接程序 < 输出 >[记录长度的统一 :]

格　式　　REcord = {H16 | H20 | H32 | S1 | S2 | S3}

说　明　　与地址范围无关，以固定的数据记录进行输出。
如果存在大于指定的数据记录的地址，就根据地址选择数据记录。
在省略此选项时，根据各地址输出各种数据记录。

备　注　　在没有指定 form=hexadecimal 或者 stype 时，此选项无效。

ROM 化支持

ROm

连接程序 < 输出 >[选项项目 :][从 ROM 映像到 RAM 的段]

格　式　　ROm = < 子选项 >[,…]
< 子选项 > : <ROM 段名 >=<RAM 段名 >

说　明　　确保初始化数据区的 ROM 区和 RAM 区，将 ROM 段内的定义符号重新定位为 RAM 段内的地址。
给 ROM 段指定有初始值的可再定位段。
给 RAM 段指定不存在的段或者容量为 0 的可再定位段。

例　　rom=D=R
start=D/100,R/8000
确保和 D 段相同容量的 R 段，将 D 段内的定义符号重新定位为 R 段内的地址。

备　注　　在指定 form={object | relocate | library} 或者 strip 时，此选项无效。

输出文件

OUtput

连接程序 < 输出 >[选项项目 :][输出文件信息的抑制][输出文件的分割]

格　式　　OUtput = < 子选项 >[,…]
< 子选项 > : < 文件名 >[=< 输出范围 >]
< 输出范围 > : {< 起始地址 >-< 结束地址 > | < 段名 >[:…]}

说　明　　指定输出文件名。当 form={absolute | hexadecimal | stype | binary} 时，能指定多个文件。能用 16 进制数指定地址。当开头为 A ～ F 时，先检索段，如果没有对应的段，就判断为地址。当开头附加 0 时，总是解释为地址。
在省略此选项时解释为 < 起始输入文件名 >.< 默认扩展名 >。
默认扩展名如下：
form=absolute：“abs”、form=relocate：“rel”、form=object：“obj”
form=library：“lib”、form=hexadecimal：“hex”、form=stype：“mot”
form=binaruy：“bin”

例　　output=file1.abs=0-ffff,file2.abs=10000-1ffff
将 0 ～ 0xffff 输出到 file1.abs，将 0x10000 ～ 0x1ffff 输出到 file2.abs。
output=file1.abs=sec1:sec2,file2.abs=sec3
将 sec1 段和 sec2 段输出到 file1.abs，将 sec3 段输出到 file2.abs。

备　注　　当单片机种类为 RX 族并且为大端法时，如果以段为单位进行输出，就必须将段的容量设定为 4 的倍数。

外部符号分配信息文件的输出

MAp

连接程序 < 输出 >[外部符号分配信息文件的输出]

格　式　　MAp [= < 文件名 >]

说　明　　输出编译程序在外部变量存取优化中使用的外部变量分配信息文件。
如果不指定 < 文件名 >，就输出 output 选项指定的文件名或者起始输入文件名的文件（扩展名为 bls）。
如果建立外部变量分配信息文件时的变量声明顺序和读被重新编译后的目标时的变量声明顺序发生变化，就输出错误。

备　注　　此选项只在指定 form={absolute | hexadecimal | stype | binary} 时有效。
SuperH 族和 RX 族的单片机有效。

空区域输出的指定

SPace

连接程序 < 输出 >[选项项目 :][空区域输出的指定][空区域的输出]

格　式　　SPace [= {< 数值 > | Random}]

说　明　　通过用户指定的数据，填充输出范围的存储器空区域。
能指定随机数或者 16 进制数的数值为填充数据。
根据指定 output 选项时的输出范围的指定方法，空区域的填充方法有以下不同点：

- 输出范围：段的指定
  如果在指定的段之间存在空区域，就输出指定的数据。
- 输出范围：地址范围的指定
  如果在指定的范围内存在空区域，就输出指定的数据。

输出数据的长度以 1、 2、 4 字节为有效单位。输出数据的长度取决于 space 选项指定的 16 进制数的个数。如果指定 3 字节数据，就对高位进行 0 扩展，作为 4 字节数据进行处理；如果指定奇数位的数据，也对高位进行 0 扩展，作为偶数位的输入进行处理。
如果空区域的容量不是输出数据长度的倍数，就尽量输出并且发出警告消息。

备　注　　如果此选项没有指定数值，就不输出到空区域。
此选项只在指定 form={ binary |stype | hexadecimal} 选项时有效。
如果没有通过 output 选项指定输出范围，此选项的指定就无效。

信息级消息

Message
NOMessage

连接程序 < 输出 >[选项项目 :][输出文件信息的抑制][信息级消息的抑制]

格　式

```
Message
NOMessage [= < 子选项 >[,…]]
< 子选项 > : < 错误号 >[-< 错误号 >]
```

说　明　指定是否输出信息级消息。
当指定 message 选项时，输出信息级消息。
当指定 nomessage 选项时，抑制信息级消息的输出。如果指定错误号，就能抑制所指定的错误号的消息输出。也能使用连字符 （-）指定要抑制的错误号范围。如果将警告级消息号和错误级消息号指定为错误号，就假设由 change_message 更改为信息级并且抑制消息的输出。
在省略此选项时解释为 nomessage。

例

```
nomessage=4,200-203,1300
```

抑制 L0004、 L0200 ～ L0203 和 L1300 的消息输出。

没被参照的定义符号的通知

MSg_unused

连接程序 < 输出 >[选项项目 :][信息输出的指定][没被参照的定义符号的通知]

格　式

```
MSg_unused
```

说　明　如果指定此选项，就通过消息输出将连接处理中一次也没被参照的外部定义符号通知用户。

例

```
optlnk -msg_unused a.obj
```

备　注
- 在输入文件为 absolute 格式时，此选项的指定无效。
- 为了输出消息，需要同时指定 message 选项。
- 有可能在编译时对已进行 inline 展开的函数输出消息。此时，能通过对函数定义进行 static 声明，抑制消息的输出。
- 在以下的任意情况下，因为不能正确地解析参照关系，所以通过输出消息通知的信息不正确。
  - 在汇编时没有指定 goptimize 选项并且在同一文件中有向同一段转移的情况 （只限于单片机为 H8 族、 H8S 族或者 H8SX 族的情况）。
  - 参照同一文件中的常数符号。
  - 在编译时优化有效并且调用直接从属函数。
  - 在编译时外部变量存取的优化有效 （只限于单片机为 SuperH 族的情况）。
  - 在源文件中记述 #pragma tbr 时直接指定偏移值 （只限于单片机为 SH-2A/SH2A-FPU 的情况）。
  - 通过连接时的优化，统一常数和字面常量。

段内数据的紧凑分配

DAta_stuff

连接程序 < 输出 >[选项项目 :][段内数据的紧凑分配]

格　式　　DAta_stuff

说　明　　在连接时，紧凑分配段内的数据。此选项功能的对象段是常数区、初始化数据区和未初始化数据区。
如果指定此选项，就在紧凑因每个编译段的调整而产生的空区域后进行连接。
但是，不更改数据的分配顺序。
如果不指定此选项，就根据每个编译段的调整进行连接。通过指定此选项，能紧凑因调整产生的冗长空区域以及减小整个数据段的容量。

例

```
<tp1.c>                 <tp2.c>
----------------        --------------------
long a;                 char  d;
char b,c;               long  e;
                        char  f;
```

< 编译后的数据段容量 （SuperH 族编译程序的输出例子） >

```
  tp1.obj : 4+1+1 = 6 字节
  tp2.obj : 1+3[*]+4+1 = 9 字节
```

< 连接 tp1.obj 和 tp2.obj 后的数据段容量 >

1) 没有指定 data_stuff 的情况

根据各段的调整，连接目标文件 （常规处理）。

```
  6 字节 [tp1] + 2 字节 [*] + 9 字节 [tp2] = 17 字节
```

2) 指定 data_stuff 的情况

紧凑分配段内的数据，在填充因调整而产生的冗长空区域后进行连接。

```
  (4+1+1) 字节 + 1 字节 + 1 字节 [*] + 4 字节 + 1 字节 = 13 字节
```

【注 1】　* ：因调整而产生的空区域。

【注 2】 编译后的数据段容量因编译时的选项指定而发生变化，可能和上述例子不同。

备　注　　如果在连接已指定 SuperH 族编译程序 smap 选项的目标文件时指定此选项，就无法保证运行。
此选项功能不适用于汇编程序输出的目标文件。
在以下的任意条件下，此选项的指定无效：

- 指定form=library,object时
- 输入绝对文件时
- 指定memory=low时
- 没有指定nooptimize时

连接时的优化不适用于指定此选项而建立的可再定位文件。
在单片机种类为 RX 族时，不能使用此功能。

数据记录的字节数指定

BYte_count

连接程序 < 输出 >[数据记录的长度 :]

格　式　　BYte_count=< 数值 >

说　明　　在生成 Intel-Hex 格式的文件时，此选项用于指定数据记录的字节数最大值。能指定 1byte 的 16 进制数 （01 ～ FF）的字节数。在不记述此选项时，以字节数最大值 FF 生成 Intel-Hex 文件。

例　　byte_count=10

备　注　　如果生成的文件格式不是 Intel-Hex 格式 （form=hex），此选项就无效。

CRC 运算

CRc

连接程序 < 输出 >[选项项目 :][CRC 码]

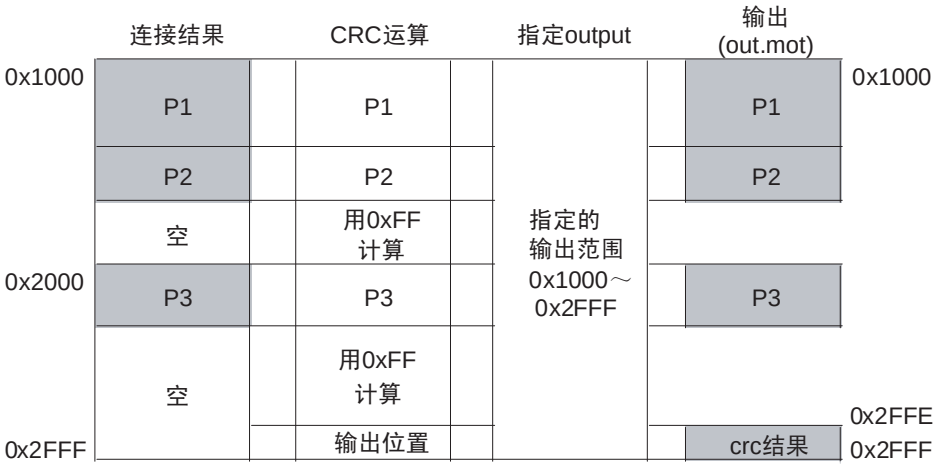
格　式　　CRc = < 子选项 >
< 子选项 >: < 输出位置 >=< 计算范围 >[/< 多项式 >][:< 字节序 >]
< 输出位置 > : < 地址 >
< 计算范围 > : < 起始地址 >-< 结束地址 >[,...]
< 多项式 > : { CCITT | 16 }
< 字节序 > : {BIG|LITTLE}

说　明　　按照从低位地址到高位地址的顺序，对通过计算范围指定的内容进行 CRC （Cyclic Redundancy Check）运算，将计算结果输出到输出位置的地址。
字节序是能对 RX 族指定的选项。当指定字节序时，根据字节序将计算结果输出到输出位置的地址，否则就以绝对文件的字节序将计算结果输出到输出位置的地址。
多项式可选择 CRC-CCITT 或者 CRC-16 （默认值为 CRC-CCITT）。

多项式
CRC-CCITT
X^16+X^12+X^5+1
位表示 （10001000000100001）
CRC-16
X^16+X^15+X^2+1
位表示 （11000000000000101）

例 1

```
optlnk *.obj -form=stype -start=P1,P2/1000,P3/2000
        -crc=2FFE=1000-2FFD -output=out.mot=1000-2FFF
```



crc 选项：-crc=2FFE=1000-2FFD

对 0x1000 ～ 0x2FFD 的区域进行 CRC 运算，将结果输出到地址 0x2FFE。

在没有指定 space 选项时，假设指定 space=0xFF，对计算范围内的空区域进行 CRC 运算。

output 选项：-output=out.mot=1000-2FFF

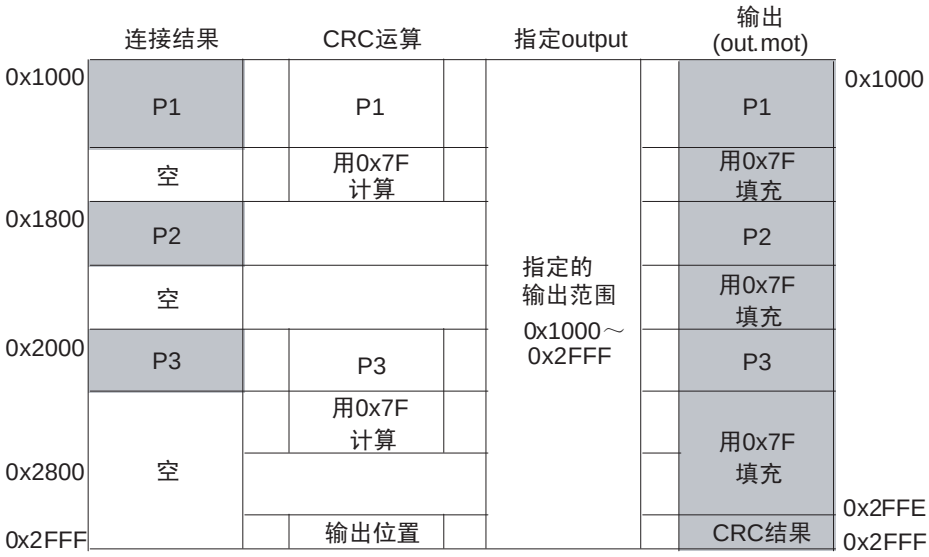
因为没有指定 space 选项，所以不将空区域输出到 “out.mot” 文件。在空区域用 0xFF 进行 CRC 运算，但是不填充 0xFF。

【注】 1. CRC 输出位置不在计算范围内。

2. CRC 输出位置必须在 output 选项的输出范围内。

例 2

```
optlnk *.obj -form=stype -start=P1/1000,P2/1800,P3/2000
        -space=7F -crc=2FFE=1000-17FF,2000-27FF
        -output=out.mot=1000-2FFF
```



crc 选项：-crc=2FFE=1000-2FFD

对 0x1000 ～ 0x17FF 的区域和 0x2000 ～ 0x27FF 的区域进行 CRC 运算，将结果输出到地址 0x2FFE。

能将不连续的多个计算范围指定为计算对象，进行 CRC 运算。

space 选项：-space=7F

用 space 选项的值 （0x7F）计算被指定的计算范围的空区域。

output 选项：-output=out.mot=1000-2FFF

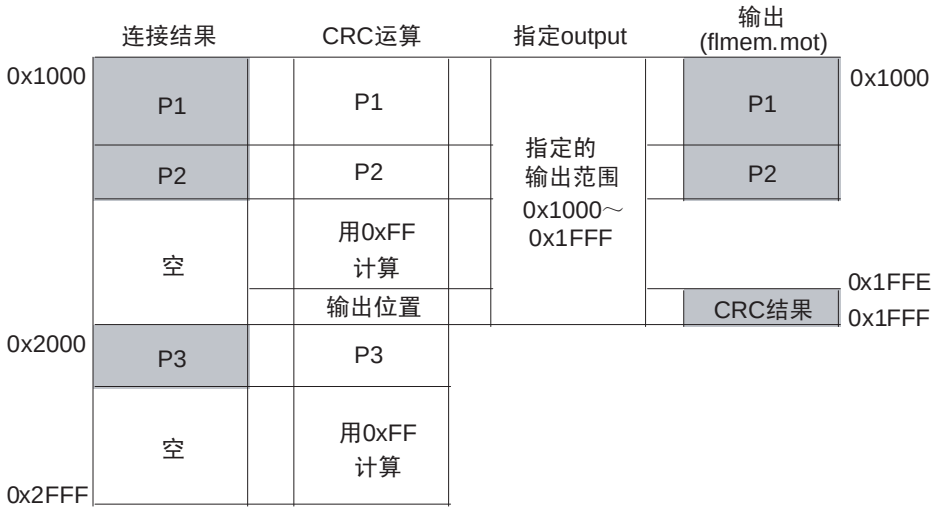
因为指定了 space 选项，所以将空区域输出到 “out.mot” 文件，并且用 0x7F 填充空区域。

【注】 1. CRC 运算的计算顺序不是计算范围的指定顺序，而是按照从低位地址到高位地址的顺序进行计算。

2. 在同时指定 crc 选项和 space 选项时，不能给 space 选项指定 random 或者大于等于 2 字节的值。必须指定 1 字节的数据。

例 3

```
optlnk *.obj -form=stype -start=P1,P2/1000,P3/2000
        -crc=1FFE=1000-1FFD,2000-2FFF
        -output=flmem1.mot=1000-1FFF
```



crc 选项：-crc=1FFE=1000-1FFD,2000-2FFF

对 0x1000 ～ 0x1FFD 的区域和 0x2000 ～ 0x2FFF 的区域进行 CRC 运算，将结果输出到地址 0x1FFE。

在没有指定 space 选项时，假设指定 space=0xFF，对计算范围内的空区域进行 CRC 运算。

output 选项：-output=flmem1.mot=1000-1FFF

因为没有指定 space 选项，所以不将空区域输出到 “flmem1.mot” 文件。

用 0xFF 对空区域进行 CRC 运算，但是不填充 0xFF。

备　注　　当输入多个绝对文件时，此选项无效。

此选项在输出格式为 form={hexadecimal | stype} 时有效。

如果没有指定 space 选项并且存在未输出到计算范围的空区域，就当作给空区域设定了 0xFF 进行 CRC 计算。

如果 CRC 运算的计算范围包括覆盖指定的区域，就发生错误。

样本代码

此样本代码用于比较 crc 选项计算得出的 CRC 运算结果。

样本代码的程序和 optlnk 的 CRC 运算结果相同。

多项式 CRC-CCITT 的情况

```
typedef  unsigned   char    uint8_t;
typedef  unsigned   short   uint16_t;
typedef  unsigned   long    uint32_t;

uint16_t CRC_CCITT(uint8_t *pData, uint32_t iSize)
{
  uint32_t   ui32_i;
  uint8_t       *pui8_Data;
  uint16_t   ui16_CRC = 0xFFFFu;

  pui8_Data = (uint8_t *)pData;

  for(ui32_i = 0; ui32_i < iSize; ui32_i++)
  {
    ui16_CRC  = (uint16_t) ((ui16_CRC >> 8u) |
                  ((uint16_t)((uint32_t)ui16_CRC << 8u)));
    ui16_CRC ^= pui8_Data[ui32_i];
    ui16_CRC ^= (uint16_t)((ui16_CRC & 0xFFu) >> 4u);
    ui16_CRC ^= (uint16_t) ((ui16_CRC << 8u) << 4u);
    ui16_CRC ^= (uint16_t)(((ui16_CRC & 0xFFu) << 4u) << 1u);
  }
  ui16_CRC = (uint16_t)( 0x0000FFFFul &
               ((uint32_t)~(uint32_t)ui16_CRC) );
  return ui16_CRC;
}
```

多项式 CRC-16 的情况

```
#define POLYNOMIAL 0xa001       // 生成多项式 CRC-16

typedef  unsigned   char    uint8_t;
typedef  unsigned   short   uint16_t;
typedef  unsigned   long    uint32_t;

uint16_t CRC16(uint8_t *pData, uint32_t iSize)
{
  uint16_t crcdData = (uint16_t)0;
  uint32_t data = 0;
  uint32_t i,cycLoop;

  for(i=0;i<iSize;i++){
    data = (uint32_t)pData[i];
    crcdData = crcdData ^ data;
    for (cycLoop = 0; cycLoop < 8; cycLoop++) {
        if (crcdData & 1) {
            crcdData = (crcdData >> 1) ^ POLYNOMIAL;
        } else {
            crcdData = crcdData >> 1;
        }
    }
  }
  return crcdData;
}
```

段尾填充

PADDING

连接程序 < 输出 >[填充 :]

格　式　　PADDING

说　明　　为了使段容量为段调整数的倍数，将数据填充到段尾。

例　　-start=P,C/0 -padding
P 段的调整数：4 字节
P 段的容量：0x06 字节
C 段的调整数：1 字节
C 段的容量：0x03 字节
在上述情况下，将 2 字节的填充数据填充到 P 段并且将容量调整为 0x08 字节，然后进行连接。
-start=P/0,C/7 -padding
P 段的调整数：4 字节
P 段的容量：0x06 字节
C 段的调整数 :1 字节
C 段的容量：0x03 字节
在上述情况下，将 2 字节的填充数据填充到 P 段并且将容量调整为 0x08 字节，然后进行连接，因为重复使用了 C 段，所以输出 L2321 错误。

备　注　　生成的填充数据值为 0x00。
因为不填充绝对地址段，所以必须由用户调整绝对地址段的容量。

特定向量号的地址设定

VECTN

连接程序 < 输出 >[选项项目 :][特定向量]

格　式　　VECTN = < 子选项 >[,...]
< 子选项 > : < 向量号 > = {< 符号 > | < 地址 >}

说　明　　对可变向量表（C$VECT 段）的特定向量号设定由选项指定的地址。
如果使用此选项，即使在源文件中没有记述中断函数，也将可变向量表建立为 C$VECT 段，并且将地址设定到向量表中。

对于 < 向量号 >，必须用 10 进制数指定 0 ～ 255 范围内的向量号。
对于 < 符号 >，必须用对象函数的外部名指定符号。
对于 < 地址 >，必须用 16 进制数指定地址。

例　　-vectn=30=_f1,31=0000F100 ; 给向量号 30 设定 _f1 的地址，
; 给向量号 31 设定 0x0f100。

备　注　　此选项在单片机种类为 RX 族时有效。
在用户通过源程序建立 C$VECT 段时，因为不自动生成可变向量表，所以此选项无效。

空向量区的地址设定

VECT

连接程序 < 输出 >[选项项目 :][空向量]

格　式　　VECT={< 符号 >|< 地址 >}

说　明　　在可变向量表（C$VECT 段）中对未设定地址的向量号设定此选项指定的地址。
如果使用此选项，即使在源文件中没有记述中断函数，也将可变向量表建立为 C$VECT 段，并且将地址设定到向量表中。
对于 < 符号 >，必须记述对象函数的外部名。
对于 < 地址 >，必须用 16 进制数记述要设定的地址。

备　注　　此选项在单片机种类为 RX 族时有效。
在用户通过源程序建立 C$VECT 段时，因为不自动生成可变向量表，所以此选项无效。
通过记述 {< 符号 >|< 地址 >}，将以 0 开始的记述全部判断为地址。

5.2.3 列表选项

表 5.5 列表种类选项一览表

	项目	命令行格式	对话菜单	指定内容
1	列表文件	LISt [= < 文件名 >]	Link/Library <List> [Generate list file]	指定列表文件的输出。
2	列表内容	SHow [= <sub>[,...]] <sub> : {SYmbol | Reference | SEction | Xreference | Total_size | VECTOR | ALL }	Link/Library <List> [Contents :]	 符号信息 参照次数 段信息 对照表信息 段的总容量 向量信息的输出 全部信息的输出

列表文件

LISt

连接程序 < 列表 >[连接列表的输出]

格　式　　`LISt [= < 文件名 >]`

说　明　　指定列表文件的输出和列表文件名。
在不指定列表文件名时，建立的列表文件的文件名和输出（或者起始输出）文件的文件名相同，其扩展名在指定 `form=library` 或者 `extract` 时为 “`lbp`”，否则为 “`map`”。

列表内容

SHow

连接程序 < 列表 >[列表内容 :]

格　式　　SHow[= <sub>[,…]]
　　<sub> : { SYmbol | Reference | SEction | Xreference |
　　Total_size
　　| VECTOR | ALL }

说　明　　指定列表的输出内容。
子选项一览表如**表 5.6** 所示。
有关各列表的具体例子，请参照 “7.3 连接列表的参照方法”和 “7.4 库列表的参照方法”。

表 5.6　show 选项的子选项一览表

	输出格式	子选项名	含义
1	form=library 或者指定 extract 时	symbol	输出模块内的符号名一览表。
		reference	不能指定。
		section	输出模块内的段一览表。
		xreference	不能指定。
		total_size	不能指定。
		vector	不能指定。
		all	不能指定 （指定 extract 时）。 输出模块内的符号名一览表和段一览表 （指定 form=library 时）。
2	form≠library 和没有指定 extract 时	symbol	输出符号地址、长度、种类和优化内容。
		reference	输出符号的参照次数。
		section	不能指定。
		xreference	输出对照表信息。
		total_size	表示各 ROM 分配对象和 RAM 分配对象的段的总容量。
		vector	输出向量信息。
		all	输出和指定 show=symbol、 xreferrence、 total_size 时相同的内容 （form=rel）。 输出和指定 show=symbol、 total_sizel 时相同的内容 （form=rel、data_stuff）。 输出和指定 show=symbol、reference、xreference、total_size 时相同的内容 （form=abs）。 输出和指定 show=symbol、reference、xreference、total_size 时相同的内容 （form=hex/stype/bin）。 当 form=obj 时，不能指定。

备　注　　当指定选项 form 和选项 show 并且 show=all 时，有效或者无效的组合如下：

		Symbol	Reference	Section	Xreference	Vector	Total_size
form=abs	只指定 show	有效	有效	无效	无效	无效	无效
	show=all	有效	有效	无效	有效	有效	有效
form=lib	只指定 show	有效	无效	有效	无效	无效	无效
	show=all	有效	无效	有效	无效	无效	无效
form=rel	只指定 show	有效	无效	无效	无效	无效	无效
	show=all	有效	无效	无效	有效 *1	无效	有效
form=obj	只指定 show	有效	有效	无效	无效	无效	无效
	show=all	无效	无效	无效	无效	无效	无效
form=hex/ bin/sty	只指定 show	有效	有效	无效	无效	无效	无效
	show=all	有效	有效	无效	有效	有效 *1	有效 *1

*1　此选项在输入文件为 absolute 格式时无效。

对照表信息的输出有下述限制：

- 在输出文件为relocatable格式并且使用data_stuff选项时，不能输出对照表信息。
- 在输入文件为absolute格式时，不能输出参照侧地址的信息。
- 如果在汇编时没有指定goptimize选项，就不输出同一文件内转移的有关信息 （只限于单片机为H8族、 H8S族和H8SX族的情况）
- 不输出有关同一文件内的常数符号的参照信息。
- 在编译时优化有效并且调用直接从属函数时，不输出信息。
- 在外部变量存取的优化有效的情况下，除基本符号以外，不输出变量的参照信息 （只限于单片机为SuperH族和RX族的情况）。
- 在源文件中记述#pragma tbr时直接指定偏移量的情况下，不输出该函数的有关信息（只限于单片机为SH-2A/SH2A-FPU的情况）
- 当指定连接时的优化时，如果统一常数和字面常量，就不输出有关该常数和字面常量的参照信息。
- show=total_size表示的信息和其他选项total_size显示的内容相同。
- 在单片机种类为RX族时，能使用show=vector。
- 在show=reference有效的情况下，将#pragma address指定的变量参照次数作为0进行输出 （只限于单片机为SuperH族和RX族的情况）

5.2.4　优化选项

表 5.7　优化种类选项一览表

	项目	命令行格式	对话菜单	指定内容
1	优化	OPtimize [= <sub>[,...]] <sub> :　{ STring_unify 	SYmbol_delete 	Variable_access 	Register 	SAMe_code 	SHort_format 	Function_call 	Branch 	SPeed 	SAFe } NOOPtimize	Link/Library <Optimize> [Show entries for :] [Optimize items] [Optimize :]	进行优化。 统一常数和字符串。 删除未参照的符号。 有效利用短绝对寻址方式。 进行寄存器保存 / 恢复的优化。 统一共用码。 缩短寻址方式。 有效利用间接寻址方式。 优化转移指令。 进行执行速度优先的优化。 进行安全的优化。 不进行优化。
2	共用码的长度	SAMESize = < 长度 > (省略时：sames=1e)	Link/Library <Optimize> [Eliminated size :]	指定共用码统一对象的最小长度。
3	配置信息	PROfile = < 文件名 >	Link/Library <Optimize> [Include profile :]	指定配置信息文件 （进行动态优化）。
4	高速缓存的容量	CAchesize = <sub> <sub>: Size = < 容量 >| 　　Align = < 块的容量 > (省略时：ca=s=8,a=20)	Link/Library <Optimize> [Cache size :]	 指定高速缓存的容量。 指定高速缓存块的容量 （面向 SuperH 族）。
5	优化的 部分抑制	SYmbol_forbid = 　　< 符号名 >[,...] SAMECode_forbid = 　　< 函数名 >[,...] Variable_forbid = 　　< 符号名 >[,...] FUnction_forbid = 　　< 函数名 >[,...] SEction_forbid = <sub>[,...] 　　<sub> : [< 文件名 >| 　　< 模块名 >] 　　(< 段名 >[,...]) Absolute_forbid = 　< 地址 > [+ < 容量 >] [,...]	Link/Library <Optimize> [Show entries for :] [Forbid item]	未参照符号删除的抑制符号 共用码统一的抑制符号 有效利用短绝对寻址方式的抑制符号 有效利用间接寻址方式的抑制符号 优化抑制段 优化抑制的地址范围

优化

OPtimize
NOOPtimize

连接程序 < 优化 >[优化方法 :][优化设定][设定 :]

格　式

```
OPtimize[= < 子选项 >[,…]]
NOOPtimize
< 子选项 > : {STring_unify | SYmbol_delete | Variable_access
            | Register | SAMe_code | SHort_format
            | Function_call | Branch | SPeed | SAFe}
```

说　明

指定是否进行模块之间的优化。
当指定 `optimize` 选项时，对在编译或者汇编时指定 `goptimize` 选项的文件进行优化。
当指定 `nooptimize` 选项时，不进行模块之间的优化。
在省略此选项时解释为 `optimize`。子选项一览表如**表 5.8** 所示。

表 5.8　optimize 选项的子选项一览表

子选项	含义	优化对象程序 *1					
		SHC	SHA	H8C	H8A	RXC	RXA
无参数	进行全部优化。	○	×	○	○	○	○
string_unify	对有 const 属性的常数，统一相同值的常数。有 const 属性的常数包括以下内容： • C/C++ 程序中的 const 限定型变量 • 字符串数据的初始值 / 字面常量	○	×	○	×	×	×
symbol_delete	删除一次也没有被参照的变量 / 函数。必须指定 entry 选项。	○	×	○	×	○	×
variable_access	通过 8/16 位绝对寻址方式将存取次数多的变量分配到能存取的区域。必须在编译和汇编时指定 cpu 选项。	×	×	○	○	×	×
register	解析函数的调用关系，重新分配寄存器并且删除冗长的寄存器保存 / 恢复代码。必须指定 entry 选项。	○	×	○	×	×	×
same_code	将多个相同的指令串进行子程序化。	○	×	○	×	×	×
short_format	在能缩短位移量 / 立即数的代码长度时，替换为代码长度短的指令。	×	×	○	○	×	×
function_call	如果在 0 ～ 0xFF 的范围内有空区域，就分配存取次数多的函数地址。在单片机种类为 H8SX 族时，也使用以下区域： H8SXN ：0x100 ～ 0x1FF H8SXM、H8SXA、H8SXX：0x200 ～ 0x3FF 必须在编译和汇编时指定 cpu 选项。	×	×	○	○	×	×
branch	根据程序的分配信息，优化转移指令长度。如果执行其他优化项目，无论是否指定此选项，都必须执行此选项。	○	×	○	○	○	○

子选项	含义	优化对象程序 *1					
		SHC	SHA	H8C	H8A	RXC	RXA
speed	执行不可能降低目标速度的优化，和 optimize=string_unify、 symbol_delete、 varaible_access,、 register、 short_format、 branch 相同。	○	×	○	○	○ *2	○ *2
safe	执行不可能受变量和函数属性限制的优化，和 optimize=string_unify、 register、 short_format、 branch 相同。	○	×	○	○	○ *3	○ *3

【注】 *1 SHC：SuperH 族 C/C++ 程序 SHA：SuperH 族的汇编程序
H8C：H8族、 H8S族和H8SX族C/C++程序 H8A：H8族、 H8S族和H8SX族的汇编程序
RXC：RX族C/C++程序 RXA：RX族汇编程序
*2 symbol_delete 和 branch 有效。
*3 branch 有效。

备　注
- 当指定 form=｛object | relocate | library｝或者 strip 时，此选项无效。
- 如果在编译时指定外部变量存取的优化，常数 / 字面常量的统一优化（optimize=string_unify）就无效。
- 只有在单片机种类为 H8SX 族时， optimize=short_format 的指定才有效。
- 在单片机种类为 SH-2A/SH2A-FPU 时，代码长度有可能因 optimize=register 的功能而增加。

共用码的长度

SAMESize

连接程序 < 优化 >[统一长度 :]

格　式　　SAMESize = < 长度 >

说　明　　通过统一优化共用码（optimize=same_code），指定优化对象的最小代码长度。必须用 16 进制数指定 8 ～ 7FFF 之间的值。
在省略此选项时解释为 samesize=1E。

备　注　　在没有指定 optimize=same_code 时，此选项无效。

配置信息

PROfile

连接程序 < 优化 >[配置信息 :]

格　式　　PROfile = < 文件名 >

说　明　　指定配置信息文件。
只能将瑞萨综合开发环境 Ver. 2.0 以上的输出配置信息文件指定为配置信息文件。
如果指定配置信息文件，就能通过模块之间的优化，进行基于动态信息的优化。
受配置信息输入影响的优化如表 5.9 所示。

表 5.9　配置信息和优化的关系

子选项	含义	优化对象程序 *1			
		SHC	SHA	H8C	H8A
variable_access	优先分配动态存取次数多的变量。	×	×	○	○
function_call	降低动态存取次数多的函数优化的优先级。	×	×	○	○
branch	将动态调用次数多的函数分配在调用源函数的附近。 在 SuperH 族程序的情况下，在考虑 cachesize 选项指定的高速缓存容量的基础上进行分配优化。	○	Δ *2	○	Δ

【注】 *1 SHC：SuperH 族 C/C++ 程序　SHA：SuperH 族汇编程序
H8C：H8族、 H8S族和H8SX族的C/C++程序　H8A：H8族、 H8S族和H8SX族的汇编程序
*2 不以函数为单位而以输入文件为单位进行移动。

备　注　　在没有指定 optimize 时，此选项无效。

高速缓存的容量

CAchesize

连接程序 < 优化 >[高速缓存的容量 :]

格　式　　CAchesize = <sub>
<sub>:Size = < 容量 > | Align = < 块容量 >

说　明　　指定高速缓存和高速缓存块的容量。
在指定 profile 选项时，用于转移指令的优化 （optimize=branch）。
必须用 16 进制数指定以千字节为单位的容量以及以字节为单位的块容量。
在省略此选项时解释为 cachesize=size=8,align=20。

备　注　　在没有指定 profile 时，此选项无效。

优化的部分抑制

SYmbol_forbid
SAMECode_forbid
Variable_forbid
FUnction_forbid
SEction_forbid
Absolute_forbid

连接程序 < 优化 >[优化方法 :][优化的部分抑制]

格　式

```
SYmbol_forbid = < 符号名 >[,…]
SAMECode_forbid = < 函数名 >[,…]
Variable_forbid = < 符号名 >[,…]
FUnction_forbid = < 函数名 >[,…]
SEction_forbid  = <sub>[,…]
   <sub>: [< 文件名 >|< 模块名 >](< 段名 >[,…])
Absolute_forbid = < 地址 >[+ < 容量 >][,…]
```

说　明　抑制特定的符号、段和地址范围的优化。必须用 16 进制数指定地址和容量。对于 C/C++ 变量名和 C 函数名，在程序中的定义名前附加 “_”。在 C++ 函数的情况下，用双引号程序中含有参数串的定义名括起来指定。但是在参数为 void 时，用 “函数名 ()” 指定。各选项的含义如表 5.10 所示。

表 5.10　优化的部分抑制选项一览表

选项	参数	含义
symbol_forbid	函数名 | 变量名	抑制未参照符号的删除优化。
samecode_forbid	函数名	抑制共用码的统一优化。
variable_forbid	变量名	抑制有效利用短绝对寻址方式的优化。
function_forbid	函数名	抑制有效利用间接寻址方式的优化。
section_forbid	段名 文件名 模块名	抑制特定段的优化。通过同时指定输入文件名或者库模块名，不仅能对整个段，而且能对特定文件限定优化抑制对象。
absolute_forbid	地址 [+ 容量]	抑制地址 + 容量的范围优化。

例

```
symbol_forbid="f(int)"   ; 即使参照次数为 0，也不删除 C++ 函数 f(int)。
section_forbid=(P1)      ; 抑制 P1 段的全部优化。
section_forbid=a.obj(P1,P2)
; 抑制 a.obj 内的 P1 段和 P2 段的全部优化。
```

备　注　在不使用优化的连接处理中，此选项无效。
在对记述路径的输入文件进行优化抑制时，必须通过 section_forbid 选项给文件名记述路径。

5.2.5 段选项

表 5.11 段种类选项一览表

	项目	命令行格式	对话菜单	指定内容
1	段地址	STARt= <sub>[,...] <sub> : [(]< 段名 > [{: | ,} < 段名 >[,...]] [)][,...] [/< 地址 >]	Link/Library <Section> [Show entries for :] [Section]	指定段的起始地址。
2	符号地址文件	FSymbol = < 段名 >[,...]	Link/Library <Section> [Show entries for :] [Symbol file]	输出外部定义符号地址的定义文件。

段地址

STARt

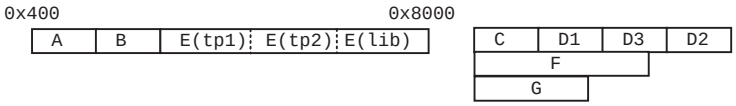
连接程序 < 段 >[设定项目 :][段]

格　式　　STARt = <sub> [,…]
<sub> : [(]< 段名 >[{:|,} < 段名 >[,…]][)][,…][/< 地址 >]

说　明　　指定段的起始地址。必须用 16 进制数指定段的起始地址。
也能用通配符 “*” 指定段名。按照输入顺序展开通配符指定的段。
能通过用冒号 “:” 分开段，将多个段分配到相同的地址 （段的覆盖分配）。
在分配到相同地址的指定段之间，按照指定顺序进行分配。
能用圆括号 “()” 括起来更改覆盖分配的对象段。
按照输入文件的指定顺序和输入库的指定顺序，分配同一段内的目标。
如果没有指定地址，就从地址 0 开始分配。
将没有用 start 选项指定的段分配到最后的分配地址之后。

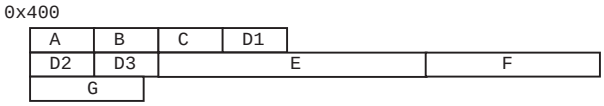
例　　按照以下顺序，举例说明输入目标时的段分配：
（括号内是有目标的各段）
tp1.obj(A,D1,E) -> tp2.obj(B,D3,F) -> tp3.obj(C,D2,E,G)

(1) -start=A,B,E/400,C,D*:F:G/8000



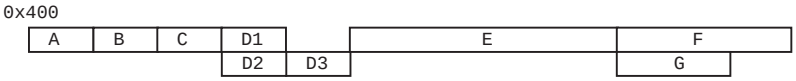
- 用 “:” 分开的C、 F、 G段分配在相同地址。
- 按照输入顺序，分配用通配符记述的段 （此处是以D开头的段）。
- 在同名的段内 （此处是E段），从输入的目标开始按顺序分配。
- 将由库输入的同名段 （此处是E段）分配到输入的目标之后。

(2) -start=A,B,C,D1:D2,D3,E,F:G/400



- 以紧接在 “:” 后的段 （此例中是A、 D2、 G）为起始位置，分别分配在相同的起始地址。

(3) -start=A,B,C,(D1:D2,D3),E,(F:G)/400



- 如果用 “()” 将相同地址的分配括起来，就以紧接在 “()” 前的段（此例中为C、E）之后为起始位置，分配 “()” 内的相同地址。
- 将紧接在 “()” 之后的段 （此处为E）分配到 “()” 内的最后的段之后。

备　注　　在指定 form={object | relocate | library} 或者 strip 时，此选项无效。
不能嵌套记述括号 “()”。
在括号 “()” 内至少需要记述 1 个冒号 “:”。如果不记述冒号 “:”，就不能记述括号 “()”。
如果记述括号 “()”，就不能在 “()” 外记述冒号 “:”。
如果使用括号 “()” 记述此选项，连接程序的优化功能就无效。

符号地址文件

FSymbol

连接程序 < 段 >[设定项目 :][符号地址文件]

格　式　　FSymbol = < 段名 >[,…]

说　明　　以汇编程序的控制指令格式，将指定的段内外部定义符号输出到文件。
文件名为 < 输出文件 >.fsy。

例　　
```
fsymbol=sct2,sct3
output=test.abs
```
将段 sct2 和段 sct3 的外部定义符号输出到 test.fsy。

[test.fsy 的输出例子]
```
;OPTIMIZING LINKAGE EDITOR GENERATED FILE 1999.11.26
;fsymbol = sct2, sct3

;SECTION NAME = sct2
 .export _f
_f: .equ h'00000000
 .export _g
_g: .equ h'00000016
;SECTION NAME = sct3
 .export _main
_main: .equ h'00000020
 .end
```

备　注　　在指定 form={object | relocate | library} 或者 strip 时，此选项无效。
在单片机种类为 H8 族、 H8S 族、 H8SX 族和 SuperH 族时，能使用此选项。

5.2.6　　验证选项

表 5.12　验证种类选项一览表

	项目	命令行格式	对话菜单	指定内容
1	地址一致性的检查	CPu = { <cpu 信息文件名 > | < 存储器种类 > = < 地址范围 >[,...] | STRIDE } < 存储器种类 > = { ROm | RAm | XROm | XRAm | YROm | YRAm | FIX} < 地址范围 >: < 起始地址 > - < 结束地址 >	Link/Library <Verify> [CPU information check :]	指定能分配的段地址范围。 给段分割的对象指定段名。
2	物理空间的重叠检查	PS_check=<sub>[:<sub>...] <sub>: <LS>,<LS>[,...] <LS>: < 起始地址 > -< 结束地址 >	Link/Library <Verify> [Physical space overlap check :]	指定物理空间的重叠地址范围。
3	非段分割对象的指定	CONTIGUOUS_SECTION = < 段名 >[,…]	Link/Library <Verify> [Not divide the specified section :]	给非段分割对象的段指定段名。

CPu

连接程序 < 验证 >[地址一致性的检查 :]

格　式

```
CPu = { <cpu 信息文件名 >
      | < 存储器种类 > = < 地址范围 >[,…]
      | STRIDE}
＜存储器种类＞ = { ROm | RAm | XROm | XRAm | YROm | YRAm | FIX }
＜地址范围 > : < 起始地址 > - < 结束地址 >
```

说　明

在没有指定 cpu=stride 时，如果段的分配地址不在地址范围内，就输出错误。
在指定 cpu=stride 时，如果段的分配地址不在地址范围内，就将段分配到下一个相同存储器种类的区域或者在将段分割后进行分配。

[例] 不指定子选项 stride 的情况

```
start=D1,D2/100
cpu=ROM=100-1FF,RAM=200-2FF
```

当 D1 在 100-1FF 的范围内并且 D2 在 200-2FF 的范围内时，正常结束，否则就输出错误。

[例] 指定子选项 stride 的情况

```
start=D1,D2/100
cpu=ROM=100-1FF,RAM=200-2FF,ROM=300-3FF
cpu=stride
```

当 D1 和 D2 在 ROM 属性的区域内 （分割或者不分割段）时，正常结束。即使分割段也不在范围内时，就输出连接错误。

xrom/xram 指定 DSP 的 X 存储区， yrom/yram 指定 DSP 的 Y 存储区。
必须用 16 进制数指定能进行段分配的地址范围。 ROM/RAM 的属性用于模块之间的优化。
给存储器种类 “FIX” 指定固定地址的区域 （I/O 区等）。
如果存储器种类 “FIX” 和其他存储器种类的地址范围重叠，就将存储器种类 “FIX” 设定为有效。

如果存储器种类为 ROM 或者 RAM 并且段不在地址范围内，子选项 stride 就将段分割后分配到相同存储器种类的区域。
通过子选项 stride，以模块为单位分割段。
[例]

```
cpu=ROM=0-FFFF,RAM=10000-1FFFF
```

检查段地址是否在 0-FFFF 或者 10000-1FFFF 之间。
在模块之间的优化过程中，在不同属性之间不进行目标转移。

```
cpu=ROM=100-1FF,ROM=400-4FF,RAM=500-5FF
cpu=stride
```

如果段地址不在 100-1FF 之间，就以模块为单位将段分割后分配到 400-4FF。

备　注

在指定 form={object | relocate | library} 或者 strip 时，此选项无效。
在单片机种类不是 SH2DSP、 SH3DSP、 SH4ALDSP 时，存储器种类的 xrom、 xram、 yrom、 yram 的指定无效。
在 cpu=stride 和 optimize=register 有效时，有可能输出 L2230 错误。此时，必须将 optimize=register 设定为无效。
在指定 cpu=stride 并且分割 B 段时，对于要初始化为 0 的信息， C$BSEC 段的容量只增加 8 字节 × 分割数的容量。

物理空间的重叠检查

PS_check

连接程序 < 验证 >[物理空间的重叠检查 :]

格　式　　PS_check=<sub>[:<sub>...]
<sub> : <LS>,<LS>[,...]
<LS> : < 起始地址 >-< 结束地址 >

说　明　　此选项用于检测地址值没有重叠而实际分配在存储器时有重叠的目标。
通过使用此选项，对于 SH3 和 SH4 等，能检测到逻辑地址不重叠而实际分配在存储器时有重叠的目标。
如果此选项检测到重叠，就以错误结束连接处理。
必须给选项记述存储器中重叠的地址范围 （格式中的 <LS>）。
要检查多个物理存储器时，能通过用冒号 “:” 分开记述进行检查。

例　　在 SH4 的 MMU 无效时，将 4G 字节的地址空间映像到 512M 字节 （29bit）的外部存储器空间 （映像时忽视 4G 字节地址的高 3bit）。
例如，能通过以下记述检测到将用户模式能使用的 U0 区域 （00000000 ～ 0x7fffffff）映像到外部存储器 （512M）时的重叠目标。

```
-PS_check=00000000-1fffffff,20000000-3fffffff,40000000-
5fffffff,60000000-7fffffff
```

根据此选项的记述，表示在实际存储器空间中地址 00000000、 20000000、 40000000、 60000000 全部被分配在相同的位置。

备　注　　此选项只对 SuperH 族的单片机有效。
在输出格式 （form 选项）为 object、 relocate、 library，或者在处理 absolute 文件的输入时，此选项无效。
有关单片机的地址空间规格，请参照各单片机的硬件使用手册。

非段分割对象的指定

CONTIGUOUS_SECTION

连接程序 < 验证 >[非分割对象的段 :]

格　式　　CONTIGUOUS_SECTION=< 段名 >[,...]

说　明　　在 cpu=stride 有效时，给不分割段而能分配在相同存储器种类的地址区域指定要分配的段。

[例]

```
start=P,PA,PB/100
cpu=ROM=100-1FF,ROM=300-3FF,ROM=500-5FF
cpu=stride
contiguous_section=PA
```

将段 P 分配到地址 100。
在不能将指定 contiguous_section 的段 PA 分配到地址 1FF 的情况下，不分割段 PA 而从地址 300 开始分配。
在不能将未指定 contiguous_section 的段 PB 分配到地址 3FF 的情况下，将段 PB 分割后从地址 500 开始分配。

备　注　　在 cpu 选项的子选项 stride 无效时，此选项无效。

5.2.7　其他选项

表 5.13　其他种类选项一览表

	项目	命令行格式	对话菜单	指定内容
1	结束码	S9	Link/Library <Other> [Miscellaneous options :] [Always output S9 record at the end]	总是输出 S9 记录。
2	堆栈信息文件	STACk	Link/Library <Other> [Miscellaneous options :] [Stack information output]	输出堆栈使用量的信息文件。
3	调试信息的压缩	COmpress NOCOmpress	Link/Library <Other> [Miscellaneous options :] [Compress debug information]	压缩调试信息。 不压缩调试信息。
4	存储器使用量的削减指定	MEMory = [High | Low]	Link/Library <Other> [Miscellaneous options :] [Low memory use during linkage]	指定输入文件加载时的存储器使用量。
5	符号名的变更	REName = <sub>[,...] <sub> : { [< 文件名 >] (< 名称 >=< 名称 >[,...]) | [< 模块名 >] (< 名称 >=< 名称 >[,...]) }	Link/Library <Other> [User defined options :]	更改符号名和段名。
6	符号名的删除	DELete = <sub>[,...] <sub> : { < 模块名 > | [< 文件名 >] (< 名称 >[,...]) }	Link/Library <Other> [User defined options :]	删除符号名和模块名。
7	模块的替换	REPlace = <sub>[,…] <sub> : < 文件 > [(< 模块 >[,…])]	Link/Library <Other> [User defined options :]	替换库文件中的同名模块。
8	模块的抽出	EXTract = < 模块 >[,...]	Link/Library <Other> [User defined options :]	抽出库文件中指定的模块。
9	调试信息的删除	STRip	Link/Library <Other> [User defined options:]	删除绝对文件和库文件的调试信息。
10	消息级	CHange_message =<sub>[,...] <sub>: {Information | Warning | Error} [=< 错误号 > [-< 错误号 >] [,...]]	Link/Library <Other> [User defined options:]	更改消息级。
11	局部符号名的隐含指定	Hide	Link/Library <Other> [User defined options:]	删除局部符号名的信息。
12	段的总容量的显示	Total_size	Link/Library <Other> [Miscellaneous options :] [Displays total section size]	能给标准输出显示连接后的段的总容量。
13	仿真器的信息文件	RTs_file	Link/Library <Other> [Miscellaneous options :] [Rts information output]	输出仿真器的信息文件 （面向 SuperH 族）。

结束码

S9

连接程序 < 其他 >[其他选项 :][将 S9 记录输出到末端]

格　式　　S9

说　明　　即使入口地址超过 0x10000，也将 S9 记录输出到末端。

备　注　　在没有指定 form=stype 时，此选项无效。

堆栈信息文件

STACk

连接程序 < 其他 >[其他选项 :][堆栈信息文件 (sni) 的输出]

格　式　　STACk

说　明　　输出堆栈使用量的信息文件。
文件名为 < 输出文件名 >.sni。

备　注　　在指定 form={object | relocate | library} 或者 strip 时，此选项无效。

调试信息的压缩

COmpress
NOCOmpress

连接程序 < 其他 >[其他选项 :][调试信息的压缩]

格　式　　COmpress
NOCOmpress

说　明　　指定是否压缩调试信息。
当指定 compress 选项时，压缩调试信息。
当指定 nocompress 选项时，不压缩调试信息。
如果压缩调试信息，就能加快调试程序的加载速度。如果指定 nocompress 选项，就能缩短连接时间。
在省略此选项时解释为 nocompress。

备　注　　在指定 form={object | relocate | library | hexadecimal | stype | binary} 或者 strip 选项时，此选项无效。

存储器使用量的削减指定

MEMory

连接程序 < 其他 >[其他选项 :][削减输入文件加载时的存储器使用量]

格　式　　MEMory = [High | Low]

说　明　　指定连接时使用的存储器容量。
当指定 memory=high 选项时，进行正常处理。
当指定 memory=low 选项时，通过将连接时所需的信息进行小单位加载，削减存储器的使用量。由于文件存取频率的增加，所以在存储器使用量不超过实际存储器容量的情况下，比指定 memory=high 选项时的处理速度慢。

在连接大规模的工程时，如果因优化连接编辑程序的存储器使用量超过工作计算机的实际存储器容量而使运行速度变慢，就尝试指定 memory=low 选项。

备　注　　如果指定以下选项，此选项的指定就无效：
同时指定 optimize、compress、delete、rename、map、stack、replace、list、show[={reference | xreference}]
根据输入文件和输出文件的格式，有些组合也可能无效，详细内容请参照“5.2.2 输出选项”的表 5.4。

符号名的变更

REName

连接程序 < 其他 >[用户指定选项 :]

格　式　　REName = < 子选项 >[,…]
< 子选项 > : { [< 文件 >](< 名称 > = < 名称 >[,…])
| [< 模块 >](< 名称 > = < 名称 >[,…])}

说　明　　更改外部符号名和段名。
也能更改特定文件或者特定库内的模块中包含的符号名和段名。
在 C/C++ 变量名的情况下，在程序中的定义名前头附加“_”。
如果更改函数名，就无法保证运行。
如果段和符号中都有指定的名称，就优先符号名。
如果有多个相同的文件名或者模块名，先输入的文件或者模块就优先。

例　　rename=(_sym1=data) ; 将 _sym1 更改为 data。
rename=lib1(P=P1)　; 将库模块 lib1 内的 P 段更改为 P1 段。

备　注　　在指定 extract 或者 strip 时，此选项无效。
在指定 form=absolute 时，不能更改输入库的段名。

符号名的删除

DELete

连接程序 < 其他 >[用户指定选项 :]

格　式　　DELete = < 子选项 >[,…]
< 子选项 > : { [< 文件 >](< 名称 >[,…])
| < 模块 > }

说　明　　删除外部符号名或者库模块。
也能删除特定文件中包含的符号名和模块。
对于 C/C++ 变量名和 C 函数名，在程序中的定义名前头附加 “_”。在 C++ 函数的情况下，用双引号括将程序中含有参数串的定义名括起来指定。但是在参数为 void 时，用 “函数名 ()” 指定。如果有多个相同的文件名，先输入的文件名就优先。
如果此选项指定符号名的删除，就不删除目标而将属性更改为内部符号。

例　　
```
delete=(_sym1)            ; 删除全部文件中的符号名 _sym1。
delete=file1.obj(_sym2)   ; 删除 file1.obj 内的符号名 _sym2。
```

备　注　　在指定 extract 或者 strip 时，此选项无效。

模块的替换

REPlace

连接程序 < 其他 >[用户指定选项 :]

格　式　　REPlace = < 子选项 >[,…]
< 子选项 > : < 文件名 >[(< 模块名 >[,…])]

说　明　　替换库模块。
将指定的文件或者库模块和 library 选项指定的库内同名模块进行替换。

例　　
```
replace=file1.obj         ; 将模块 file1 和模块 file1.obj 进行替换。
replace=lib1.lib(mdl1)    ; 将模块 mdl1 和库文件 lib1.lib 内的模块 mdl1 进行
                          ; 替换。
```

备　注　　在指定 form={object | relocate | absolute | hexadecimal | stype | binary}、extract 或者 strip 时，此选项无效。

模块的抽出

EXTract

连接程序 < 其他 >[用户指定选项 :]

格　式　　EXTract = < 模块名 >[,…]

说　明　　抽出库模块。
从 library 选项指定的库文件中抽出指定的库模块。

例　　extract=file1　　; 抽出模块 file1。

备　注　　在指定 form={absolute | hexadecimal | stype | binary} 或者 strip 时，此选项无效。

调试信息的删除

STRip

连接程序 < 其他 >[用户指定选项 :]

格　式　　STRip

说　明　　删除绝对文件和库文件的调试信息。
在指定 strip 选项时，输入文件和输出文件一一对应。

例　　input=file1.abs file2.abs file3.abs
strip
删除 file1.abs 和 file2.abs 的调试信息，分别输出到 file1.abs、 file2.abs 和 file3.abs。将输出调试信息前的文件备份到 file1.abk、 file2.abk 和 file3.abk。

备　注　　在指定 form={object | relocate | hexadecimal | stype | binary} 时，此选项无效。

消息级

CHange_message

连接程序 < 其他 >[用户指定选项 :]

格　式　　CHange_message = < 子选项 >[,…]
< 子选项 > : < 错误级 >[=< 错误号 >[-< 错误号 >][,…]]
< 错误级 > : {Information | Warning | Error}

说　明　　更改信息、警告和错误级的消息级。
能更改在输出消息时是继续执行还是中止执行。

例　　change_message=warning=2310
将 L2310 更改为警告级，并且在输出 L2310 时继续进行处理。

change_message=error
将全部信息消息和警告消息更改为错误级消息。
只要输出 1 个信息，就中止处理。

局部符号名的隐含指定

Hide

连接程序 < 其他 >[用户指定选项 :]

格　式　　Hide

说　明　　如果指定此选项，就删除输出文件中的局部符号名信息。
因为删除与局部符号有关的名称信息，所以即使通过二进制编辑程序等打开文件，也无法确认局部符号名。完全不影响生成文件的运行。
要隐含指定局部符号名时，必须指定此选项。

隐含对象的符号种类如下：

- 源文件：指定static型限定词的变量名和函数名等
- 源文件：goto语句的标号名
- 汇编源：没有进行外部定义（参照）符号声明的符号名

例　　源文件中此选项功能有效的记述例子如下所示：

```
int g1;
int g2=1;
const int g3=3;
static int s1;            //<--- static 变量名为隐含对象。
static int s2=1;          //<--- static 变量名为隐含对象。
static const int s3=2;    //<--- static 变量名为隐含对象。

static int sub1()         //<--- static 函数名为隐含对象。
{
   static int s1;        //<--- static 变量名为隐含对象。
   int l1;

   s1 = l1; l1 = s1;
   return(l1);
}

int main()
{
   sub1();
   if (g1==1)
      goto L1;
   g2=2;
L1:                       //<--- goto 语句的标号名为隐含对象。
   Return(0);
}
```

备　注　　只有在输出文件格式为 absolute、 relocate、 library 时，此选项才有效。
如果在输入编译和汇编时指定 goptimize 选项的文件或者输出文件格式为 relocate 或者 library，就不能指定此选项。
如果在进行外部变量存取优化的情况下指定此选项，就不能在第一次连接时指定，而只能在第二次连接时指定。
即使指定此选项，也不删除调试信息内的符号名。

段的总容量的显示

Total_size

连接程序 < 其他 >[其他选项 :][段的总容量的画面显示]

格　式　　Total_size

说　明　　此选项在标准输出时显示连接后的段的总容量。
分成以下 3 种段，显示总容量：

- 能执行的程序段
- 程序段以外的ROM区分配段
- RAM区分配段

通过使用此选项，能容易判断分配到 ROM 和 RAM 的段的总容量。

备　注　　在给连接列表显示段的总容量时，需要单独使用 show=total_size 选项。
在具有 ROM 化支持功能 （rom 选项）的对象段的情况下，因为传送源 （ROM）和传送目标（RAM）都使用区域，所以总容量为双方的总容量加上段容量。

仿真器的信息文件

RTs_file

连接程序 < 其他 >[其他选项 :][函数出口信息文件 (rts) 的输出]

格　式　　-RTs_file

说　明　　这是生成用于仿真器的信息以及函数出口信息文件 （.rts 文件）的选项。
必须根据仿真器的使用手册使用此选项，有可能因仿真器的种类而不能使用此选项。
生成文件名为 “< 输出的加载模块名 >.rts” 的函数出口信息文件。例如，如果将 output 选项指定的输出文件名设定为 “test.abs”，就生成文件名为 “test.rts” 的函数出口信息文件。
函数出口信息文件建立在和加载模块相同的目录下。

备　注　　• 在指定 form={object | relocate | library} 时，此选项无效。
• 在输入绝对文件时，此选项无效。
• 必须根据仿真器的使用手册使用此选项，有可能因仿真器的种类而不能使用此选项。

5.2.8 子命令文件选项

表 5.14 子命令文件种类选项一览表

	项目	命令行格式	对话菜单	指定内容
1	子命令文件	SUbcommand = < 文件名 >	Link/Library <Subcommand file> [Use external subcommand file]	通过子命令文件指定选项。

子命令文件

SUbcommand

连接程序 < 子命令文件 >[指定子命令文件]

格　式　　SUbcommand = < 文件名 >

说　明　　通过子命令文件指定选项。
子命令文件的格式如下：

< 选项 > {= | △ } [< 子选项 > [,…]] [△ &] [;< 注释 >]

也能用空格代替 “=” 分开指定选项和子选项。
在 input 选项的情况下，能指定空格来分开子选项。
在子命令文件内，1 行指定 1 个选项。
如果无法在 1 行内记述子选项，就能用 & 继续指定。
不能在子命令文件中指定 subcommand 选项。

例　　命令行的指定：optlnk file1.obj -sub=test.sub file4.obj
子命令的指定：input file2.obj file3.obj ; 这是注释。
library lib1.lib, & ; 指定继续行。
lib2.lib
在将子命令文件指定的选项内容展开到命令行的子命令指定位置后执行。
文件的输入顺序为 file1.obj、file2.obj、file3.obj、file4.obj。

5.2.9 单片机选项

表 5.15 CPU 选项卡的选项一览表

	项目	命令行格式	对话菜单	指定内容
1	SBR 地址的指定	SBr = { <SBR 地址 > | User}	CPU [Specify SBR address :]	指定 8bit 绝对区域的起始地址（面向 H8SX 族）。

8bit 绝对区域地址值的指定

SBr

CPU[SBR 值 :]

格　式　　SBr = { < 地址 > | User }

说　明　　指定 SBR 的地址值。
能通过此选项指定地址值，使用 abs8 区域进行优化。如果此选项指定 user，就抑制对 abs8 区域的优化。

备　注　　此选项只在单片机种类为 H8SX 族时有效。
如果在源程序中或者在指定工具选项时指定多个 SBR 地址，就作为指定 user 进行处理，与此选项的指定无关。

5.2.10 剩余选项

表 5.16 剩余选项一览表

	项目	命令行格式	对话菜单	指定内容
1	版权	LOgo NOLOgo	—	输出版权。 不输出版权。
2	继续指定	END	—	执行已输入的选项串，在结束处理后输入以后的选项串，继续处理。
3	结束指定	EXIt	—	指定选项输入的结束。

版权

LOgo
NOLOgo

无（nologo 总是有效）

格　式　　LOgo
NOLOgo

说　明　　指定是否输出版权。
当指定 logo 选项时，输出版权的显示。
当指定 nologo 选项时，抑制版权显示的输出。
在省略此选项时解释为 logo。

继续处理

END

无

格　式　　END

说　明　　执行在 END 前指定的选项串。在结束连接处理后，输入在 END 后指定的选项串，继续进行连接处理。
不能在命令行指定此选项。

例

```
input=a.obj,b.obj          ; 处理 (1)
start=P,C,D/100,B/8000     ; 处理 (2)
output=a.abs               ; 处理 (3)
end
input=a.abs                ; 处理 (4)
form=stype                 ; 处理 (5)
output=a.mot               ; 处理 (6)
```

执行 (1) ～ (3) 的处理，输出 a.abs。
此后，执行 (4) ～ (6) 的处理并且输出 a.mot。

结束处理

EXIt

无

格　式　　EXIt

说　明　　指定由此选项指定的结束。
不能在命令行指定此选项。

例　　命令行的指定：optlnk -sub=test.sub -nodebug

```
test.sub:          input=a.obj,b.obj        ;处理(1)
                   start=P,C,D/100,B/8000   ;处理(2)
                   output=a.abs             ;处理(3)
                   exit
```

执行(1)～(3)的处理，输出a.abs。
在执行Exit后，命令行指定的nodebug选项无效。

6. 环境变量

6.1 环境变量一览表

环境变量一览表如表 6.1 所示。

表 6.1 环境变量

	环境变量	说明	省略设定时的解释
1	path	指定执行文件的保存目录。	不能省略。
2	BIN_RX	指定 ccrx 的保存目录。	＜ ccrx 的保存目录＞ 在使用 lbgrx 命令时，不能省略。
3	CPU_RX	指定 CPU 种类。 <CPU 种类 > RX600	当有指定 cpu 选项时，可以省略。 当没有指定 cpu 选项时，不能省略。 在省略时，不设定值。
4	INC_RX	指定编译程序的 include 文件的保存目录。	＜ ccrx 的保存目录＞ \..\include
5	INC_RXA	指定汇编程序的 include 文件的保存目录。	在省略时，不设定值。
6	TMP_RX	指定要建立临时文件的目录。	在使用 ccrx 命令时，为 %TEMP%。
7	HLNK_LIBRARY1 HLNK_LIBRARY2 HLNK_LIBRARY3	指定优化连接编辑程序要使用的默认库名。 优先连接 library 选项指定的库。此后，如果有未解决的符号，就按照 1、 2、 3 的顺序检索默认库。	在省略时，不设定值。
8	HLNK_TMP	指定优化连接编辑程序建立临时文件的文件夹。如果没有指定此环境变量，就在当前文件夹中建立临时文件。	在省略时，不设定值。
9	HLNK_DIR	指定优化连接编辑程序的输入文件的保存文件夹。 按照当前文件夹、 HLNK_DIR 指定文件夹的顺序，检索 input 选项和 library 选项指定的文件。 但是，只在当前文件夹中检索用通配符指定的文件。	在省略时，不设定值。

如果不指定 cpu 选项，就必须设定 CPU_RX。如果将 RX600 以外的系列指定为 CPU 种类，就发生错误。

如果通过 INC_RX、 INC_RXA、 HLNK_LIBRARY1、 HLNK_LIBRARY2、 HLNK_LIBRARY3 和 HLNK_DIR 指定多个目录，就必须用 “;”（分号）分开指定。

在执行 ccrx 命令时，如果有已设定的 BIN_RX、 INC_RX 和 TMP_RX 环境变量值，就使用该值，否则就使用省略设定时的解释值。

能通过执行在安装时建立的批文件 setccrx.bat，简单地设定这些环境变量。 setccrx.bat 保存在＜ High-performance Embedded Workshop 的保存目录＞ \Tools\Renesas\RX\1_0_0 或者＜ ccrx 的保存目录＞ \..。

6.2 预定义宏

根据选项的指定和版本，定义以下预定义宏：

表 6.2 编译程序的预定义宏

	选项		预定义宏
1	cpu=rx600	#define __RX600	1
2	endian=big	#define __BIG	1
	endian=little	#define __LIT	1
3	dbl_size=4	#define __DBL4	1
	dbl_size=8	#define __DBL8	1
4	int_to_short	#define __ INT_SHORT	1
5	signed_char	#define __SCHAR	1
	unsigned_char	#define __UCHAR	1
6	signed_bitfield	#define __SBIT	1
	unsigned_bitfield	#define __UBIT	1
7	round=zero	#define __ROZ	1
	round=nearest	#define __RON	1
8	denormalize=off	#define __DOFF	1
	denormalize=on	#define __DON	1
9	bit_order=left	#define __BITLEFT	1
	bit_order=right	#define __BITRIGHT	1
10	auto_enum	#define __AUTO_ENUM	1
11	library=function	#define __FUNCTION_LIB	1
	library=intrinsic	#define __INTRINSIC_LIB	1
12	fpu	#define __FPU	1
13	—	#define __RENESAS__*1	1
14	—	#define __RENESAS_VERSION__*1	0x*AABBCC*00*2
15	—	#define __RX*1	1

【注】 *1 与选项无关，总是被定义。

*2 在 V.AA.BB.CC 版本的情况下， __RENESAS_VERSION__ 的值为 0xAABBCC00。
例）在V.1.00.00版本的情况下，为#define __RENESAS_VERSION__ 0x01000000。

表 6.3 汇编程序的预定义宏

	选项		预定义宏
1	cpu=rx600	__RX600	.DEFINE 1
2	endian=big	__BIG	.DEFINE 1
	endian=little	__LITTLE	.DEFINE 1
3	—	__RENESAS_VERSION__*1	.DEFINE *AABBCC*00*2
4	—	__RX*1	.DEFINE 1

【注】 *1 与选项无关，总是被定义。

*2 在 V.AA.BB.CC 版本的情况下， __RENESAS_VERSION__ 的值为 AABBCC00H。
例）在 V.1.00.00 版本的情况下，为 __RENESAS_VERSION__ .DEFINE 01000000H。

7. 文件规格

7.1 文件名的命名方法

如果在指定文件名时省略扩展名，就使用附加标准文件扩展名的文件名。综合开发环境中使用的标准文件扩展名如表 7.1 所示。

表 7.1 综合开发环境中使用的标准文件扩展名

No.	扩展名	含义
1	c	C 源程序文件
2	cpp、 cc、 cp	C++ 源程序文件
3	h	include 文件
4	p	用于 C 程序的预处理器展开文件
5	pp	用于 C++ 程序的预处理器展开文件
6	src	汇编源程序文件
7	lst	用于汇编程序的列表文件
8	obj	可再定位目标程序文件
9	abs	绝对装入模块文件
10	map	连接列表文件
11	lib	库文件
12	lbp	库列表文件
13	mot	Motorola S 格式
14	hex	Intel （扩展） HEX 格式
15	bin	二进制文件
16	sni	堆栈信息文件
17	pro	配置信息文件
18	dbg	调试信息文件
19	rti	包含由扩展名为 td 的文件指定定义的目标文件。
20	cal	调用信息文件
21	bls	外部符号分配信息文件

因为以 rti_ 开头的文件名为系统保留名，所以不能使用。

暂时输出到 tpldir 文件夹下的各文件扩展名如表 7.2 所示。

表 7.2 tpldir 文件夹的输出文件

No.	扩展名	含义
1	td	tentative 定义的变量信息文件
2	ti	模板信息文件
3	pi	参数信息文件
4	ii	示例信息文件

7.2 源列表的参照方法

7.2.1 源列表的结构

源列表文件显示编译结果和汇编结果的信息。
源列表的结构和内容如表 7.3 所示。

表 7.3 源列表的结构和内容

No.	显示在列表文件中的信息	内容	子选项 *	省略 show 选项时
1	源信息	对应汇编源，显示 C/C++ 语言源。	show=source	不输出
2	目标信息	目标程序的机器语言和汇编源码	无	输出
3	统计信息	错误的总数、源程序的行数、段容量、符号数	无	输出
4	命令指定的信息	显示用命令指定的文件名和选项。	无	输出

【注】 * 在指定 list 选项时有效。

7.2.2 源信息

通过指定 show=source 选项，以填充目标信息的形式输出源列表信息，输出例子请参照 “7.2.3 目标信息”。

7.2.3 目标信息

目标信息的输出例子如图 7.1 所示。

```
* RX FAMILY ASSEMBLER V.1.00.00 *  SOURCE LIST  Sat May 16 11:56:15 2009

LOC.     OBJ.              0XMDA SOURCE STATEMENT
(1)      (2)                (3)      (4)

                                    ;C LABEL    INSTRUCTION OPERAND    COMMENT
                                       (5)          (6)                 (7)
                                    ;LineNo. C-SOURCE STATEMENT
                                       (8)          (9)

                                                .SECTION    P,CODE
                                    ;       1 #include      "include.h"
                                    ;       1 extern int    x;
                                    ;       2 extern int    y = 1;
                                    ;       2 int func01(int);
                                    ;       3 int func03(int);
                                    ;       4
                                    ;       5 int func02(int z)
                                                .glb        _func02
00000000                            _func02:                            ; function: func02
                                                .STACK      _func02=8
00000000 7EA6                                   PUSH.L      R6
00000002                            L10:
                                                .LINE       "D:\RXC\work\list\now\sample.c",7
                                    ;       6 {
                                    ;       7       x = func01(z);
00000002 EF16                                   MOV.L       R1,R6
00000004 05rrrrrr             A                 BSR         _func01
00000008 FB42rrrrrrrr                           MOV.L       #_x,R4
0000000E E341                                   MOV.L       R1,[R4]
                                                .LINE       "D:\RXC\work\list\now\sample.c",8
                                    ;       8       if (z == 2) {
00000010 6126                                   CMP         #02H,R6
00000012 18                   S                 BNE         L12
00000013                            L11:
                                                .LINE       "D:\RXC\work\list\now\sample.c",9
                                    ;       9               x++;
00000013 711501                                 ADD         #01H,R1,R5
00000016 E345                                   MOV.L       R5,[R4]
00000018 2E11                 B                 BRA         L13
0000001A                            L12:
                                                .LINE       "D:\RXC\work\list\now\sample.c",11
                                    ;      10       } else {
                                    ;      11               x = func03(x + 2);
0000001A 6221                                   ADD         #02H,R1
0000001C 391200               W                 BSR         _func03
0000001F FB42rrrrrrrr                           MOV.L       #_x,R4
00000025 E341                                   MOV.L       R1,[R4]
00000027 EF15                                   MOV.L       R1,R5
00000029                            L13:
                                                .LINE       "D:\RXC\work\list\now\sample.c",13
                                    ;      12       }
                                    ;      13       return x;
00000029 EF51                                   MOV.L       R5,R1
0000002B 3F6601                                 RTSD        #04H,R6-R6
                                                .LINE       "D:\RXC\work\list\now\sample.c",16
                                    ;      14 }
                                    ;      15
                                    ;      16 int func03(int p)
                                                .glb        _func03
0000002E                            _func03:                            ; function: func03
                                                .STACK      _func03=4
0000002E                            L14:
                                                .LINE       "D:\RXC\work\list\now\sample.c",18
                                    ;      17 {
                                    ;      18       return p+1;
0000002E 6211                                   ADD         #01H,R1
00000030 02                                     RTS
                                    ;      19 }
                                                .glb        _x
                                                .glb        _func01
                                                .SECTION    D,ROMDATA,ALIGN=4
                                                .glb        _y
00000000                            _y:                                 ; static: y
00000000 01000000                               .lword       00000001H
                                                .END
```

图 7.1　目标信息的输出例子

(1) 定位信息 （LOC.）

输出在汇编时能决定范围的目标码的定位地址。

(2) 目标码信息 （OBJ.）

输出对应助记符的目标码。

(3) 行信息 （0XMDA）

输出编译程序处理源码后的结果信息，各符号的含义如下所示：

表 7.4 汇编源码的行信息

0	X	M	D	A	内 容
0-30					表示 include 文件的嵌套层。
	X				在指定 show=conditions 时，显示在进行条件汇编时不满足条件的行。
		M			在指定 show=expansions 时，显示宏指令的展开行。
		D			在指定 show=definitions 时，显示宏指令的定义行。
			S		表示选择了转移距离说明符 S。
			B		表示选择了转移距离说明符 B。
			W		表示选择了转移距离说明符 W。
			A		表示选择了转移距离说明符 A。
				*	表示给条件转移指令选择了代替指令。

(4) 源列表信息 （SOURCE STATEMENT）

显示汇编源文件的内容。

(5) 标号信息 （LABEL）

(6) 汇编程序指令串 （INSTRUCTION OPERAND）

显示编译程序输出的汇编程序指令串。

(7) 对应汇编源程序的注释 （COMMENT）

(8) C/C++ 源行号 （CLineNo.）

(9) C/C++ 源码 （C-SOURCE STATEMENT）

如果指定 show=source 选项，就输出 C/C++ 源码。

7.2.4 统计信息

统计信息的输出例子如图 7.2 所示。

```
Information List  (1)

TOTAL ERROR(S)    00000
TOTAL WARNING(S)  00000
TOTAL LINE(S)     00071   LINES

Section List  (2)

Attr          Size                Name
CODE      0000000047(0000002FH)   P
ROMDATA   0000000004(00000004H)   D
```

图 7.2 统计信息的输出例子

(1) 错误消息数、警告消息数和源行总数
(2) 段信息 （段属性、容量和段名）

7.2.5 编译程序的命令指定信息

显示在启动编译程序时用命令指定的文件名和选项。将编译程序的命令指定信息输出到列表文件的开头，命令指定信息的输出例子如图 7.3 所示。

```
                        ; *** CPU TYPE ***  (1)

                        ; -CPU=RX600

                        ; *** COMMAND PARAMETER ***  (2)

                        ; -output=src=C:\tmp\elp1894\sample.src
                        ; -nologo
                        ; -show=source
                        ; sample.c
```

图 7.3 命令指定信息

(1) 所选的单片机
(2) 传递给编译程序的文件名和选项

7.2.6 汇编程序的命令指定信息

显示在启动汇编程序时用命令指定的文件名和选项。将汇编程序的命令指定信息输出到列表文件的最后，命令指定信息的输出例子如图 7.4 所示。

```
Cpu Type    (1)

-CPU=RX600

Command Parameter    (2)

-output=sample.obj
-nologo
-listfile=sample.lst
```

图 7.4 命令指定信息

(1) 通过汇编程序选择的单片机
(2) 传递给汇编程序的文件名和选项

7.3 连接列表的参照方法

说明优化连接编辑程序输出的连接列表的内容和格式。

7.3.1 连接列表的结构

连接列表的结构和内容如**表 7.5** 所示。

表 7.5 连接列表的结构和内容

No.	显示在列表文件中的信息	内容	指定 show 选项 * 时	省略 show 选项时
1	选项信息	显示用命令行和子命令指定的选项列。	无	输出
2	错误信息	显示错误信息	无	输出
3	连接映像信息	显示段名、起始地址 / 结束地址、容量和种类。	无	输出
4	符号信息	按照地址顺序显示静态定义符号名、地址、长度和种类。 如果指定 show=reference，也显示各符号的参照次数以及是否进行优化。	show=symbol show=reference	不输出 不输出
5	符号删除优化信息	显示通过优化而删除的符号。	show=symbol	不输出
6	对照表信息	显示符号的参照信息。	show=xreference	不输出
7	段的总容量	显示 RAM、 ROM 和程序段的总容量。	show=total_size	不输出
8	向量信息	显示向量号和地址的信息。	show=vector	不输出
9	CRC 信息	显示 CRC 的运算结果和输出位置的地址	无	在指定 CRC 选项时，总是输出。

【注】 * show 选项在指定 list 选项时有效。

7.3.2 选项信息

输出用命令行和子命令文件指定的选项列，选项信息的输出例子如**图 7.5** 所示 （指定 optlnk -sub=test.sub -list -show 时）。

```
(testsub的内容)
INPUT test.obj
```

```
*** Options ***

-sub=test.sub
INPUT test.obj (2)      } (1)
-list
-show
```

图 7.5 选项信息的输出例子 （连接列表）

(1) 按照指定顺序输出用命令行和子命令指定的选项列。
(2) 这是子命令文件test.sub内的子命令。

7.3.3 错误信息

输出错误信息，错误信息的输出例子如图 7.6 所示。

```
*** Error Information ***

** L2310 (E) Undefined external symbol “strcmp” referred to in “test.obj”  (1)
```

图 7.6 错误信息的输出例子 （连接列表）

(1) 输出错误信息。

7.3.4 连接映像信息

按照地址顺序输出各段的起始地址 / 结束地址、容量和种类，连接映像信息的输出例子如图 7.7 所示。

```
*** Mapping List ***

SECTION                    START      END        SIZE   ALIGN
  (1)                        (2)        (3)        (4)     (5)

P
                         00001000   00001000          1       1
C
                         00001004   00001007          4       4
D_2
                         00001008   000014dd        4d6       2
B_2
                         000014de   000050b3        3bd6      2
```

图 7.7 连接映像信息的输出例子 （连接列表）

(1) 显示段名。
(2) 显示起始地址。
(3) 显示结束地址。
(4) 显示段容量。
(5) 显示段的调整数。

7.3.5 符号信息

如果指定 show=symbol，就按照地址顺序输出外部定义符号或者静态内部定义符号的地址、长度和种类。如果指定 show=reference，也输出各符号的参照次数以及是否进行优化。符号信息的输出例子如图 7.8 所示。

```
*** Symbol List ***

SECTION=(1)
FILE=(2)          START          END            SIZE
                    (3)          (4)             (5)
  SYMBOL          ADDR          SIZE           INFO            COUNTS   OPT
  (6)              (7)          (8)             (9)             (10)    (11)

SECTION=P
FILE=test.obj
                  00000000       00000428          428
  _main
                  00000000              2       func ,g              0
  _malloc
                  00000000             32       func ,l              0
FILE=mvn3
                  00000428       00000490           68
  $MVN#3
                  00000428              0       none ,g              0
```

图 7.8 符号信息的输出例子 （连接列表）

(1) 显示段名。
(2) 显示文件名。
(3) 显示(2)的文件包含的相应段的起始地址。
(4) 显示(2)的文件包含的相应段的结束地址。
(5) 显示(2)的文件包含的相应段的容量。
(6) 显示符号名。
(7) 显示符号地址。
(8) 显示符号长度。
(9) 符号种类如下所示：

数据种类：	func	・・・・・・	函数名
	data	・・・・・・	变量名
	entry	・・・・・・	入口函数名
	none	・・・・・・	未设定 （标号、汇编程序符号）
声明种类：	g	・・・・・・	外部定义
	l	・・・・・・	内部定义

(10) 显示符号的参照次数，只在指定show=reference时显示符号的参照次数。在不显示参照次数时，显示*。
(11) 优化的有无如下所示：

ch	・・・・・・・・	通过优化而更改的符号
cr	・・・・・・・・	通过优化而生成的符号
mv	・・・・・・・・	通过优化而移动的符号

7.3.6 符号删除优化信息

输出通过符号删除优化 （optimize=symbol_delete）而删除的符号长度和种类。符号删除优化信息的输出例子如图 7.9 所示。

```
*** Delete Symbols ***

SYMBOL                          SIZE    INFO
  (1)                           (2)     (3)
  _Version
                                   4    data ,g
```

图 7.9 符号删除信息的输出例子 （连接列表）

(1) 显示要删除的符号名。
(2) 显示要删除的符号长度。
(3) 删除的符号种类如下所示：

数据种类：	func	・・・・・・・・	函数名
	data	・・・・・・・・	变量名
声明种类：	g	・・・・・・・・	外部定义
	l	・・・・・・・・	内部定义

7.3.7 对照表信息

输出符号的参照信息（对照表信息），对照表信息的输出例子如图 7.10 所示。

```
 *** Cross Reference List ***

No    Unit Name  Global.Symbol   Location    External Information
(1)      (2)          (3)          (4)              (5)
0001  a
      SECTION=P  _func
                                 00000100
                 _func1
                                 00000116
                 _main
                                 0000012c
                 _g
                                 00000136
      SECTION=B
                 _a
                                 00000190    0001(00000140:P)
                                             0002(00000178:P)
                                             0003(0000018c:P)
0002  b
      SECTION=P
                 _func01
                                 00000154    0001(00000148:P)
                 _func02
                                 00000166    0001(00000150:P)
0003  c
      SECTION=P
                 _func03
                                 00000184
```

图 7.10 对照表信息的输出例子（连接列表）

(1) 这是Unit号，以目标为单位的识别号。
(2) 这是目标名，为连接时的输入指定顺序。
(3) 这是符号名，按照各段分配地址的升序进行输出。
(4) 这是符号的分配地址。
在指定form=rel时，符号的分配地址是从段的起始地址开始的相对值。
(5) 表示正在参照的外部符号的地址。
输出格式如下：
<Unit号>(<地址 or 段内的偏移量>:<段名>)

7.3.8 段的总容量

输出 ROM 段、 RAM 段和程序段的总容量，总容量的输出例子如图 7.11 所示。

```
*** Total Section Size ***

RAMDATA SECTION:        00000660 Byte(s)
(1)
ROMDATA SECTION:        00000174 Byte(s)
(2)
PROGRAM SECTION:        000016d6 Byte(s)
(3)
```

图 7.11 段的总容量的输出例子 （连接列表）

(1) 这是RAM数据段的总容量。
(2) 这是ROM数据段的总容量。
(3) 这是程序段的总容量。

7.3.9 向量信息

显示可变向量表的内容，段的总容量的输出例子如图 7.12 所示。

```
*** Variable Vector Table List ***

NO.     SYMBOL/ADDRESS
(1)     (2)
  0     $fdummy
  1     $fa
  2     00ff8800
  3     $fdummy
    :
  <省略>
```

图 7.12 段的总容量的输出例子 （连接列表）

(1) 这是向量号。
(2) 显示符号。在没有定义符号的情况下，用地址显示符号。

7.3.10 CRC 信息

在指定 CRC 选项时，输出 CRC 的运算结果和输出位置的地址。

```
*** CRC Code ***

CODE    : cb0b
(1)
ADDRESS : 00007ffe
(2)
```

图 7.13 段的总容量的输出例子 （连接列表）

(1) 这是CRC运算结果。
(2) 这是CRC运算结果的输出位置的地址。

7.4 库列表的参照方法

本节说明优化连接编辑程序输出的库列表的内容和格式。

7.4.1 库列表的结构

库列表的结构和内容如**表 7.6** 所示。

表 7.6 库列表的结构和内容

No.	列表的建立	内容	子选项 *	省略 show 选项时
1	选项信息	显示用命令行和子命令指定的选项列。	—	输出
2	错误信息	显示错误信息。	—	输出
3	库信息	显示库信息。	—	输出
4	库内模块、段、符号信息	显示库内模块。	—	输出
		当指定 show=symbol 时，也显示模块内的符号名一览表。	show=symbol	不输出
		当指定 show=section 时，也显示各模块内的段名和符号名一览表。	show=section	不输出

【注】 * 全部选项在指定 list 选项时都有效。

7.4.2 选项信息

输出用命令行或者子命令文件指定的选项列，选项信息的输出例子如图 7.14 所示（指定 optlnk -sub=test.sub -list -show 时）。

```
(test.sub的内容)
form    library
in      adhry.obj
output  test.lib
```

```
*** Options ***

-sub=test.sub
form    library
in      adhry.obj      (2)
output  test.lib
-list                  (1)
-show
```

图 7.14 选项信息的输出例子 （库列表）

(1) 按照指定顺序输出用命令行或者子命令指定的选项列。
(2) 这是子命令文件test.sub内的子命令。

7.4.3 错误信息

输出错误和警告等信息，错误信息的输出例子如图 7.15 所示。

```
*** Error Information ***

** L1200 (W) Backed up file “main.lib” into “main.lbk”    (1)
```

图 7.15 错误信息的输出例子 （库列表）

(1) 输出警告信息。

7.4.4 库信息

输出库的种类，库信息的输出例子如图 7.16 所示。

```
*** Library Information ***

LIBRARY NAME=test.lib   (1)
CPU=SuperH              (2)
ENDIAN=Big              (3)
ATTRIBUTE=system        (4)
NUMBER OF MODULE=1      (5)
```

图 7.16 库信息的输出例子 （库列表）

(1) 显示库。
(2) 显示单片机名。
(3) 显示字节序种类。
(4) 显示库文件的属性是系统库还是用户库。
(5) 显示库内的模块数。

7.4.5 库内的模块、段和符号信息

输出库内的模块一览表。

当指定 show=symbol 时，输出模块内的符号名一览表；当指定 show=section 时，输出模块内的段名和符号名一览表。

库内的模块、段和符号信息的输出例子如图 7.17 所示。

```
*** Library List ***

MODULE          LAST UPDATE
 (1)                (2)
  SECTION
    (3)
    SYMBOL
      (4)
adhry
                29-Feb-2000 12:34:56
  P
   _main
   _Proc0
   _Proc1
  C
  D
   _Version
  B
   _IntGlob
   _CharGlob
```

图 7.17 库内的模块、段和符号信息的输出例子 （库列表）

(1) 显示模块名。
(2) 显示模块的注册日期。如果模块被更新，就显示最新的更新日期。
(3) 显示模块内的段名。
(4) 显示段内的符号。

8. 编程

8.1 程序结构

8.1.1 段

段由汇编程序输出的可再定位文件的执行指令以及数据的各区域构成，是分配存储器的最小单位，具有以下性质：

- 段属性
  code 保存执行指令。
  data 保存能更改的数据。
  romdata 保存固定数据。
- 方式种类
  相对地址方式 …… 这是能通过优化连接编辑程序重新分配的段。
  绝对地址方式 …… 这是由地址决定的段，不能通过优化连接编辑程序重新分配。
- 初始值
  表示在开始执行程序时有无初始值。有初始值的数据和没有初始值的数据不能在同一个段内。只要有1个初始值，就将没有初始值的区域初始化为0。
- 写操作
  表示是否能在执行程序时进行写操作。
- 调整数
  这是用于调整段的分配地址的值。通过优化连接编辑程序将各段的分配地址分别调整为调整数的倍数。

8.1.2 C/C++ 程序的段

C/C++ 程序和标准库使用的存储区种类与段的对应如表 8.1 所示。

表 8.1 存储区的种类及其性质的概要

No.	名称	段		方式种类	初始值	调整数	内容
		名称	属性		写操作		
1	程序区	P*1	code	相对	有 不能	1byte	保存机器语言。
2	常数区	C*1*2	romdata	相对	有 不能	4byte	保存 const 型数据。
		C_2*1*2	romdata	相对	有 不能	2byte	
		C_1*1*2	romdata	相对	有 不能	1byte	
3	初始化数据区	D*1*2	romdata	相对	有 能	4byte	保存有初始值的数据。
		D_2*1*2	romdata	相对	有 能	2byte	
		D_1*1*2	romdata	相对	有 能	1byte	
4	未初始化数据区	B*1*2	data	相对	无 能	4byte	保存没有初始值的数据。
		B_2*1*2	data	相对	无 能	2byte	
		B_1*1*2	data	相对	无 能	1byte	
5	switch 语句转移表区	W*1	romdata	相对	有 不能	4byte	保存 switch 语句的转移表。
		W_2*1	romdata	相对	有 不能	2byte	
		W_1*1	romdata	相对	有 不能	1byte	
6	C++ 初始处理 / 后处理数据区	C$INIT	romdata	相对	有 不能	4byte	对于全局类目标，保存被调用的构造函数和析构函数的地址。
7	C++ 虚拟函数表区	C$VTBL	romdata	相对	有 不能	4byte	保存在类声明中有虚拟函数时用于调用虚拟函数的数据。
8	用户堆栈区	SU	data	相对	无 能	4byte	执行程序所需的区域。
9	中断堆栈区	SI	data	相对	无 能	4byte	执行程序所需的区域。
10	堆区	—	—	相对	无 能	—	库函数 malloc、 realloc、 calloc、 new 使用的区域。

No.	名称	段		方式种类	初始值	调整数	内容
		名称	属性		写操作		
11	绝对地址变量区	$ADDR_<section>_<address> *3	data	绝对	有 / 无 能 / 不能 *4	—	保存 #pragma address 指定的变量。
12	可变向量区	C$VECT	romdata	相对	无 能	4byte	可变向量表

【注】 *1 能通过 section 选项或者扩展名 #pragma section 转换段名。但是，字符串的字面常量等部分数据不受 #pragma section 的影响，详细内容请参照 #pragma section 的详细说明。

*2 在转换段名时，也通过指定调整数为 4 的段，更改调整数为 1 或者 2 的段名。
当通过#pragma endian指定和endian选项不同的字节序时，如果是#pragma endian big，就生成在段名后附加 “_B”的专用段，保存该数据；如果是#pragma endian little，就生成在段名后附加 “_L”的专用段，保存该数据。但是，字符串的字面常量等部分数据不受#pragma endian的影响，详细内容请参照#pragma endian的详细说明。

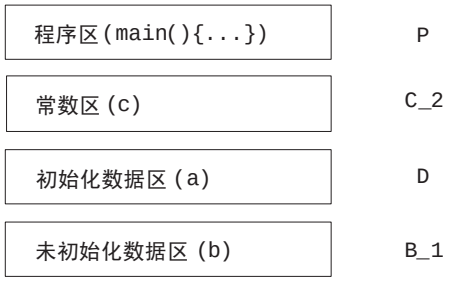
*3 <section> 是 C、 D、 B 的段名， <address> 是绝对地址值 （16 进制数）。

*4 初始值和写操作取决于 <section> 的属性。

例 1. 用程序例子表示 C 程序和编译程序生成段的对应。

```
int a=1;
char b;
const short c=0;
void main (){
  ...
}
```

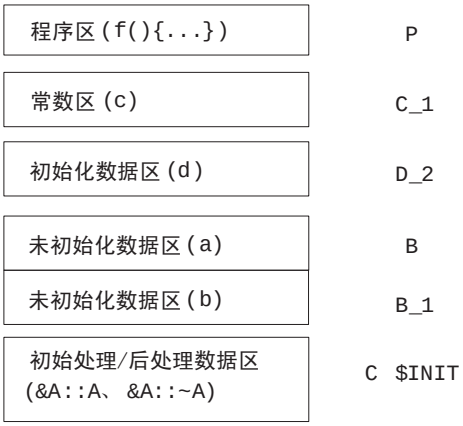
C程序



例 2. 用程序例子表示 C++ 程序和编译程序生成段的对应。

```
class A{
  int m;
  A(int p);
  ~A();
};
A a(1);
char b;
extern const char c='a';
short d=1;
void f(){...}
```

C++程序



8.1.3 汇编程序的段

在汇编程序中，用 .SECTION 控制指令声明段的开始和属性，用 .ORG 控制指令声明段的方式种类。
有关各控制指令的详细内容，请参照 “10.3 汇编程序控制指令的记述方法”。

例：汇编程序的段声明例子如下：

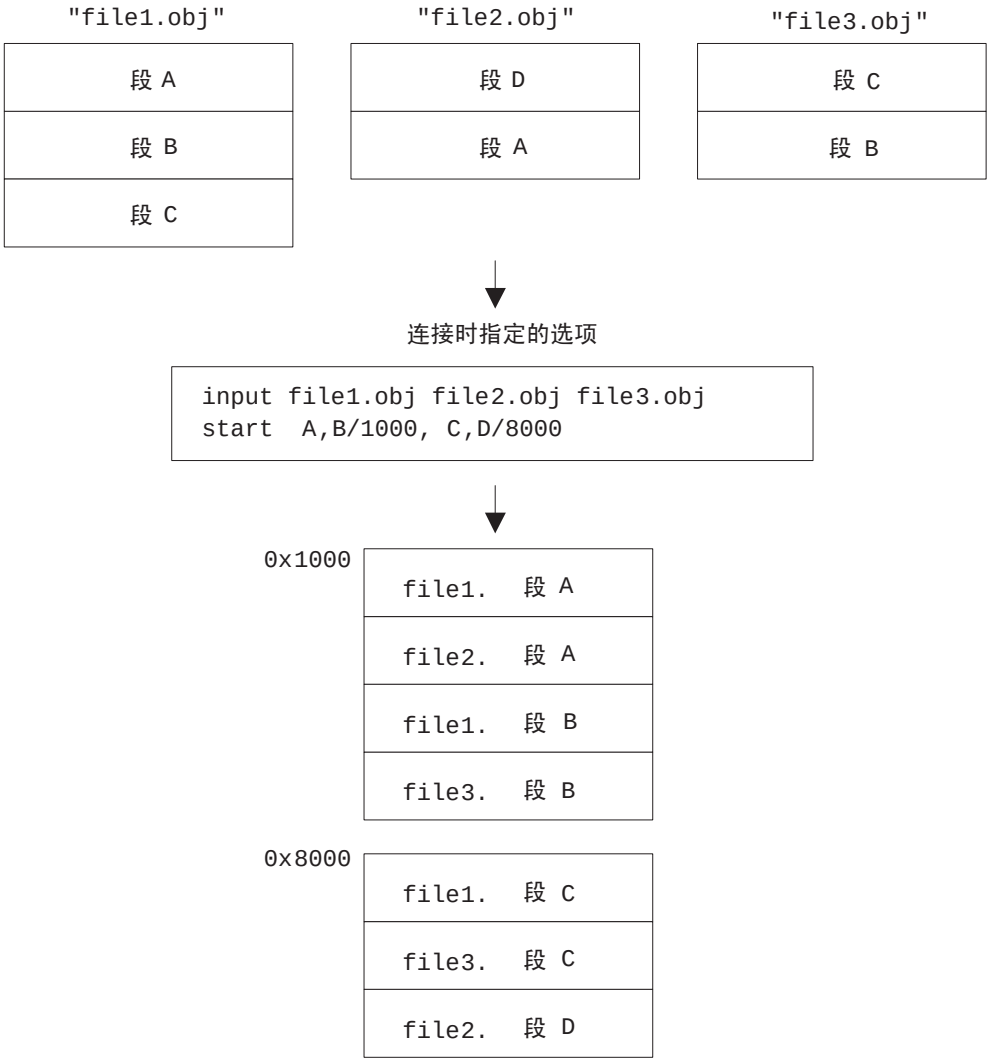
```
            .SECTION        A,CODE,ALIGN=4      ;(1)

START:
            MOV.L           #CONST,R4
            MOV.L           [R4],R5
            ADD             #10,R5,R3
            MOV.L           #100,R4
            MOV.L           #ARRAY,R5
LOOP:
            MOV.L           R3,[R5+]
            SUB             #1,R4
            CMP             #0,R4
            BNE             LOOP
EXIT:
            RTS

;
            .SECTION        B,ROMDATA           ;(2)
            .ORG            02000H
            .glb            CONST
CONST:
            .LWORD          05H
;
            .SECTION        C,DATA,ALIGN=4      ;(3)
            .glb            BASE
BASE:
            .blkl           100
            .END
```

(1) 声明段名为 A、调整数为 4 的相对地址方式的 code 段。
(2) 声明段名为 B、分配地址为 2000H 的绝对地址方式的 romdata 段。
(3) 声明段名为 C、调整数为 4 的相对地址方式的 stack 段。

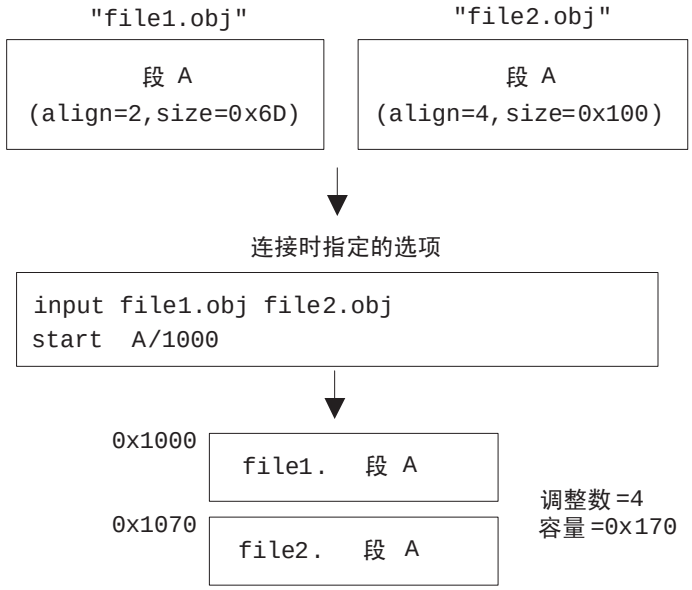
8.1.4 段的连接

通过优化连接编辑程序连接输入可再定位文件内的同一个段，并且分配到 start 选项指定的地址。

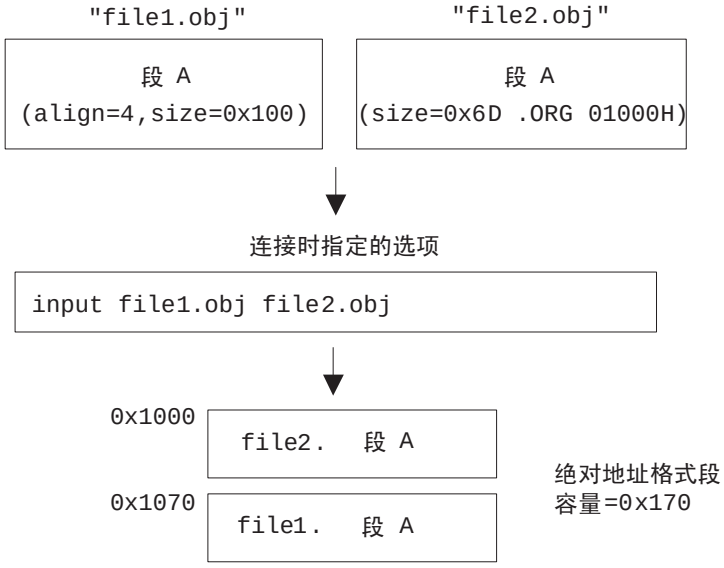
1. 按照文件的输入顺序连续分配不同文件的同名段。



2. 在调整后连接调整数不同的同名段，段的调整数以大的为准。

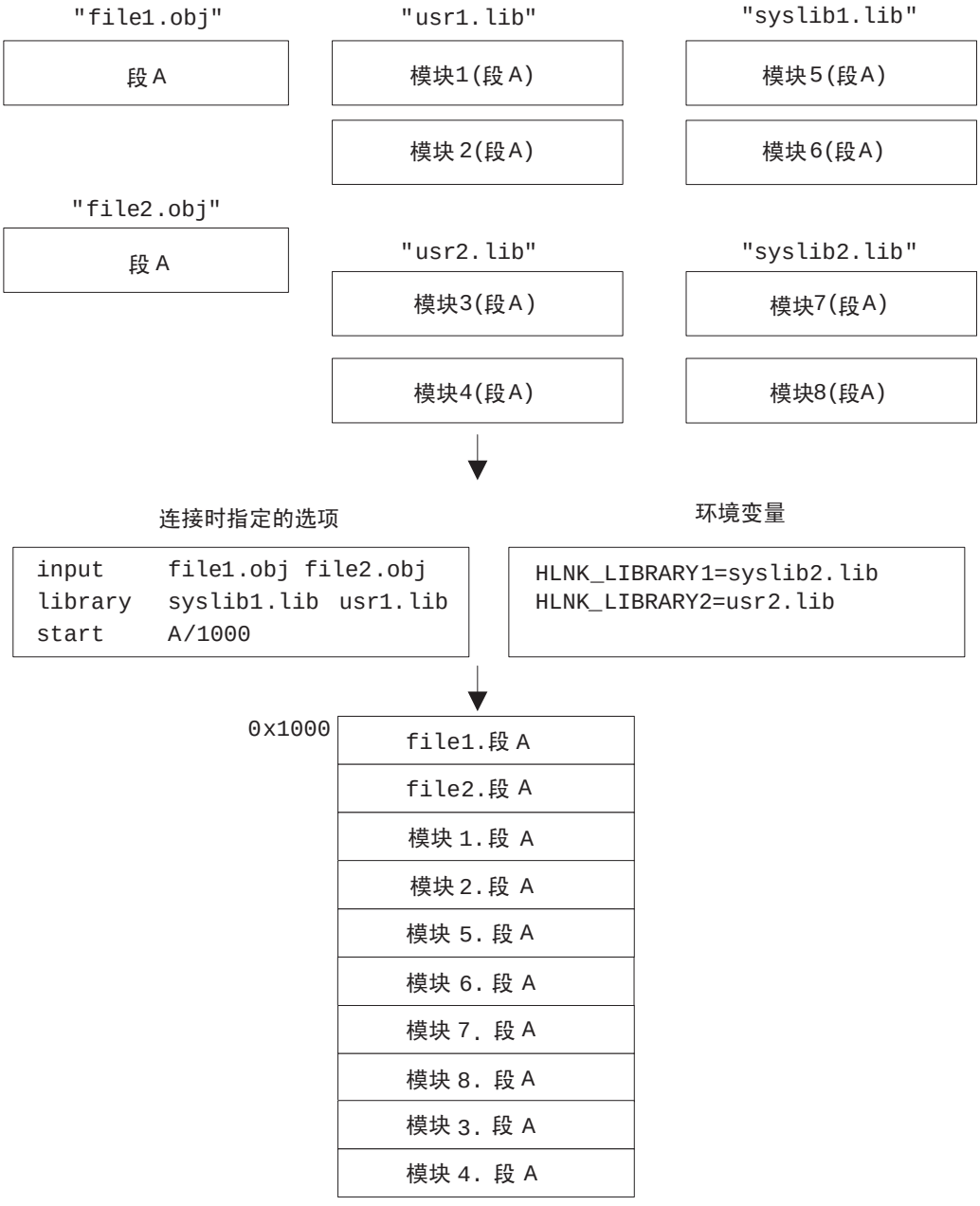


3. 在同名段有绝对地址方式和相对地址方式时，将相对地址方式的段连接到绝对地址方式段的后面。



4. 按照从高到低的优先级顺序，同名段的连接顺序规则如下：

- input选项或者命令行的输入文件指定顺序
- library选项的用户库指定顺序和库内模块的输入顺序
- library选项的系统库指定顺序和库内模块的输入顺序
- 环境变量 （HLNK_LIBRARY1～3）的库指定顺序和库内模块的输入顺序



8.2 函数调用的规则

本章节说明在调用函数时编译程序的寄存器和堆栈区的使用规则。在 C/C++ 程序和汇编程序之间相互调用函数时，需要遵守这些规则编制汇编程序。

- 有关堆栈的规则
- 有关寄存器的规则
- 有关参数的设定和参照的规则
- 有关返回值的设定和参照的规则
- 外部名的相互参照方法

8.2.1 有关堆栈的规则

(1) 堆栈指针

不能将有效数据保存到低于堆栈指针指向地址 （地址 0 的方向）的堆栈区，否则保存的数据有可能被中断处理破坏。

(2) 堆栈帧的分配和释放

在调用函数时 （在刚执行 JSR 或者 BSR 指令后），堆栈指针指向调用侧函数所用堆栈的最低位地址。调用侧函数分配和设定高于此堆栈指针指向的区域地址的数据。

在函数返回时，先释放由被调用侧函数确保的区域，然后用通常的 RTS 指令返回到调用侧函数。调用侧函数释放高于此区域地址的区域 （返回值地址和参数区）。

函数调用后的堆栈帧状态的说明如图 8.1 所示。

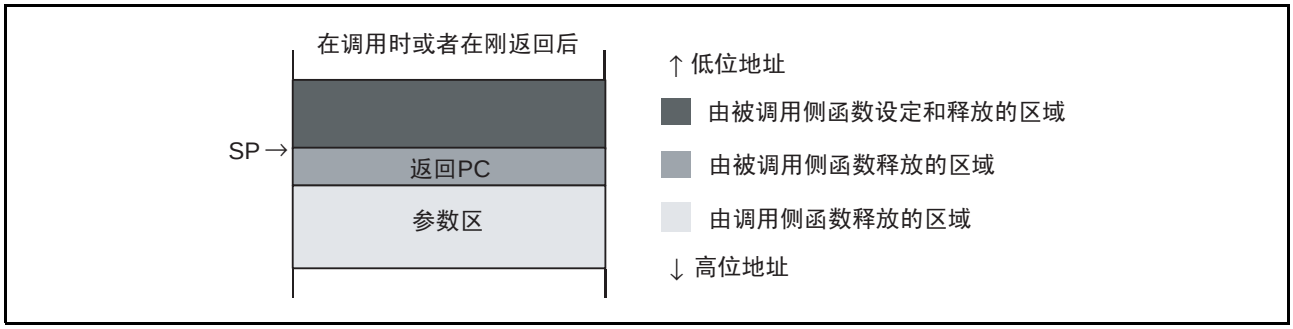


图 8.1 有关堆栈帧的分配和释放规则

8.2.2 有关寄存器的规则

在调用函数前后，是否保证相同的寄存器值因寄存器而不同。根据选项，还有特定用途的寄存器。寄存器的使用规则如表 8.2 所示。

表 8.2 寄存器的使用规则

寄存器	函数调用前后的保证值	函数入口	函数出口	高速中断寄存器 *1	基址寄存器 *2
R0	保证	堆栈指针	堆栈指针		
R1	不保证	参数 1	返回值 1		
R2	不保证	参数 2	返回值 2		
R3	不保证	参数 3	返回值 3		
R4	不保证	参数 4	返回值 4		
R5	不保证		（不定值）		
R6	保证		（保持入口值）		
R7	保证		（保持入口值）		
R8	保证		（保持入口值）		○
R9	保证		（保持入口值）		○
R10	保证		（保持入口值）	○	○
R11	保证		（保持入口值）	○	○
R12	保证		（保持入口值）	○	○
R13	保证		（保持入口值）	○	○
R14	不保证		（不定值）		
R15	不保证	指向结构体返回值的指针	（不定值）		

【注】 *1 有可能通过 fint_register 选项将 R10 ～ R13 的部分或者全部用于 “高速中断功能”。不能将分配为 “高速中断功能”的寄存器用于其他功能，功能的详细内容请参照选项的说明。

*2 有可能通过 base 选项将 R8 ～ R13 的部分或者全部用于 “基址寄存器功能”。不能将分配为 “基址寄存器功能”的寄存器用于其他功能，功能的详细内容请参照选项的说明。

8.2.3 有关参数的设定和参照的规则

以下阐述参数的一般规则和参数的分配方法。

参数的具体分配请参照 “8.2.5 参数分配的具体例子”。

(1) 参数的传递方法

必须在将参数值复制到寄存器或者堆栈的参数分配区后调用函数。因为在调用侧函数中不参照返回后的参数分配区，所以即使在被调用侧函数中更改参数值，也不直接影响调用侧的处理。

(2) 型转换的规则

(a) 将函数原型声明的参数转换为被声明的型。

(b) 函数原型未声明的参数的型转换规则如下：

- 将不超过2字节的整数型转换为4字节整数型。
- 将float型参数转换为double型。
- 不转换上述以外的参数。

例

```
void p(int,... );
void f( )
{
  char c;
  p(1.0, c);
}
```

因为c没有对应参数的型声明，所以被转换为4字节整数型。

因为1.0对应参数的型是int型，所以被转换为4字节整数型。

(3) 参数分配区

有将参数分配到寄存器的情况以及分配到堆栈参数区的情况，参数分配区如图 8.2 所示。

通常按照源程序的参数声明顺序，从序号小的寄存器开始将全部寄存器分配到堆栈。但是，对于有可变个数参数的函数等，即使寄存器有剩余也将参数分配到堆栈。另外，总是将 C++ 程序的动态函数成员的 this 指针分配到 R1。

参数分配区的一般规则分别如表 8.3 所示。

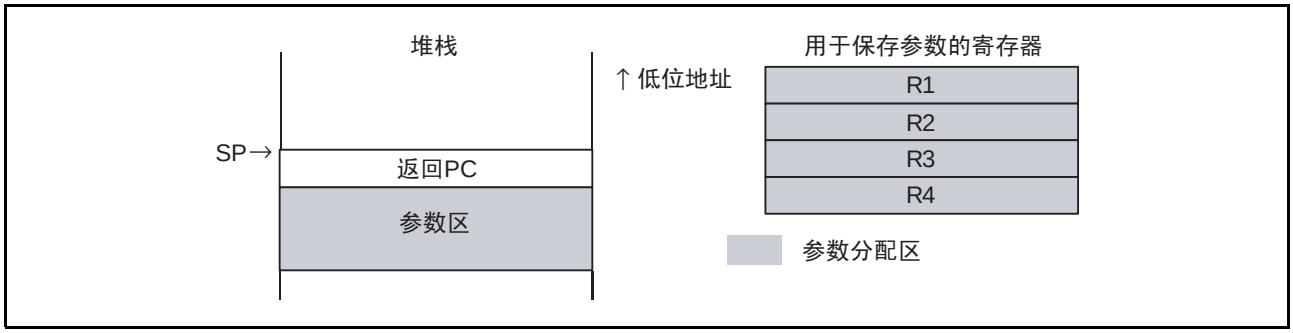


图 8.2 参数的分配区

表 8.3 参数分配区的一般规则

通过寄存器传递的参数			堆栈传递的参数
对象的型	用于保存参数的寄存器	分配方法	
signed char、(unsigned)char、 bool、(signed)short、 unsigned short、 (signed)int、unsigned int、 (signed)long unsigned long、 float、double*1、 long double*1、指针、指向数据成员的指针、参考	R1 ～ R4 中的 1 个	将 signed char、(signed)short 进行符号扩展以及 (unsigned)char、unsigned short 进行零扩展后的结果分配到寄存器。将其他型直接分配到寄存器。	1. 参数型不是寄存器传递对象的型。 2. 用函数原型对有可变个数参数的函数进行了声明的参数 *3。 3. 在 R1 ～ R4 中，尚未被分配为其他参数的个数少于分配时需要的个数。
(signed)long long、unsigned long long、 double*2、 long double*2	R1 ～ R4 中的 2 个	将低位 4 字节分配到序号小的寄存器，将高位 4 字节分配到序号大的寄存器。	
不超过 16 字节并且长度为 4 的倍数的结构体型、联合体型、类 （class）型	在 R1 ～ R4 中，容量被 4 除后的数	从存储器映像的前头，向寄存器序号增大的方向，按每 4 个字节进行分配。	

【注】 *1 这是没指定 dbl_size=8 的情况。

*2 这是指定了 dbl_size=8 的情况。

*3 在用函数原型对有可变个数参数的函数进行了声明的情况下，在声明中没有对应型的参数及其前面的参数为堆栈传递对象。在将不超过 2 字节的整数转换为 long 型，将 float 型转换为 double 型后，将没有型的参数全部作为边界调整数为 4 的参数进行处理。

例

```
int f2(int,int,int,int,...);
:
f2(a,b,c,x,y,z); → x、y、z 为堆栈传递。
```

(4) 堆栈传递参数的分配方法

堆栈传递参数的分配地址以及分配到堆栈的方法如**表 8.3** 所示：

- 将各参数分配到该边界调整数据对应的地址。
- 为了向堆栈加深的方向进行分配，按照参数从左到右的排列顺序保存到堆栈的参数区。即，当参数A及其右边的参数B都为堆栈传递参数时，参数B的地址是将参数A的分配地址加上参数A的占有大小后的地址调整为参数B的边界调整数后的地址。

8.2.4 有关返回值的设定和参照的规则

以下阐述返回值的一般规则和返回值的设定位置。

(1) 返回值的型转换

将返回值转换为该函数的返回型。

例

```
long f( );
long f( )
{
    float x;
    return x;   <—— 根据函数原型，将返回值转换为long型。
}
```

(2) 返回值的设定位置

根据函数返回值的型，有将返回值设定到寄存器的情况以及设定到存储器的情况。返回值的型和设定位置的关系请参照**表 8.4**。

表 8.4 返回值的型和设定位置

	返回值的型	返回值的设定位置
1	singed char,、 (unsigned)char、 (singed)short、 unsigned short、 (singed)int、 unsigned int、 (signed)long、 unsigned long、 float、 double*2、long double*2、指针、 bool、参考、指向数据成员的指针	R1 但是，设定 signed char (signed)short 进行符号扩展或者 (unsigned)char、 unsigned short 进行零扩展后的结果。
2	double*3、 long double*3、 (signed)long long、 unsigned long long	R1、 R2 将低位 4 字节设定到 R1，将高位 4 字节设定到 R2。
3	不超过 16 字节并且长度为 4 的倍数的结构体、联合体、类 （class）型	从存储器映像的前头，以 R1、 R2、 R3、 R4 的顺序，按每 4 个字节进行设定。
4	3. 以外的结构体、联合体、类 （class）型	返回值的设定区域 （存储器） *1

【注】 *1 在将函数的返回值设定到存储器时，将返回值设定到返回值地址指向的区域。在调用侧，除参数区以外还确保返回值的设定区域，在将该地址设定到 R15 后调用函数。

*2 这是没有指定 dbl_size=8 的情况。

*3 这是指定了 dbl_size=8 的情况。

8.2.5 参数分配的具体例子

参数分配的具体例子如下所示。在全部图中，地址从右向左增加 （左侧为高位地址）。

例 1. 按照声明顺序将寄存器传递对象的型的参数分配到寄存器 R1 ～ R4。
如果在中途有不是寄存器传递的参数，以后的参数就变为寄存器传递的对象。在堆栈中，将参数分配到调整为该参数的边界调整数后的地址。

```
int f(
    unsigned char,
    long long,
    long long,
    short,
    int,
    char,
    short,
    char,
    char,
    short);
        :
f(1,2,3,4,5,6,7,8,9,10);
 /*
** 1、2、4为寄存器传递的对象。
*/
```

＜寄存器＞

R1	0x000000（零扩展）	0x01
R2	0x00000002	
R3	0x00000000	
R4	0x0000（符号扩展）	0x0004

＜堆栈＞

*(R0+0)	0x00000003		
*(R0+4)	0x00000000		
*(R0+8)	0x00000005		
*(R0+12)	0x0007	空区域	0x06
*(R0+16)	0x000A	0x09	0x08

例 2. 不超过 16 字节并且为 4 的倍数的结构体和联合体型的参数为寄存器传递的对象，其他结构体和联合体型的参数为堆栈传递的对象。

```
union U {int a[2]; int b;} u;
struct S {short d; char c[4];} s;
struct T {char g; char f[2]; char e;} t;
int f(union U, struct S, struct T);
:
f(u, s, t);
/*
** u为8字节，是寄存器传递的对象。
** s为6字节，是堆栈传递的对象。
** t为4字节，是寄存器传递的对象。
*/
```

＜寄存器＞

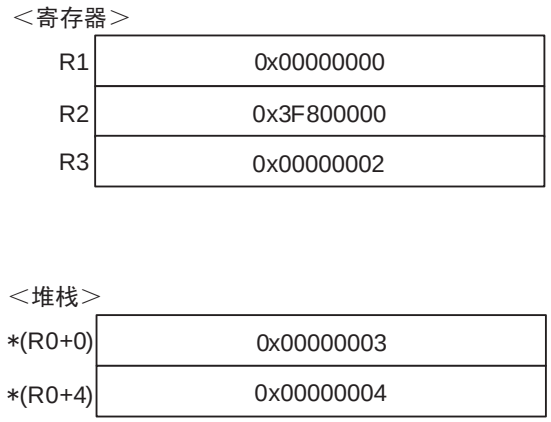
R1	u.a[0] (=u.b)			
R2	u.a[1]			
R3	e	f[1]	f[0]	g

＜堆栈＞

*(R0+0)	s.c[1]	s.c[0]	s.d	
*(R0+4)	空区域		s.c[3]	s.c[2]

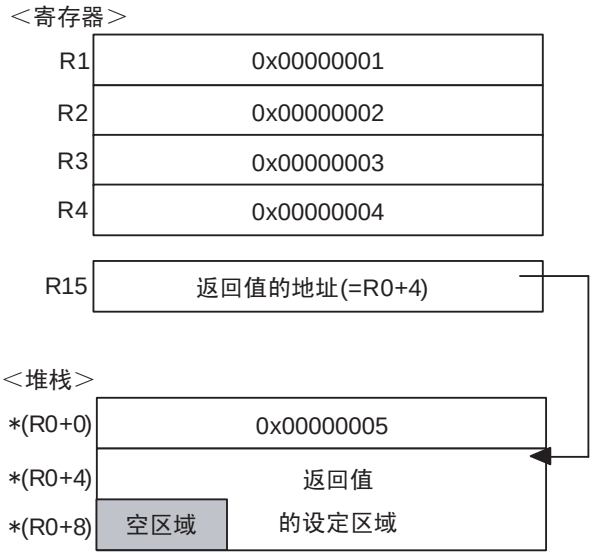
例 3. 在用函数原型对有可变个数参数的函数进行声明时，按照声明顺序将没有对应的型的参数及其前一个参数传递到堆栈。

```
int f(int, float, int, int...)
    :
f(0, 1.0, 2, 3, 4)
```



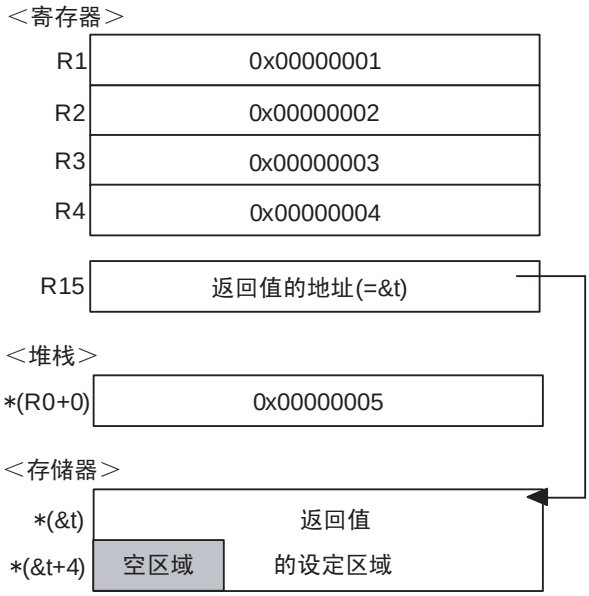
例 4. 在函数的返回型超过 16 字节或者不是 4 的倍数的结构体或者联合体型的情况下，将返回值的地址设定到 R15。

```
struct S{char a[7];};
struct S f(
   int a1,
   int a2,
   int a3,
   int a4,
   int a5);
   :
f(1,2,3,4,5);
```



例 5. 当将返回值设定到存储器时，如例 4 一样，通常在确保堆栈后进行设定；当将返回值设定为变量时，不确保堆栈而直接设定到该变量的存储区。此时，将该变量的地址设定到 R15。

```
struct S{char a[7];}t;
struct S f(
   int a1,
   int a2,
   int a3,
   int a4,
   int a5);
   :
t=f(1,2,3,4,5);
```



8.2.6 外部名的相互参照方法

C/C++ 程序中声明的外部名能在 C/C++ 程序和汇编程序之间相互参照或者进行更新。编译程序将以下内容作为外部名处理：

- 不是static存储类的全局变量 （C/C++程序）
- extern存储类声明的变量名 （C/C++程序）
- 没有指定static存储类的函数名 （C程序）
- 没有指定static存储类的非成员非inline函数名 （C++程序）
- 非inline的成员函数名 （C++程序）
- 静态数据成员名 （C++程序）

(1) C/C++ 程序中参照汇编程序的外部名的方法

在汇编程序中，用 .EXPORT 声明外部定义的符号名 （符号名的前头附加下划线 “_”）。
在 C/C++ 程序中，用 “extern” 声明符号名 （符号名的前头不加下划线 “_”）。

汇编程序 （定义侧）

```
      . .glb     _a, _b
      .SECTION D,ROMDATA,ALIGN=4
_a: .LWORD 1
_b: .LWORD 1
      .END
```

C/C++ 程序 （参照侧）

```
extern int a,b;

void f()
{
  a+=b;
}
```

(2) 从汇编程序参照 C/C++ 程序的外部 （变量和 C 函数）名的方法

在 C/C++ 程序中，对变量名 （变量名的前头不加下划线 “_”）进行外部定义。
在汇编程序中，用 .IMPORT 声明外部参照的外部名 （外部名的前头加下划线 “_”）。

C/C++ 程序 （定义侧）

```
int a;
```

汇编程序 （参照侧）

```
        .GLB      _a
        .SECTION  P,CODE
        MOV.L     #A_a,R1
        MOV.L     [R1],R2
        ADD       #1,R2
        MOV.L     R2,[R1]
        RTS
        .SECTION  D,ROMDATA,ALIGN=4
A_a:    .LWORD    _a
        .END
```

(3) 从汇编程序参照 C++ 程序的外部 （函数）名的方法

通过在汇编程序中用 “extern "C"”声明要参照的函数，能按和 (2) 相同的规则进行参照。但是，不能重复定义用 “extern "C"”声明的函数。

C++ 程序 （被调用侧）

```
 extern "C"
void sub ( )
{
    :
}
```

汇编程序 （调用侧）

```
        .GLB    _sub
        .SECTION P,CODE
                :



        PUSH.L  R13
        MOV.L   4[R0],R1
        MOV.L   R3,R12
        MOV.L   #_sub,R14
        JSR     R14
        POP     R13
        RTS
                 :
        .END
```

8.3 启动程序的建立

本章说明为设定程序执行时需要的环境而进行的处理。但是，执行程序的环境因用户系统而不同，需要根据用户系统的规格建立执行环境的设定程序。

需要的步骤概要如下所示：

- 固定向量表的设定
  为了通过上电复位启动初始设定例程 （PowerON_Reset），设定固定向量表。除复位向量以外，还能将特权指令异常、存取异常、未定义指令异常、浮点异常、非屏蔽中断的各处理例程注册到固定向量表。
- 初始设定
  进行到main函数为止需要的处理，对寄存器和段进行初始化以及调用各种初始设定例程。
- 低级接口例程的建立
  这是在使用标准输入/输出 （stdio.h、 ios、 streambuf、 istream、 ostream）和存储器管理库 （stdlib.h、 new）时需要的库函数与用户系统之间的接口例程。
- 结束处理例程 （exit、 atexit、 abort） *1的建立
  进行程序的结束处理。

【注】 *1 在使用程序结束处理的 C 库函数 exit、 atexit、 abort 函数时，需要根据用户系统建立这些函数。
在使用 C++ 程序或者 C 库函数 assert 宏时，需要建立 abort 函数。

8.3.1 固定向量表的设定

为了在上电复位时调用初始设定例程 PowerON_Reset，将 PowerON_Reset 的地址设定到固定向量表的复位向量。编码例子如下所示。

除复位向量以外，还能将特权指令异常、存取异常、未定义指令异常、浮点异常、非屏蔽中断的各处理程序注册到固定向量表。

有关固定向量表的详细内容，请参照硬件手册。

例

```
extern void PowerON_Reset_PC(void);

#pragma section VECTTBL    /* 通过 #pragma section 声明将 RESET_Vectors 输出到      */
                           /* CVECTTBL 段。                                          */
                           /* 指定在连接时通过 start 选项将 CVECTTBL 段分配到        */
                           /* 复位向量。                                             */
void (*const RESET_Vectors[])(void)={
    (void*) PowerON_Reset_PC,
};
```

8.3.2 初始设定

初始设定例程 PowerON_Reset 是记述在执行 main 函数前后所需步骤的函数。按顺序记述初始设定例程中需要的处理。

(1) 为初始设定处理而进行的 PSW 寄存器初始化

为了进行初始设定处理，对 PSW 寄存器进行初始化。例如，在初始设定处理中，为了设定为不接收中断，将 PSW 设定为中断禁止。

将复位时的 PSW 的全部位都初始化为 0，并且将中断允许位 （I 位）初始化为中断禁止状态 （0）。

(2) 堆栈指针的初始化

对堆栈指针 （USP 寄存器和 ISP 寄存器）进行初始化。通过对 PowerON_Reset 函数进行 “#pragma entry”声明，编译程序在函数的起始位置自动生成 ISP/USP 初始化代码。

因为对 PowerON_Reset 函数进行了 #pragma entry 声明，所以不需要记述此步骤。

(3) 用于基址寄存器的通用寄存器初始化

如果在编译程序中使用 base 选项，就需要对整个程序中用作基址的通用寄存器进行初始化。通过对 PowerON_Reset 函数进行 “#pragma entry”声明，编译程序在函数的起始位置自动生成各寄存器的初始化代码。

因为对 PowerON_Reset 函数进行了 #pragma entry 声明，所以不需要记述此步骤。

(4) 各种控制寄存器的初始化

将可变向量表的分配地址写到 INTB。除此以外，根据需要对 FINTV、FPSW、BPC、BPSW 进行初始化。能使用编译程序的内部函数对这些寄存器进行初始化。

但是，为了维持中断屏蔽的设定，只有 PSW 还没有进行初始化。

(5) 段的初始化处理

调用 RAM 区段的初始化例程 （_INITSCT）。将未初始化的数据段初始化为 0，初始化数据段将 ROM 区的初始值复制到 RAM 区。作为标准库，提供 _INITSCT。

需要用户将初始化对象的段记述到段的初始化表 （DTBL、 BTBL）。用段地址运算符设定 _INITSCT 函数使用的段的起始地址和结束地址。

段的初始化表的段名通过 C$BSEC 声明未初始化数据区，通过 C$DSEC 声明初始化数据区。

编码例子如下所示：

例

```
#pragma section C C$DSEC   // 将段名设定为 C$DSEC。
static const struct {
    void *rom_s;           // 初始化数据段的 ROM 中的起始地址成员
    void *rom_e;           // 初始化数据段的 ROM 中的结束地址成员
    void *ram_s;           // 初始化数据段的 RAM 中的起始地址成员
} DTBL[] = {__sectop("D"), __secend("D"), __sectop("R")};

#pragma section C C$BSEC   // 将段名设定为 C$BSEC。
static const struct {
    void *b_s;             // 未初始化数据段的起始地址成员
    void *b_e;             // 未初始化数据段的结束地址成员
} BTBL[] = {__sectop("B"), __secend("B")};
```

(6) 库的初始化处理

在使用 C/C++ 语言库函数时，调用需要初始化的执行例程 （_INITLIB）。
为了按照实际使用的功能进行最低限度的初始设定，请参考以下方针：

- 如果在已建立的低级接口例程中需要进行初始设定，就需要根据低级接口例程规格进行初始设定（_INIT_LOWLEVEL）。
- 在使用rand函数和strtok函数时，需要进行标准输入/输出以外的初始设定 （_INIT_OTHERLIB）。

进行库的初始设定的程序例子如下所示：

```
#include <stdio.h>
#include <stdlib.h>
#define IOSTREAM 3
const size_t _sbrk_size = 520;          // 指定堆区确保容量的最小单位。
                                        //  （在省略时：1024）
extern char *_s1ptr;

#ifdef __cplusplus
extern "C" {
#endif
void _INITLIB (void)
{
   _INIT_LOWLEVEL();                    // 对低级接口例程进行初始设定。
   _INIT_IOLIB();                       // 对输入 / 输出库进行初始设定。
   _INIT_OTHERLIB();                    // 对 rand 函数和 strtok 函数进行初始设定。
}

void _INIT_LOWLEVEL (void)
{                                       // 必须进行低级库需要的初始设定。
}

void _INIT_OTHERLIB(void)
{
srand(1);                               // 这是使用 rand 函数时的初始设定。
_s1ptr=NULL;                            // 这是使用 strtok 函数时的初始设定。
}
#ifdef __cplusplus
}
#endif
```

【注】 *1 指定标准输入 / 输出文件的文件名，此文件名由低级接口例程 “open”使用。
*2 在控制台等交互设备的情况下，将不进行缓冲的标志置位。

(7) 全局类目标的初始化

在开发 C++ 语言程序时，对调用被声明为全局类目标的构造函数的例程 （_CALL_INIT）进行调用。作为标准库，提供 _CALL_INIT。

(8) 为执行 main 函数而进行的 PSW 初始化

对 PSW 寄存器进行初始化，也解除中断屏蔽的设定。

(9) PSW 的 PM 位的更改

复位后，在特特权模式（PSW 的 PM 位为 0）运行，如果要转换用户模式，因为不能直接操作 PSW 的 PM 位，所以需要进行个别更改。在转换为用户模式时，具体的转换步骤如下：

- 将PSW压栈。
- 在转换PM位后立即将PC压栈。
  ※此时，将RTE指令后的位置压栈。
- 将堆栈中的PSW的PM位取反。
- 执行RTE指令。

(10) 用户程序的执行

执行 main 函数。

(11) 全局类目标的后处理

在开发 C++ 语言程序时，对调用被声明为全部类目标的析构函数的例程（_CALL_END）进行调用。作为标准库，提供 _CALL_END。

8.3.3 初始设定例程的记述例子

“8.3.2 初始设定”中说明的 PowerON_Reset 函数的编码例子如下所示：

```
#include     <machine.h>
#include     <_h_c_lib.h>
#include     "typedefine.h"
#include     "stacksct.h"

#ifdef __cplusplus
extern "C" {
#endif
void PowerON_Reset_PC(void);
void main(void);
#ifdef __cplusplus
}
#endif

#ifdef __cplusplus                              // Use SIM I/O
extern "C" {
#endif
extern void _INITLIB(void);
#ifdef __cplusplus
}
#endif

#define PSW_init  0x00010000
#define FPSW_init 0x00000100

#pragma section ResetPRG
#pragma entry PowerON_Reset_PC
void PowerON_Reset_PC(void)
{
             set_intb(__sectop("C$VECT"));
             set_fpsw(FPSW_init);

             _INITSCT();
             _INITLIB();
   nop();
             set_psw(PSW_init);
             main();
             brk();
}
```

8.3.4 低级接口例程

在 C/C++ 程序中使用标准输入 / 输出和存储器管理库时，必须建立低级接口例程。 C 库函数使用的低级接口例程一览表如表 8.5 所示。

表 8.5 低级接口例程一览表

	名称	功能
1	open	打开文件。
2	close	关闭文件。
3	read	读文件。
4	write	写文件。
5	lseek	设定文件的读写位置。
6	sbrk	确保存储区。
7	error_addr*	取得 errno 地址。
8	wait_sem*	确保信标。
9	signal_sem*	释放信标。

【注】 * 使用可重入库时必需。

需要在启动程序时进行低级接口例程所需的初始化，并且必须在库初始设定处理 _INITLIB 的“_INIT_LOWLEVEL”函数中进行初始化。

以下在说明低级输入 / 输出的基本方法后，说明各接口例程的规格。

【注】 函数名 open、close、read、write、lseek、sbrk、error_addr、wait_sem、signal_sem 是低级接口例程的保留标识符，在用户程序中不能使用。

(1) 输入 / 输出的基本方法

标准输入 / 输出库通过 FILE 型数据管理文件，而低级接口例程通过分配与实际文件一一对应的正整数进行文件管理，此整数称为文件序号。

open 例程对指定的文件名分配文件序号。在 open 例程中，为了能用此序号进行文件的输入 / 输出，需要设定以下信息：

- 文件设备的种类 （控制台、打印机和磁盘文件等）
  对于控制台和打印机等特殊设备，需要由系统决定特别的文件名，并且通过open例程进行判断。
- 进行文件缓冲时的缓冲区的位置和容量等信息
- 在磁盘文件的情况下，从文件的起始位置到下次读写位置为止的字节偏移量

以 open 例程设定的信息为基础，进行输入 / 输出 （read 例程、 write 例程）以及设定读写位置 （lseek 例程）。

在进行输出文件的缓冲时， close 例程必须将缓冲区的内容写到实际的文件中，并且使 open 例程设定的数据区能被重新使用。

(2) 低级接口例程的规格

本项说明建立低级接口例程的规格。在调用各例程时的接口及其运行以及实现时的注意事项如下所示。

用以下格式表示各例程的接口。低级接口例程必须为函数原型，在 C++ 程序内声明时，必须附加“extern "C"”。

凡例：

简单说明

(例程名)

说　明　　（表示例程的功能概要）

返回值　　正常：　　（表示正常结束时的返回值）
　　异常：　　（表示发生错误时的返回值）

参　数　　（名称）　　（含义）
　　（接口中表示的参数名）（表示作为参数传递的值）

文件的打开

long open(const char *name, long mode, long flg)

说　明　　为操作对应第 1 参数文件名的文件作准备。
在 open 例程中，为了以后的读写，必须决定文件的种类（控制台、打印机、磁盘文件等）。
在每次使用 open 例程返回的文件序号进行读写时，需要参照文件的种类。
第 2 参数的 mode 指定打开文件时的处理。此数据的各位含义如下所示：

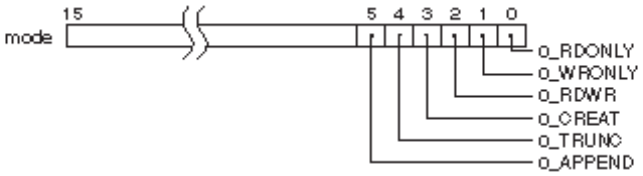


表 8.6　open 例程的 mode 位说明

Mode 位	说明
O_RDONLY （bit0）	当此位为 1 时，用只读模式打开文件。
O_WRONLY （bit1）	当此位为 1 时，用只写模式打开文件。
O_RDWR （bit2）	当此位为 1 时，用读写模式打开文件。
O_CREAT （bit3）	当此位为 1 时，如果文件名表示的文件不存在，就新建文件。
O_TRUNC （bit4）	当此位为 1 时，如果文件名表示的文件存在，就删除文件的内容，并且将文件大小更新为 0。
O_APPEND （bit5）	设定下次读写时的文件中的位置： 当此位为 0 时，设定到文件的起始位置。 当此位为 1 时，设定到文件的结束位置。

如果 mode 指定的文件处理和实际的文件性质出现矛盾，就发生错误。如果能正常打开文件，就必须返回被以后的 read、 write、 lseek、 close 例程使用的文件序号（正整数）。需要用低级接口例程管理文件序号和实际文件的对应。如果打开失败，就返回 -1。

返回值　　正常：　　正常打开的文件序号
　　异常：　　-1

参　数　　name　　指向文件的文件名字符
　　mode　　指定文件打开时的处理。
　　flg　　指定文件打开时的处理（总是 0777）。

文件的关闭

long close(long fileno)

说　明　将 open 例程得到的文件序号作为参数进行传递。
为了能重新使用，必须释放 open 例程设定的文件管理信息区。如果在低级接口例程中进行输出文件的缓冲，就必须将缓冲器的内容写到实际的文件中。
如果能正常关闭文件，就返回 0，否则返回 -1。

返回值　正常：　0
异常：　-1

参　数　fileno　要关闭的文件序号

读数据

long read(long fileno, unsigned char *buf, long count)

说　明　将数据从第 1 参数（fileno）指定的文件读到第 2 参数（buf）指向的区域，第 3 参数（count）指定读数据的字节数。
如果文件结束，就只能读小于等于 count 指定的字节数。
文件的读写位置只向前移动被读的字节数。
如果能正常读，就返回实际读的字节数，否则返回 -1。

返回值　正常：　实际读到的字节数
异常：　-1

参　数　fileno　读对象的文件序号
buf　保存读数据的区域
count　读数据的字节数

写数据

long write(long fileno, const unsigned char *buf, long count)

说　明　将数据从第 2 参数 （buf）指向的区域写到第 1 参数 （fileno）指定的文件中，第 3 参数 （count）指定写数据的字节数。
如果要写文件的设备 （磁盘等）没有空间，就只能写小于等于 count 指定的字节数。建议在实际写的字节数连续几次为 0 字节时，可认为磁盘已满，返回错误 （-1）。
文件的读写位置只向前移动被写的字节数。
如果能正常写，就返回实际写的字节数，否则就返回 -1。

返回值　正常：　实际写的字节数
异常：　-1

参　数　fileno　写对象的文件序号
buf　写数据的区域
count　写数据的字节数

文件中的位置设定

long lseek(long fileno, long offset, long base)

说　明　以字节为单位设定文件读写时的文件中的位置。
必须根据第 3 参数 （base），用以下方法计算并且设定文件中的新位置。
(1) 在 base 为 0 时，设定到从文件的起始位置开始的 offset 字节的位置。
(2) 在 base 为 1 时，设定到从当前位置加上 offset 字节后的位置。
(3) 在 base 为 2 时，设定到文件大小加上 offset 字节后的位置。
如果文件是控制台或者打印机等交互设备，或者新的偏移量为负或者在 (1)(2) 时超过文件的大小，就发生错误。
如果能正确设定文件位置，就返回新的读写位置 （从文件起始位置开始的偏移量），否则返回 -1。

返回值　正常：　新的读写位置 （从文件起始位置开始的偏移量）（以字节为单位）
异常：　-1

参　数　fileno　对象的文件序号
offset　读写位置的偏移量 （以字节为单位）
base　偏移的起点

存储区的分配

char *sbrk(size_t size)

说　明　　将存储区的分配容量作为参数进行传递。
在连续调用 sbrk 例程时，必须从低位地址开始依次分配连续的区域。如果分配的存储区不够，就发生错误。
如果能正常分配，就返回分配区的起始地址，否则返回 “(char *)-1”。

返回值　　正常：　　分配区的起始地址
异常：　　(char *)-1

参　数　　size　　分配的数据长度

long *errno_addr(void)

说　明　　返回当前任务的错误号地址。
在使用由标准库生成工具指定 reent 选项而建立的标准库时，需要此函数。

返回值　　当前任务的错误号地址

信标的确保

long wait_sem(long semnum)

说　明　　确保 semnum 指定的信标。
如果能确保，就返回 1，否则返回 0。
在使用由标准库生成工具指定 reent 选项而建立的标准库时，需要此函数。

返回值　　正常：　　1
异常：　　0

参　数　　semnum　　信标 ID

信标的释放

long signal_sem(long semnum)

说　明	释放 semnum 指定的信标。 如果能释放，就必须返回 1，否则返回 0。 在使用由标准库生成工具指定 reent 选项而建立的标准库时，需要此函数。	
返回值	正常：	1
	异常：	0
参　数	semnum	信标 ID

(3) 低级接口例程的编码例子

```
/************************************************************************/
/*                                  lowsrc.c:                           */
/*----------------------------------------------------------------------*/
/*      RX 族  仿真调试程序  接口例程                                   */
/*              - 只支持标准输入 / 输出 （stdin,stdout,stderr） -       */
/************************************************************************/
#include <string.h>

/*  文件序号  */
#define STDIN  0                        /*  标准输入 （控制台）     */
#define STDOUT 1                        /*  标准输出 （控制台）     */
#define STDERR 2                        /*  标准错误输出 （控制台） */

#define FLMIN 0                         /*  最小的文件序号  */
#define FLMAX 3                         /*  文件数的最大值  */

/*  文件标志  */
#define O_RDONLY 0x0001                 /*  只读  */
#define O_WRONLY 0x0002                 /*  只写  */
#define O_RDWR 0x0004                   /*  读写  */

/*  特殊字符码  */
#define CR 0x0d                         /*  回车  */
#define LF 0x0a                         /*  换行  */

/*  sbrk 管理的区域容量  */
#define HEAPSIZE 1024

/************************************************************************/
/*                              参照函数的声明                          */
/*    由仿真调试程序将字符输入或者输出到控制台而进行汇编程序的参照。    */
/************************************************************************/
extern void charput(char);              /*  一个字符的输入处理  */
extern char charget(void);              /*  一个字符的输出处理  */

/************************************************************************/
/*                              静态变量的定义                          */
/*                  定义低级接口例程使用的静态变量。                    */
/************************************************************************/
char flmod[FLMAX];                      /*  文件打开时的模式设定位置  */

union HEAP_TYPE{
    long dummy;                         /*  虚设 4 字节的边界。  */
    char heap[HEAPSIZE];                /*  声明 sbrk 管理的区域。  */
};

static union HEAP_TYPE heap_area;

static char *brk=(char*)&heap_area;     /*  sbrk 分配的区域结束地址  */
```

```
/************************************************************************/
/*                              open：打开文件。                        */
/*                          返回值：文件序号（成功）                    */
/*                                  -1    （失败）                      */
/************************************************************************/
long open(const char *name,                 /*  文件名              */
          long mode,                        /*  文件模式            */
          long flg)                         /*  处理的指定 （未使用） */
{
    /*  根据文件名检查模式，返回文件序号。  */

    if (strcmp(name,"stdin")==0) {          /*  标准输入文件 */
        if ((mode&O_RDONLY)==0) {
            return (-1);
        }
        flmod[STDIN]=mode;
        return (STDIN);
    }

    else if (strcmp(name,"stdout")==0) {    /*  标准输出文件  */
        if ((mode&O_WRONLY)==0) {
            return (-1);
        }
        flmod[STDOUT]=mode;
        return (STDOUT);
    }

    else if (strcmp(name,"stderr")==0){     /* 标准错误输出文件  */
        if ((mode&O_WRONLY)==0) {
            return (-1);
        }
        flmod[STDERR]=mode;
        return (STDERR);
    }

    else {
        return (-1);                        /*  错误  */
    }
}

/************************************************************************/
/*                             close：关闭文件。                        */
/*                          返回值： 0    （成功）                      */
/*                                  -1    （失败）                      */
/************************************************************************/
long close(long fileno)                     /*  文件序号  */
{
    if (fileno<FLMIN || FLMAX<fileno) {     /*  检查文件序号的范围。 */
        return -1;
    }

    flmod[fileno]=0;                        /*  进行文件模式的复位。  */

    return 0;
}
```

```
/************************************************************************/
/*                              read：读数据。                          */
/*                      返回值：实际读的字符数  （成功）                */
/*                              -1              （失败）                */
/************************************************************************/
long read(long fileno,                      /*  文件序号            */
          unsigned char *buf,               /*  传送目标的缓冲器地址  */
          long  count)                      /*  读数据的字符数        */
{
    unsigned long i;

    /*  根据文件名检查模式，逐个输入字符并且保存到缓冲区。  */

    if (flmod[fileno]&O_RDONLY || flmod[fileno]&O_RDWR) {
        for (i=count;i>0;i--) {
            *buf=charget();
            if (*buf==CR) {                 /*  换行字符的替换  */
                *buf=LF;
            }
        buf++;
        }
        return count;
    }

    else {
        return -1;
    }
}

/************************************************************************/
/*                              write：写数据。                         */
/*                      返回值：实际写的字符数  （成功）                */
/*                              -1              （失败）                */
/************************************************************************/
long write(long fileno,                     /*  文件序号  */
           const unsigned char *buf,        /*  传送源的缓冲器地址  */
           long count)                      /*  写数据的字符数  */
{
    unsigned long i;
    unsigned char c;

    /*  根据文件名检查模式，逐个输出字符。  */

    if (flmod[fileno]&O_WRONLY || flmod[fileno]&O_RDWR) {
        for (i=count; i>0; i--) {
            c=*buf++;
            charput(c);
        }
        return count;
    }

    else {
        return -1;
    }
}
```

```
/************************************************************************/
/*                          lseek：设定文件的读写位置。                 */
/*          返回值：读写位置，从文件起始位置开始的偏移量  （成功）      */
/*                  -1                                    （失败）      */
/*                      （在控制台输入 / 输出时，不支持 lseek） */
/************************************************************************/
long lseek(long fileno,                         /*  文件序号        */
           long offset,                         /*  读写位置        */
           long  base)                          /*  偏移的起点      */
{
    return -1;
}

/************************************************************************/
/*                                sbrk：分配存储区。                    */
/*                        返回值：分配的区域起始地址      （成功）      */
/*                                -1                      （失败）      */
/************************************************************************/
char *sbrk(size_t size)                         /*  分配的区域容量    */
{
    char *p;

    /*  检查空区域。  */

    if (brk+size>heap_area.heap+HEAPSIZE) {
        return (char *)-1;
    }

    p=brk;                                      /*  分配区域。      */
    brk+=size;                                  /*  更新结束地址。  */
    return p;
}
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                           lowlvl.src                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  RX Family Simulator/Debugger Interface Routine   ;
;             - Inputs and outputs one character -                          ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
          .GLB    _charput
          .GLB    _charget

SIM_IO    .EQU 0h

          .SECTION   P,CODE
;---------------------------------------------------------------------------
;  _charput:
;---------------------------------------------------------------------------
_charput:
          MOV.L      #IO_BUF,R2
          MOV.B      R1,[R2]
          MOV.L      #1220000h,R1
          MOV.L      #PARM,R3
          MOV.L      R2,[R3]
          MOV.L      R3,R2
          MOV.L      #SIM_IO,R3
          JSR        R3
          RTS

;---------------------------------------------------------------------------
;  _charget:
;---------------------------------------------------------------------------
_charget:
         MOV.L       #1210000h,R1
         MOV.L       #IO_BUF,R2
         MOV.L       #PARM,R3
         MOV.L       R2,[R3]
         MOV.L       R3,R2
         MOV.L       #SIM_IO,R3
         JSR         R3
         MOV.L       #IO_BUF,R2
         MOVU.B      [R2],R1
         RTS

;---------------------------------------------------------------------------
;  I/O Buffer
;---------------------------------------------------------------------------
          .SECTION  B,DATA,ALIGN=4
PARM:     .BLKL     1
          .SECTION  B_1,DATA
IO_BUF:   .BLKB     1
          .END
```

(4) 用于可重入库的低级接口例程例子

用于可重入库的低级接口例程例子如下所示。在使用由标准库生成工具指定 reent 选项而建立的库时，需要此程序。

如果 wait_sem 函数或者 signal_sem 函数返回 NG，就给 errno 设定以下内容，并且从库函数返回。

wait_sem	EMALRESM	Malloc 的信标资源的确保失败。
	ETOKRESM	Strtok 的信标资源的确保失败。
	EIOBRESM	_iob 的信标资源的确保失败。
signal_sem	EMALFRSM	malloc 的信标资源的释放失败。
	ETOKFRSM	strtok 的信标资源的释放失败。
	EIOBRESM	_iob 的信标资源的释放失败。

如果在确保信标后发生更高优先级的中断并且再次确保信标，就会发生死锁。因此，不能对共享资源等处理发生中断嵌套。

```
#define MALLOC_SEM              1               /*  malloc 的信标 No.    */
#define STRTOK_SEM              2               /*  strtok 的信标 No.    */
#define FILE_TBL_SEM            3               /*  _iob 的信标 No.      */
#define SEMSIZE                 4
#define TRUE                    1
#define FALSE                   0
#define OK                      1
#define NG                      0

extern long *errno_addr(void);
extern long wait_sem(long);
extern long signal_sem(long);

long sem_errno;
int force_fail_signal_sem = FALSE;
static int semaphore[SEMSIZE];

/************************************************************************/
/*                          errno_addr:：取得 errno 地址                 */
/*                            返回值：errno 地址                        */
/************************************************************************/
long *errno_addr(void)
{
    /*  必须返回当前任务的 errno 地址。  */
    return (&sem_errno);
}
```

```
/************************************************************************/
/*                          wait_sem：确保被指定的信标。                */
/*                            返回值：OK(=1) （成功）                   */
/*                                    NG(=0) （失败）                   */
/************************************************************************/
long wait_sem(long semnum)     /*  信标 ID     */
{
    if((0 < semnum) && (semnum < SEMSIZE)) {
        if(semaphore[semnum] == FALSE) {
            semaphore[semnum] = TRUE;
            return(OK);
        }
    }
    return(NG);
}


/************************************************************************/
/*                         signal_sem：释放被指定的信标。               */
/*                            返回值：OK(=1) （成功）                   */
/*                                    NG(=0) （失败）                   */
/************************************************************************/
long signal_sem(long semnum)   /*  信标 ID     */
{
    if(!force_fail_signal_sem) {
        if((0 <= semnum) && (semnum < SEMSIZE)) {
            if( semaphore[semnum] == TRUE ) {
                semaphore[semnum] = FALSE;
                return(OK);
                }
            }
    }
    return(NG);
}
```

8.3.5 结束处理例程

(1) 结束处理的注册和执行 （atexit）例程的建立例子

以下说明库 atexit 函数 （注册结束处理）的建立方法。

atexit 函数将作为参数传递的函数地址注册到结束处理表。如果注册的函数个数超过极限值 （在此，能注册的函数个数为 32 个）或者重复注册相同的函数，就返回非 0 值 （为 1），否则返回 0。

程序例子如下所示：

例：

```
#include <stdlib.h>

long _atexit_count=0 ;

void (*_atexit_buf[32])(void) ;

#ifdef __cplusplus
extern "C"
#endif
long atexit(void (*f)(void))
{
    int i;

    for(i=0; i<_atexit_count ; i++)          // 检查是否已经注册。
        if(_atexit_buf[i]==f)
            return 1;
    if (_atexit_count==32)                   // 检查注册数的极限值。
        return 1;
   else {
        _atexit_buf[_atexit_count++]=f;      // 注册函数的地址。
        return 0;
    }
}
```

(2) 程序结束 （exit）例程的建立例子

以下说明库 exit 函数 （进行程序的结束处理）的建立方法。程序的结束处理因用户系统而不同，所以必须参考以下的程序例子，建立符合用户系统规格的结束处理。

exit 函数根据有参数传递的程序结束码进行程序的结束处理，返回到程序启动时的环境。在此，通过将结束码设定为外部变量并且在即将调用 main 函数前返回到 setjmp 函数保存的环境来实现。为了返回到程序执行前的环境，必须在建立以下的函数 “callmain”后调用函数 “callmain”，以代替从初始设定函数 “PowerON_Reset”调用 “main”函数。

程序例子如下所示：

```
#include <setjmp.h>
#include <stddef.h>

extern long _atexit_count ;
extern void (*_atexit_buf[32])(void) ;
#ifdef __cplusplus
extern "C"
#endif
void _CLOSEALL(void);
int main(void);
extern jmp_buf _init_env ;
int _exit_code ;

#ifdef __cplusplus
extern "C"
#endif
void exit(int code)
{
   int i;
   _exit_code=code ;                          // 将返回码设定到 _exit_code。
   for(i=_atexit_count-1; i>=0; i--)          // 依次执行 atexit 函数注册的函数。
   (*_atexit_buf[i])();
   _CLOSEALL();                               // 关闭所有打开的文件。
   longjmp(_init_env, 1) ;                    // 返回到 setjmp 保存的环境。
}
#ifdef __cplusplus
extern "C"
#endif
void callmain(void)
{
    //  用 setjmp 保存当前的环境，调用 main 函数。
    if(!setjmp(_init_env))
       _exit_code=main();                    // 在从 exit 函数返回时结束处理。
}
```

(3) 异常结束 （abort）例程的建立例子

在异常结束时，必须根据所使用的用户系统规格进行程序异常结束的处理。
在使用 C++ 程序时，在以下情况下也调用 abort 函数：

- 异常处理无法正常运行时
- 调用纯虚拟函数时
- dynamic_cast 失败时
- typeid 失败时
- 在删除类数组的情况下无法取得信息时
- 在注册类目标的析构函数调用信息的情况下出现矛盾时

在以下的程序例子中，在将信息输出到标准输出设备后关闭文件，然后进入无限循环等待复位。
例：

```
#include <stdio.h>

#ifdef __cplusplus
extern "C"
#endif
void _CLOSEALL(void);
#ifdef __cplusplus
extern "C"
#endif
void abort(void)
{
      printf("program is abort !!\n");      // 输出信息。
      _CLOSEALL();                          // 关闭文件。
      while(1)    ;                         // 进入无限循环。
}
```

9. C/C++ 语言规格

9.1 语言规格

9.1.1 编译程序的规格

有关在语言规格中未规定的实现性定义项目，编译程序的规格如下所示：

(1) 环境

表 9.1 环境的规格

	项 目	编译程序的规格
1	给 main 函数的实际参数的含义	未规定。
2	交互输入 / 输出设备的结构	未规定。

(2) 标识符

表 9.2 标识符的规格

	项 目	编译程序的规格
1	不和外部连接的标识符 （内部名）的有效字符数	前 8189 个字符有效。
2	和外部连接的标识符 （外部名）的有效字符数	前 8191 个字符有效。
3	和外部连接的标识符 （外部名）大小写字母的区别	区别大小写字母。

(3) 字符

表 9.3 字符的规格

	项 目	编译程序的规格
1	源字符集和执行环境字符集的元素	都是 ASCII 字符集。但是，在字符串和字符常数中能记述 shift JIS、EUC 汉字码、Latin1 码或者 UTF-8 码。
2	在多字节字符的代码化中使用的 shift 状态	不支持 shift 状态。
3	执行程序时的字符集的字符位数	位数为 8 位。
4	字符常数和字符串内的源字符集的字符和执行环境字符集的字符的对应	对应相同的 ASCII 字符。
5	整数字符常数 （包括语言规格中未规定的字符或者扩充符号）的值	不支持语言规格中未规定的字符和扩充符号。
6	字符常数 （包括超过 1 个字符）或者宽字符常数 （包括超过 1 个字符的多字节字符）的值	字符常数的高位 2 字节有效。 不支持宽字符常数。 如果指定的的字符超过 1 个，就输出警告错误。
7	用于将多字节字符转换为宽字符的 locale 规格	不支持 locale。
8	char 型的值	值的范围和 unsigned char 型相同 *1。

【注】 *1 当指定 signed_char 选项时，值的范围和 signed char 型相同。

(4) 整数

表 9.4 整数的规格

	项 目	编译程序的规格
1	整数型的表示方法及其值	如表 9.5 所示。
2	将短于整数值长度的带符号整数型或者无符号整数型转换为相同长度的带符号整数型时的值 （用转换后的型无法表示结果值）	整数值的低位 4 字节 （转换后的型长度为 4 字节）、低位 2 字节 （转换后的型长度为 2 字节）或者低位 1 字节 （转换后的型长度为 1 字节）为转换后的值。
3	带符号整数的逐位运算结果	为带符号的值。
4	整数除法运算中的余数符号	和被除数的符号相同。
5	负数的带符号标量型的右移结果	保持符号位。

表 9.5 整数型及其值的范围

	型	值的范围	数据长度
1	char*1	0 ～ 255	1 字节
2	signed char	–128 ～ 127	1 字节
3	unsigned char	0 ～ 255	1 字节
4	short signed short	–32768 ～ 32767	2 字节
5	unsigned short*2	0 ～ 65535	2 字节
6	int*2 signed int	–2147483648 ～ 2147483647	4 字节
7	unsigned int	0 ～ 4294967295	4 字节
8	long signed long	–2147483648 ～ 2147483647	4 字节
9	unsigned long	0 ～ 4294967295	4 字节
10	long long signed long long	–9223372036854775808 ～ 9223372036854775807	8 字节
11	unsigned long long	0 ～ 18446744073709551615	8 字节

【注】 *1 当指定 signed_char 选项时，作为 signed char 型进行处理。

*2 当指定 int_to_short 选项时，将 int 型作为 short 型进行处理，将 signed int 型作为 signed short 型进行处理并且将 unsigned int 型作为 unsigned short 型进行处理。

(5) 浮点

表 9.6 浮点的规格

	项 目	编译程序的规格
1	浮点型的表示方法及其值	浮点型有 float 型、 double 型和 long double 型。在 “9.1.3 浮点型的规格” 中说明浮点型的内部表示、转换规格、运算规格等性质。浮点型能表示的值的极限值如表 9.7 所示。
2	将整数转换为不能正确表示实际值的浮点型时的舍去方向	
3	将浮点型转换为更短浮点型时的舍去方法或者舍入方法	

表 9.7　浮点型的极限值

	项　目	极限值	
		10 进制数表示 *1	内部表示 （16 进制数）
1	float 型的最大值	3.4028235677973364e+38f (3.4028234663852886e+38f)	7f7fffff
2	float 型的正最小值	7.0064923216240862e–46f (1.4012984643248171e–45f)	00000001
3	double*2 long double*2 } 型的最大值	1.7976931348623158e+308 (1.7976931348623157e+308)	7fefffffffffffff
4	double*2 long double*2 } 型的正最小值	4.9406564584124655e–324 (4.9406564584124654e–324)	0000000000000001

【注】 *1　10 进制数表示的极限值是不为 0 或者不为无穷大的极限值。( ) 内表示理论值。
*2　这是指定 dbl_size=8 时的解释。当指定 dbl_size=4 时， double 型、 long double 型的值和 float 型相同。

(6)　数组和指针

表 9.8　数组和指针的规格

	项　目	编译程序的规格
1	保持数组大小的最大值时需要的整数型 （size_t）	unsigned long 型
2	从指针型转换为整数型 （指针型的长度≥整数型的长度）	为指针型低位字节的值。
3	从指针型转换为整数型 （指针型的长度＜整数型的长度）	进行符号扩展。
4	从整数型转换为指针型 （整数型的长度≥指针型的长度）	为整数型低位字节的值。
5	从整数型转换为指针型 （整数型的长度＜指针型的长度）	进行符号扩展。
6	保持相同数组内的成员指针之间的差时需要的整数型 （ptrdiff_t）	int 型

(7)　寄存器

表 9.9　寄存器的规格

	项　目	编译程序的规格
1	能分配到寄存器的变量型	char、 unsigned char、 bool、 short、 unsigned short、 int、 unsigned int、 long、 unsigned long、 long long、 unsigned long long、 float、 指针

(8) 类、结构体、联合体、枚举型、位域

表 9.10 类、结构体、联合体、枚举型、位域的规格

	项 目	编译程序的规格
1	由不同型的成员存取的联合体型成员的参照	能参照，但是不保证值。
2	类、结构体成员的调整	在类、结构体成员中，调整数的最大值为该类、结构体的调整数。分配方法的详细规格请参照“9.1.2(2) 结构体 / 联合体 （C 语言）和类（class）型 （C++ 语言）”。
3	单一 int 型位域的符号	unsigned int 型 *3
4	int 型长度内的位域分配顺序	从低位开始分配 *1*2。
5	在 int 型长度内已分配位域时，下次分配的位域长度超过 int 型内剩余长度时的分配方法	分配到下一个 int 型的区域 *1。
6	位域中允许的型说明符	char、 unsigned char、 bool、 short、 unsigned short、 int、 unsigned int、 long、 unsigned long、 enum、 long long、 unsigned long long
7	表示枚举型值的整数型	int 型 *4

【注】 *1 有关位域的分配方法的详细内容，请参照 “9.1.2(3) 位域”。

*2 当指定 bit_order=left 选项时，从高位开始分配。

*3 当指定 signed_bitfield 选项时，为 signed int 型。

*4 当指定 auto_enum 选项时，为枚举值范围内最小的型，详细内容请参照 “2.5 单片机选项”中的 auto_enum 选项的说明。

(9) 型限定词

表 9.11 型限定词的规格

	项 目	编译程序的规格
1	volatile 型数据的存取种类	未规定。

(10) 声明

表 9.12 声明的规格

	项 目	编译程序的规格
1	限定基本型 （算术型、结构体型、联合体型）的说明符的个数	最多能指定 16 个。

以下举例说明限定基本型的型个数的计算方法：

例：

```
(i)   int a;       a 是 int 型 （基本型），限定基本型的型个数为 0。
(ii)  char *f();   f 是返回指向 char 型 （基本型）的指针型的函数型，限定基本型的型个数为 2。
```

(11) 语句

表 9.13 语句的规格

	项 目	编译程序的规格
1	一个 switch 语句中能指定的 case 标号数	最多能指定 2147483646 个。

(12) 预处理器

表 9.14 预处理器的规格

	项 目	编译程序的规格
1	在条件编译的常数表达式中，单字符的字符常数和执行环境字符集的对应	预处理器语句的字符常数和执行环境字符集相同。
2	include 文件的读取方法	从 include 选项指定的路径读用 “<”、“>”括起来的文件。 如果找不到文件，就按照环境变量 INC_RX 指定的文件夹、环境变量 BIN_RX 指定的文件夹的顺序，检索各文件夹。
3	是否支持用双重引用符括起来的 include 文件	支持。从当前文件夹读 include 文件。如果当前文件夹中不存在该文件，就按照本表项 2 的读取方法读 include 文件。
4	源文件的字符排列的对应 （宏展开后的字符串的空格字符）	空格字符串展开为 1 个空格字符。
5	#pragma 的运行	请参照 “9.2.1 #pragma”。
6	_ _DATE_ _、 _ _TIME_ _ 的值	在编译开始时，设定主机定时器的值。

9.1.2 数据的内部表示

本节阐述型名和数据的内部表示的对应。数据的内部表示由以下项目构成：

- 数据长度
  数据占有区域的大小。
- 数据的调整数
  这是有关分配数据的地址限制。有分配到任意地址的1字节调整数、分配到偶数字节的2字节调整数和分配到4的倍数字节的4字节调整数。
- 值的范围
  表示标量型 （C语言）、基本型 （C++语言）的值的范围。
- 数据的分配例子
  表示结构体/联合体 （C语言）、类 （class）型 （C++语言）元素的数据分配方法。

(1) 标量型 （C 语言）、基本型 （C++ 语言）

C 语言中的标量型和 C++ 语言中的基本型的内部表示如**表 9.15** 所示。

表 9.15 标量型和基本型的内部表示

	型名	长度（byte）	调整数（byte）	符号的有无	值的范围	
					最小值	最大值
1	char*1	1	1	无	0	2^8–1 (255)
2	signed char	1	1	有	-2^7 (–128)	2^7–1 (127)
3	unsigned char	1	1	无	0	2^8–1 (255)
4	short	2	2	有	-2^{15} (–32768)	2^{15}–1 (32767)
5	signed short	2	2	有	-2^{15} (–32768)	2^{15}–1 (32767)
6	unsigned short	2	2	无	0	2^{16}–1 (65535)
7	int*2	4	4	有	-2^{31} (–2147483648)	2^{31}–1 (2147483647)
8	signed int*2	4	4	有	-2^{31} (–2147483648)	2^{31}–1 (2147483647)
9	unsigned int*2	4	4	无	0	2^{32}–1 (4294967295)
10	long	4	4	有	-2^{31}(–2147483648)	2^{31}–1 (2147483647)
11	signed long	4	4	有	-2^{31}(–2147483648)	2^{31}–1 (2147483647)
12	unsigned long	4	4	无	0	2^{32}–1 (4294967295)
13	long long	8	4	有	-2^{63} (–9223372036854775808)	2^{63}–1 (9223372036854775807)
14	signed long long	8	4	有	-2^{63} (–9223372036854775808)	2^{63}–1 (9223372036854775807)
15	unsigned long long	8	4	无	0	2^{64}–1 (18446744073709551615)
16	float	4	4	有	–∞	+∞
17	double long double	4*4	4	有	–∞	+∞
18	size_t	4	4	无	0	2^{32}–1 (4294967295)
19	ptr_diff_t	4	4	有	-2^{31}(–2147483648)	2^{31}–1 (2147483647)
20	enum*3	4	4	有	-2^{31}(–2147483648)	2^{31}–1 (2147483647)
21	指针	4	4	无	0	2^{32}–1 (4294967295)

	型名	长度（byte）	调整数（byte）	符号的有无	值的范围	
					最小值	最大值
22	bool*5	4	4	有	—	—
23	参照 *6	4	4	无	0	2^{32}–1 (4294967295)
24	指向数据成员的指针 *6	4	4	有	0	2^{32}–1 (4294967295)
25	指向函数成员的指针 *6 *7	12	4	—	—	—

【注】 *1 当指定 signed_char 选项时，和 signed char 型相同。

*2 当指定 int_to_short 选项时， int 型和 short 型相同， signed int 型和 signed short 型相同， unsigned int 型和 unsigned short 型相同。

*3 当指定 auto_enum 选项时，为枚举值范围内最小的型。

*4 当指定 dbl_size=8 时， double 型和 long double 型的长度为 8 字节。

*5 只在 C++ 编译和 C99 编译时有效。

*6 只在 C++ 编译时有效。

*7 用以下数据结构表示指向函数成员、虚拟函数成员的指针。

```
class PMF{
 public:
   long d;          // 目标的偏移值
   long i;          // 当对象成员函数是虚拟函数时，为虚拟函数表中的 index。
   union{
     void (*f)();   // 当对象成员函数是非虚拟函数时，为函数地址。
     long offset;   // 当对象成员函数是虚拟函数时，为虚拟函数表的目标中的偏移量。
   };
 };
```

(2) 结构体 / 联合体 （C 语言）和类 （class）型 （C++ 语言）

本项说明 C 语言中的数组型、结构体型、联合体型以及 C++ 语言中的类 （class）型的内部表示。
结构体 / 联合体、类 （class）型的内部表示如表 9.16 所示。

表 9.16 结构体 / 联合体、类 （class）型的内部表示

	型名	调整数 （byte）	长度 （byte）	数据的分配例子		
1	数组型	数组元素的调整数	数组元素的个数 × 元素长度	char a[10];	调整数长度	1byte 10byte

2	结构体型	结构体成员的调整数中的最大值	成员的长度和
			参照 “(a) 结构体数据的分配方法 “
				struct {
				char a,b;
				};
					调整数长度	1byte
						2byte

3	联合体型	联合体成员的调整数中的最大值	成员的最大长度
			参照 “(b) 联合体数据的分配方法”
				union {
				char a,b;
				};
					调整数长度	1byte
						1byte

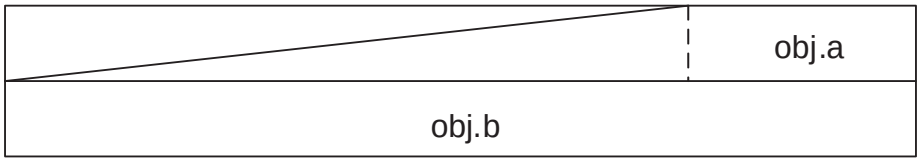
4	类（class）型	1. 当有虚拟函数时：总是为 4
		2. 上述以外：
		数据成员的调整数中的最大值
			数据成员、指向虚拟函数表的指针、指向虚拟基类的指针的和
			参照 “(c) 类数据的分配方法”
				class B: public A{
				virtual void f();
				};
				
				class A{
				char a;
				};
					调整数长度
					
					调整数长度
						4byte
						8byte
						
						1byte
						1byte

在下例中没有标明长度的 □ 表示 4 字节， ／ 表示空区域。
地址从右到左增加 （左侧为高位地址）。

(a) 结构体数据的分配方法

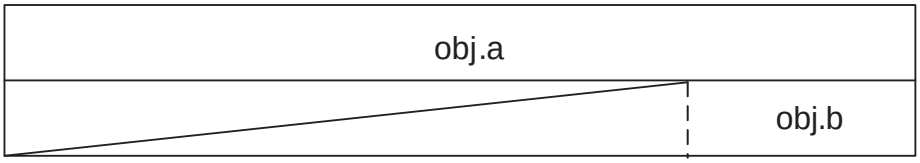
在分配结构体型的各成员时，为了符合该成员型名的边界调整数，有可能在和前一个成员之间产生空区域。
例：

```
struct {
  char a;
  int b;
  } obj;
```



在结构体有 4 字节调整数并且最后的成员以第 1 个字节、第 2 个字节或者第 3 个字节结束时，作为结构体型的区域进行处理，还包括其后的字节。
例：

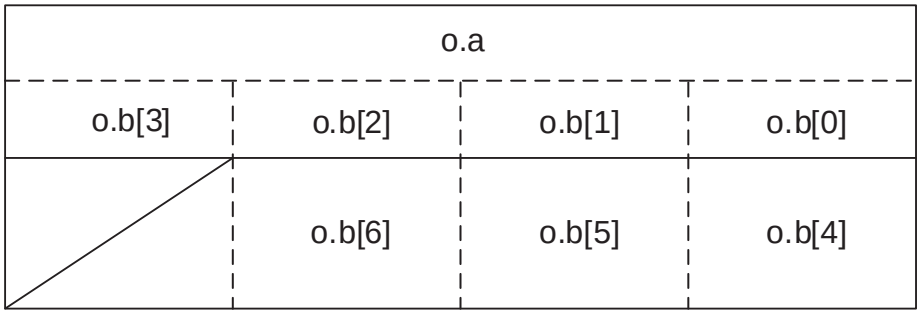
```
struct {
  int a;
  char b;
} obj;
```



(b) 联合体数据的分配方法

在联合体有 4 字节调整数并且成员的最大长度不是 4 的倍数时，作为联合体型的区域进行处理，还包括成为 4 的倍数为止的剩余字节。
例：

```
union {
  int a;
  char b[7];
} o;
```

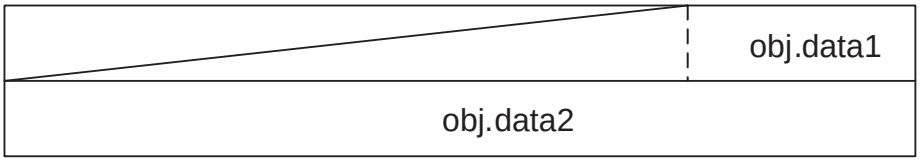


(c) 类数据的分配方法

如果是没有基类和虚拟函数的类，就按照结构体数据的分配规则，分配数据成员。

例：

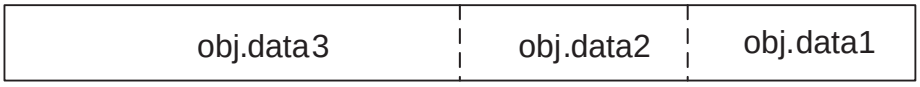
```
class A{
   char data1;
   int data2;
 public:
   A();
   char getData1(){return data1;}
}obj;
```



在从调整数为 1 的基类派生的类的第一个成员为 1byte 数据时，分配不产生空区域的数据成员。

例：

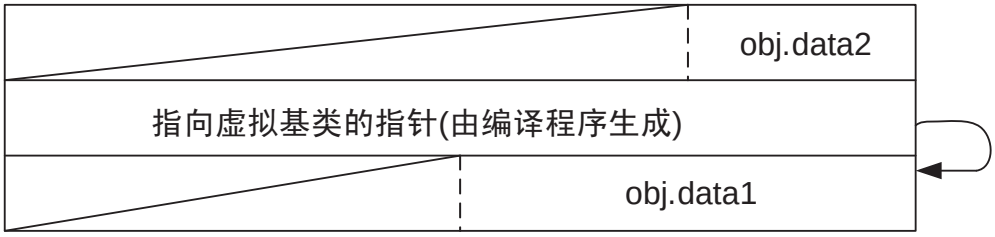
```
class A{
   char data1;
};
class B:public A{
   char data2;
   short data3;
}obj;
```



当类中有虚拟基类时，分配指向虚拟基类的指针。

例：

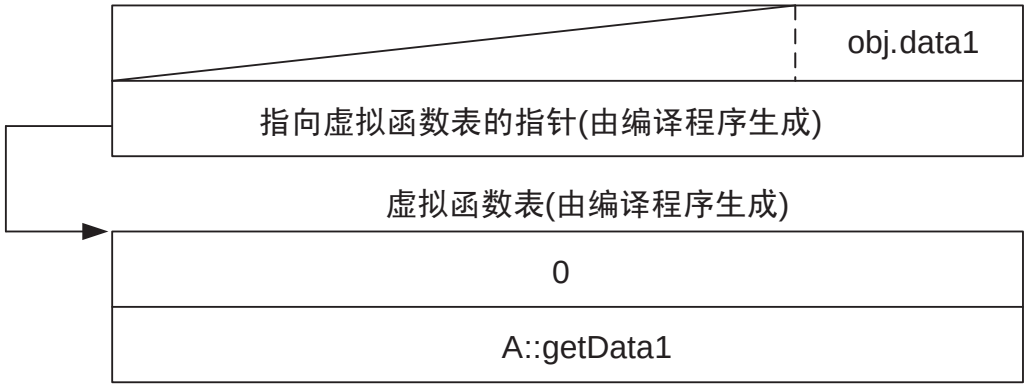
```
class A{
   short data1;
};
class B: virtual protected A{
   char data2;
}obj;
```



当类中有虚拟函数时，编译程序生成虚拟函数表，分配指向虚拟函数表的指针。

例：

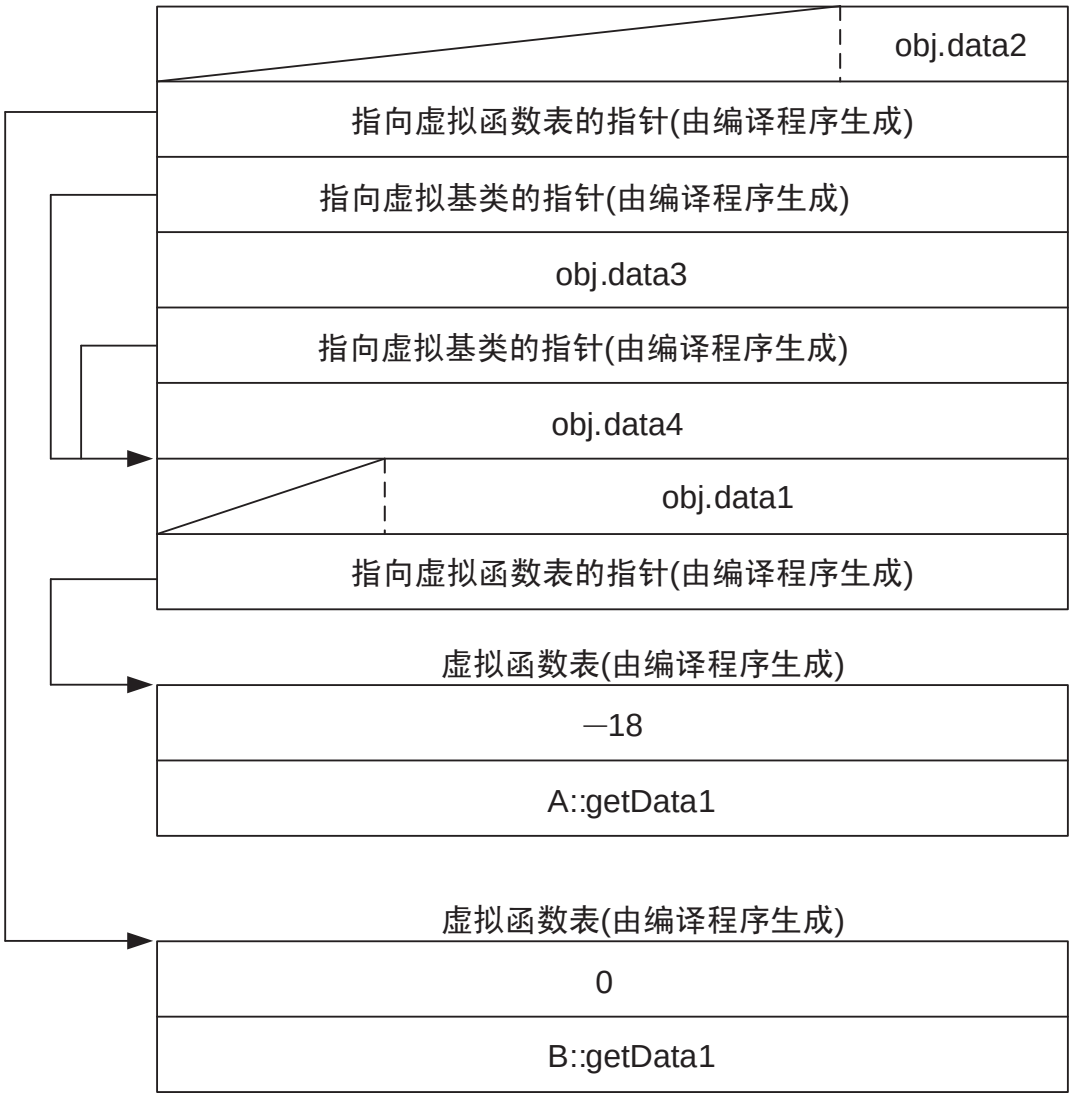
```
class A{
   char data1;
public:
   virtual char getData1();
}obj;
```



有虚拟基类、基类、虚拟函数的类的例子如下所示：

例：

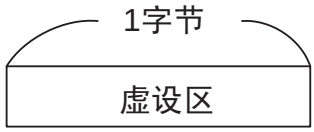
```
class A{
   char data1 ;
   virtual char getData1();
};
class B:virtual public A{
   char data2;
   char getData2();
   char getData1();
};
class C:virtual protected A{
   int data3;

};
class D:virtual public A,public B,public C{
  public:
   int data4;
   char getData1();
}obj;
```


obj.data2
指向虚拟函数表的指针(由编译程序生成)
指向虚拟基类的指针(由编译程序生成)
obj.data3
指向虚拟基类的指针(由编译程序生成)
obj.data4
obj.data1
指向虚拟函数表的指针(由编译程序生成)
虚拟函数表(由编译程序生成)
—18
A::getData1
虚拟函数表(由编译程序生成)
0
B::getData1

[1] 在空类的情况下，分配 1 字节的虚设区。

例：

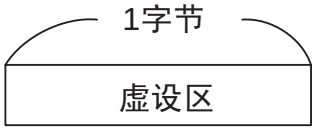
```
class A{
   void fun();
}obj;
```



即使空类有基类的空类，虚设区也为 1 字节。

例：

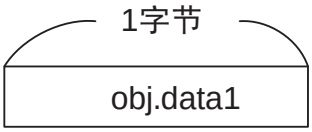
```
class A{
   void fun();
};
class B: A{
   void sub();
}obj;
```



在类长度为 0 时，分配空类的虚设区。在基类或者派生类有数据成员或者有虚拟函数的类时，不分配虚设区。

例：

```
class A{
   void fun();
};
class B: A{
   char data1;
}obj;
```



(3) 位域

位域是通过在结构体、联合体和类中指定位宽进行分配的成员。
本项说明位域特有的分配规则。

(a) 位域的成员

位域成员的规格如表 9.17 所示。

表 9.17 位域成员的规格

	项目	规格
1	位域中允许的型说明符	(unsigned)char、 signed char、 bool*1、 (unsigned)short、 signed short、 enum、 (unsigned)int、 signed int、 (unsigned)long、 signed long、(unsigned)long long、signed long long
2	扩展为声明的型时的符号处理 *2	无符号 （unsigned）的型进行零扩展 *3。 有符号 （signed）的型进行符号扩展 *4。
3	未指定符号的型的符号型	无符号 （unsigned） 如果指定 signed_bitfield 选项，就为带符号（signed）的型。
4	enum 型的符号型	有符号 (signed) 如果指定 auto_enum 选项，就取决于结果的型。

【注】 *1 只能在 C++ 程序和 C99 程序中指定 bool。

*2 在使用位域的成员时，必须先将保存在位域的数据扩展为声明的型。将带符号 （signed）声明的长度为 1 位的位域数据解释为符号。因此，能表示的值只有 0 和 –1。

*3 零扩展：在扩展时，给高位补 0。

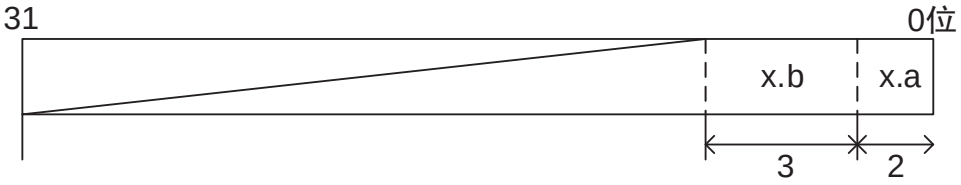
*4 符号扩展：在扩展时，将位域数据的最高位解释为符号，给高于数据的位全部补符号位。

(b) 位域的分配方法

根据以下 5 个规则分配位域：

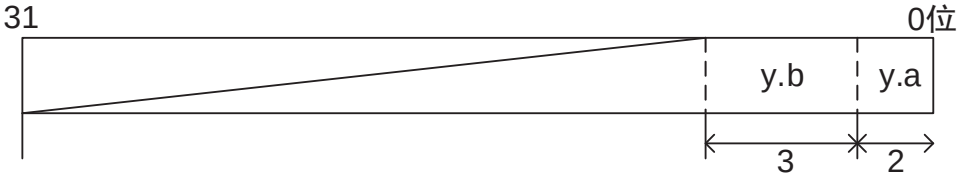
- 在区域内从右 （低位侧）开始按顺序分配位域的成员。
  例：

```
struct b1 {
  int a:2;
  int b:3;
} x;
```



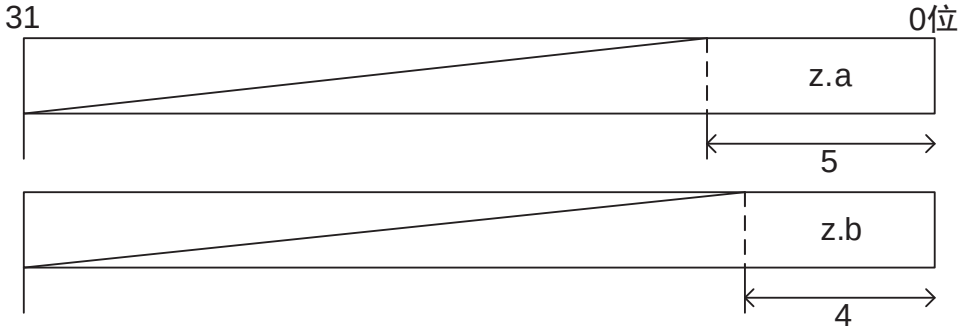
- 在连续有相同长度的型说明符时，尽可能分配到相同区域。
  例：

```
struct b1 {
  long            a:2;
  unsigned int    b:3;
} y;
```



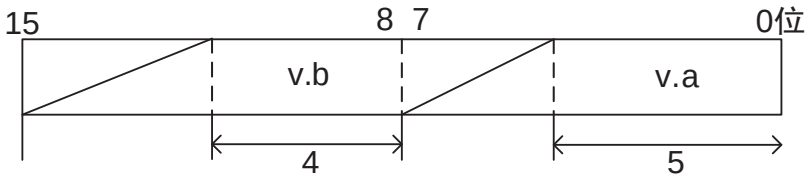
- 将用不同长度的型说明符声明的成员分配到下一个区域。
  例：

```
struct b1 {
  int             a:5;
  char            b:4;
} z;
```



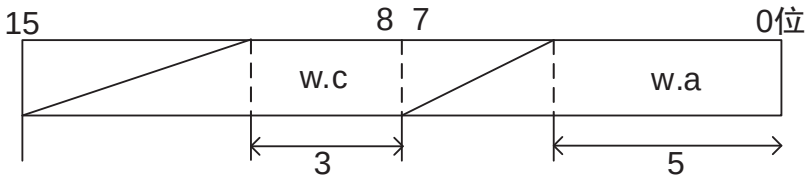
- 如果分配区域的剩余位短于下一个位域的长度，即使有连续的相同长度的型说明符，也分配到下一个区域，而剩余的区域为未使用区域。
  例：

```
struct b2 {
  char a:5;
  char b:4;
} v;
```



- 如果指定位宽为0的位域成员，就将后面的成员强制分配到下一个区域。
  例：

```
struct b2 {
  char a:5;
  char  :0;
  char c:3;
} w;
```



【备注】　也能从高位侧开始分配位域成员，详细内容请参照“2. C/C++ 编译程序的选项”的 bit_order 选项和“9.2.1　#pragma”的 #pragma bit_order。

(4) Big Endian 的存储器分配

Big Endian 的存储器的数据排列如下：

(a) 1 字节数据 （(signed)char、 unsigned char 型）

无论是 Little Endian 还是 Big Endian， 1 字节数据中的位排列顺序相同。

(b) 2 字节数据 （(signed)short、 unsigned short 型）

在 Little Endian 和 Big Endian 中， 2 字节数据中的高位字节和低位字节的字节排列顺序相反。

例

地址 0x100 有 2 字节数据 0x1234 的情况

Little Endian:	0x100 地址：0x34	Big Endian:	0x100 地址：0x12
	0x101 地址：0x12		0x101 地址：0x34

(c) 4 字节数据 （(signed)int、 unsigned int、 (signed)long、 unsigned long、 float、 bool 型）

在 Little Endian 和 Big Endian 中， 4 字节数据中的字节排列顺序相反。

例

地址 0x100 有 4 字节数据 0x12345678 的情况

Little Endian:	地址 0x100：0x78	Big Endian:	地址 0x100：0x12
	地址 0x101：0x56		地址 0x101：0x34
	地址 0x102：0x34		地址 0x102：0x56
	地址 0x103：0x12		地址 0x103：0x78

(d) 8 字节数据 （(signed)long long、 unsigned long long、 double 型）

在 Little Endian 和 Big Endian 中， 8 字节数据中的字节排列顺序相反。

例

地址 0x100 有 8 字节数据 0x0123456789abcdef 的情况

Little Endian:	地址 0x100：0xef	Big Endian:	地址 0x100：0x01
	地址 0x101：0xcd		地址 0x101：0x23
	地址 0x102：0xab		地址 0x102：0x45
	地址 0x103：0x89		地址 0x103：0x67
	地址 0x104：0x67		地址 0x104：0x89
	地址 0x105：0x45		地址 0x105：0xab
	地址 0x106：0x23		地址 0x106：0xcd
	地址 0x107：0x01		地址 0x107：0xef

(e) 结构体 / 联合体、类型数据

结构体 / 联合体、类 （class）型数据的各成员的分配和 Little Endian 时相同。但是，根据该数据的长度规则，各成员的字节排列顺序相反。

例

当地址 0x100 中有

```
struct {
   short a;
   int b;
}z = {0x1234, 0x56789abc};
```

时，

Little Endian:	地址 0x100：0x34	Big Endian:	地址 0x100：0x12
	地址 0x101：0x12		地址 0x101：0x34
	地址 0x102：空区域		地址 0x102：空区域
	地址 0x103：空区域		地址 0x103：空区域
	地址 0x104：0xbc		地址 0x104：0x56
	地址 0x105：0x9a		地址 0x105：0x78
	地址 0x106：0x78		地址 0x106：0x9a
	地址 0x107：0x56		地址 0x107：0xbc

(f) 位域

位域的各区域的分配也和 Little Endian 时相同。但是，根据该数据的长度规则，各区域的字节排列顺序相反。

例

当地址 0x100 中有

```
struct {
   long a:16;
   unsigned int b:15;
   short c:5;
}y={1,1,1};
```

时，

Little Endian:	地址 0x100：0x01	Big Endian:	地址 0x100：0x00
	地址 0x101：0x00		地址 0x101：0x01
	地址 0x102：0x01		地址 0x102：0x00
	地址 0x103：0x00		地址 0x103：0x01
	地址 0x104：0x01		地址 0x104：0x00
	地址 0x105：0x00		地址 0x105：0x01
	地址 0x106：空区域		地址 0x106：空区域
	地址 0x107：空区域		地址 0x107：空区域

9.1.3 浮点型的规格

(1) 浮点型的内部表示

编译程序处理的浮点型的内部表示符合 IEEE 的格式。在此阐述 IEEE 格式的浮点型内部表示的概要。

本节对指定 dbl_size=8 选项的内容进行说明。如果指定 dbl_size=4 选项， double 型、 long double 型的内部表示就和 float 型相同。

(a) 内部表示的格式

用 IEEE 的单精度格式 （32 位）表示 float 型，用 IEEE 的双精度格式 （64 位）表示 double 型和 long double 型。

(b) 浮点数据的格式

float 型、 double 型和 long double 型浮点数据的格式如图 9.1 所示。

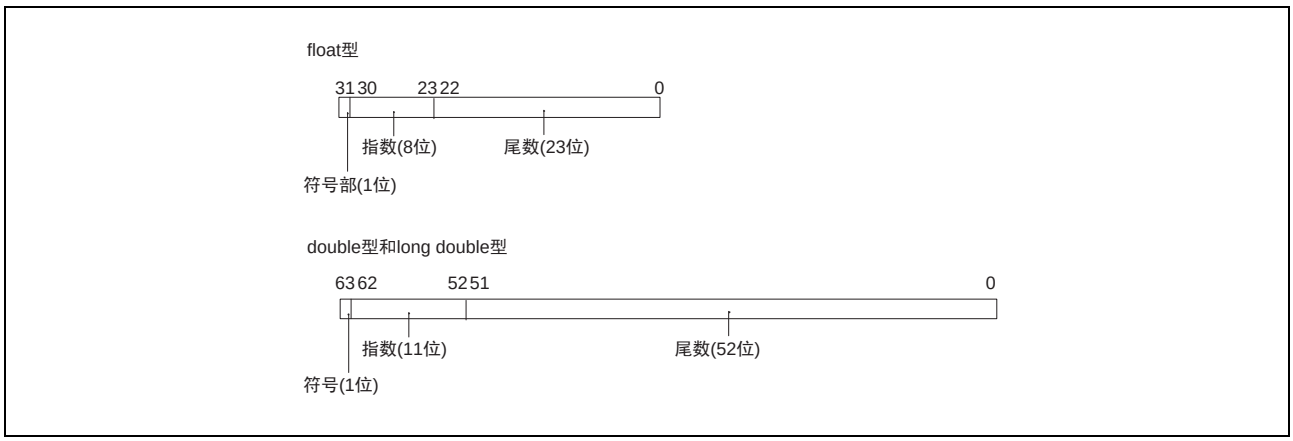


图 9.1 浮点数据的格式

内部表示的各结构要素的含义如下所示：

(i) 符号

表示浮点型的符号。 0表示正， 1表示负。

(ii) 指数

用2的乘方表示浮点型的指数。

(iii) 尾数

这是与浮点型的有效数字对应的数据。

(c) 表示的值的种类

除一般的实数值以外，浮点型还能表示无穷大等的值。浮点型表示的值的种类如下所示：

(i) 规化数

当指数不是0或者指数的全部位都不是1时，表示一般的实数值。

(ii) 非正规化数

当指数是0而尾数不是0时，表示绝对值小的实数值。

(iii) 零

当指数和尾数都是0时，表示0.0的值。

(iv) 无穷大

当指数的全部位是1并且尾数是0时，表示无穷大。

(v) 非数值

当指数的全部位是1而尾数不是0时，表示得到 “0.0/0.0”、“∞/∞”、“∞–∞”等不对应数值的运算结果。

决定浮点型表示值的条件如**表 9.18** 所示。

表 9.18　浮点型表示值的种类

尾数	指数		
	0	不是 0 并全部位都不是 1	全部位是 1
0	0	正规化数	无穷大
非 0 值	非正规化数		非数值

【注】 非正规化数表示正规化数无法表示的范围内的绝对值小的浮点型，但是有效位数比正规化数少。因此，如果运算结果或者中途结果为非正规化数，就无法保证结果的有效位数。
当指定denormalize=off时，将非正规化数作为0进行处理。
当指定denormalize=on时，将非正规化数仍作为非正规化数进行处理。

(2) float 型

float 型的内部表示由 1 位符号、 8 位指数和 23 位尾数组成。

(i) 规化数

符号是0 （正）或者1 （负），表示值的符号。

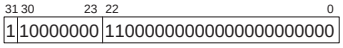
指数是1～254 （2^8–2）的值。实际指数是减去127后的值，其范围为–126～127。

尾数是0～2^{23}–1的值。实际尾数假设2^{23}的位是1，后面部分解释为有小数点的尾数。

正规化数的表示值为：

$(-1)^{<符号>} \times 2^{<指数>-127} \times (1 + <尾数> \times 2^{-23})$

例：



符号：–

指数：10000000 (2) –127=1

尾数：1.11 (2) =1.75

值　：–1.75× 2^1=–3.5

(2) 表示2进制数。

(ii) 非正规化数

符号是0（正）或者1（负），表示值的符号。

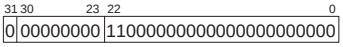
指数是0，实际指数是–126。

尾数是1～2^{23}–1的值，实际尾数假设2^{23}的位是0，后面部分解释为有小数点的尾数。

非正规化数的表示值为：

$(-1)^{<符号>} \times 2^{-126} \times (<尾数> \times 2^{-23})$

例：



符号：+

指数：–126

尾数：$0.11_{(2)} = 0.75$

值　：0.75×2^{-126}

(2) 表示2进制数。

(iii) 零

符号部是0（正）或者1（负），分别表示+0.0和–0.0。

指数和尾数都是0。

+0.0和–0.0都表示为值0.0。有关因零的符号而引起各运算中的不同功能，请参照 **“9.1.3(4) 浮点运算的规格”**。

(iv) 无穷大

符号是0（正）或者1（负），分别表示+∞和–∞。

指数是255(2^8–1)。

尾数是0。

(v) 非数值

指数是255(2^8–1)。

尾数是非0值。

【注】 尾数最高位是 0 的非数值称为 qNaN，尾数最高位是 1 的非数值称为 sNaN。对其他尾数域和符号域的值没有规定。

(3) double 型和 long double 型

double 型和 long double 型的内部表示由 1 位符号、 11 位指数和 52 位尾数组成。

(i) 正规化数

符号部是0（正）或者1（负），表示值的符号。

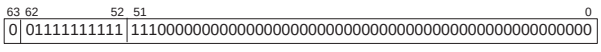
指数是1～2046(2^{11}–2)的值。实际指数是减去1023后的值，其范围为–1022～1023。

尾数是0～2^{52}–1的值。实际尾数假设2^{52}的位是1，后面部分解释为有小数点的尾数。

正规化数的表现值为：

$(-1)^{<符号>} \times 2^{<指数>-1023} \times (1 + <尾数> \times 2^{-52})$

例：



符号：+

指数：$1111111111_{(2)}$–1023=0

尾数：$1.111_{(2)} = 1.875$

值　：$1.875 \times 2^0 = 1.875$

(2) 表示2进制数。

(ii) 非正规化数

符号是0（正）或者1（负），表示值的符号。

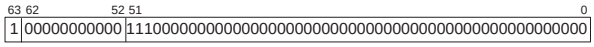
指数是0，实际指数为–1022。

尾数是1～2^{52}–1，实际尾数假设2^{52}的位是0，后面部分解释为有小数点的尾数。

非正规化的表示值为

$(-1)^{<符号>} \times 2^{-1022} \times (<尾数> \times 2^{-52})$

例：



符号：–

指数：–1022

尾数：$0.111_{(2)} = 0.875$　　(2) 表示2进制数。

值　：0.875×2^{-1022}

(iii) 零

符号是0（正）或者1（负），分别表示+0.0和–0.0。

指数和尾数都是0。

+0.0和–0.0都表示为值0.0。有关因零的符号而引起各运算的不同功能，请参照“9.1.3(4) 浮点运算的规格”。

(iv) 无穷大

符号是0（正）或者1（负），分别表示+∞和–∞。

指数是2047(2^{11}–1)。

尾数是0。

(v) 非数值

指数是2047(2^{11}–1)。

尾数是非0值。

【注】 尾数的最高位是 0 的非数值称为 qNaN，尾数的最高位是 1 的非数值称为 sNaN，对其他尾数域的值和符号域的值没有规定。

(4) 浮点运算的规格

本项说明作为 C/C++ 语言功能表示的浮点四则运算以及在编译时或者处理 C 库函数时产生的浮点 10 进制表示和内部表示之间的转换规格。

(a) 四则运算的规格

(i) 结果值的舍入方法

如果浮点四则运算结果的正确值超出内部表示的尾数有效数字，就按照以下规则进行舍入：

[1] 结果值向近似该值的2个浮点型内部表示中较近的方向舍入。

[2] 结果值在近似该值的2个浮点型的正中央时，向尾数的最后位为0的方向舍入。

(ii) 上溢、下溢和无效运算时的处理

对执行时的上溢、下溢和无效运算进行以下处理：

[1] 在上溢时，根据结果的符号，为正无穷大或者负无穷大。

[2] 在下溢时，为非正规化数。

[3] 如果将符号相反的无穷大相加，将符号相同的无穷大相减，零乘无穷大，零除零或者无穷大除无穷大，就产生无效运算。

此时，结果为非数值。

[4] 如果在从浮点型转换为整数时发生上溢，就无法保证结果值。

【注】 在编译时进行常数表达式的运算。如果此时检测到上溢、下溢或者无效运算，就发生警告级错误。

(iii) 有关特殊值（零、无穷大、非数值）运算的注意事项

[1] 正零和负零的和为正零。

[2] 同符号零的差为正零。

[3] 在操作数的单方或者双方含有非数值时，其运算结果总是为非数值。

[4] 在比较运算中，正零和负零相同。

[5] 在操作数的单方或者双方为非数值时，其比较运算或者等同运算的结果对于 “!=” 总是真，而对于其他总是伪。

(b) 10 进制表示和内部表示之间的转换

本项说明有关源程序中的浮点常数和内部表示之间的转换规格或者 C 库函数进行的 ASCII 字符串的浮点型 10 进制表示和内部表示之间的转换规格。

(i) 在从 10 进制表示转换为内部表示时，先将 10 进制表示转换为 10 进制表示的标准形。
10进制表示的标准形为 “$\pm M \times 10^{\pm N}$”， M、 N的范围如下：

[1] float型的标准形
$0 \leqslant M \leqslant 10^9 - 1$
$0 \leqslant N \leqslant 99$

[2] double型和long double型的标准形
$0 \leqslant M \leqslant 10^{17} - 1$
$0 \leqslant N \leqslant 999$

不能转换为标准形的 10 进制表示会发生上溢或者下溢。如果 10 进制表示含有多于标准形的有效数字，就舍去低位。此时，在编译时发生警告级错误，在执行时将对应的错误号设定到变量 errno。
为了转换为标准形，原来 10 进制表示的 ASCII 字符串的长度不能超过 511 个字符，否则就会在编译时发生错误，在执行时将对应的错误号设定到变量 errno。
在从内部表示转换为 10 进制表示时，先转换为 10 进制表示的标准形，然后根据指定的格式转换为 ASCII 字符串。

(ii) 10 进制表示的标准形和内部表示之间的转换
当指数太大或者太小时，无法进行10进制表示的标准形和内部表示之间的转换。以下说明有关能正确转换的范围及其范围外的误差极限值。

[1] 能正确转换的范围
对于以下所示的指数范围的浮点型，能正确进行 “(a)(i) **结果值的舍入**”所示的舍入。在此范围内不发生上溢或者下溢。
float型：$0 \leqslant M \leqslant 10^9 - 1$， $0 \leqslant N \leqslant 13$
double型和long double型：$0 \leqslant M \leqslant 10^{17} - 1$， $0 \leqslant N \leqslant 27$

[2] 误差的极限值
转换超出[1]所示范围值时的误差和正确舍入时的误差的差不能超过有效数字的最少位数的0.47倍。
如果超出[1]所示的范围，就可能在转换时出现上溢或者下溢。此时，在编译时发生警告级错误，在执行时将对应的错误号设定到变量errno。

9.1.4 运算符的判断顺序

在表达式中有多个运算符的情况下，由优先级以及用“右”或者“左”表示的结合性决定这些运算符的判断顺序。

各运算符的优先级和结合性如表 9.19 所示。

表 9.19 运算符的优先级和结合性

优先级	运算符	结合性	适用的表达式
1	++ -- (后缀) ( ) [ ] -> .	左	后缀表达式
2	++ -- (前缀) ! ~ + - * & sizeof	右	单项表达式
3	(型名)	右	数据型转换表达式
4	* / %	左	乘除法表达式
5	+ –	左	加减法表达式
6	<< >>	左	以位为单位的移位表达式
7	< <= > >=	左	关系表达式
8	== !=	左	相等表达式
9	&	左	以位为单位的 AND 表达式
10	^	左	以位为单位的 XOR 表达式
11	|	左	以位为单位的 OR 表达式
12	&&	左	逻辑 AND 表达式
13	||	左	逻辑 OR 表达式
14	?:	左	条件表达式
15	= += –= *= /= %= <<= >>= &= |= ^=	右	赋值表达式
16	,	左	逗点表达式

9.2 扩展功能

编译程序支持以下的扩展功能：

- #pragma、关键字
- 内部函数
- 段地址运算符

9.2.1 #pragma 和关键字

#pragma 和关键字的一览表如**表 9.20** 所示。

与优化有关的 #pragma 有可能因条件而不适用，必须通过输出代码确认该优化是否适用。

表 9.20 #pragma 和关键字的一览表

	对象	#pragma 扩展名	功能
1	存储器分配	#pragma section	指定段的转换。
2		#pragma stacksize	建立堆栈段。
3	函数	#pragma interrupt	建立中断函数。
4		#pragma inline #pragma noinline	指定函数的 inline 展开。
5		#pragma inline_asm	进行汇编记述函数的 inline 展开。
6		#pragma entry	建立入口函数。
7		#pragma option	指定函数单位的选项。
8	其他	#pragma bit_order	指定位域的排列顺序。
9		#pragma pack #pragma unpack #pragma packoption	指定结构体成员和类成员的边界调整数。
10		#pragma address	给变量指定绝对地址。
11		#pragma endian	指定初始值的字节序。
12		__evenaccess	保证变量型长度的存取。
13		far *1 _far *1 near *1 _near *1	保留的关键字

【注】 *1 far,_far、 near 和 _near 作为关键字被保留。
虽然识别为限定词，但是不影响代码的生成。

段转换的指定

#pragma section

格　式　　#pragma section [< 段种类 >] [△ < 变更的段名 >]
< 段种类 >: { P | C | D | B }

说　明　　转换编译程序输出的段名。
当指定段种类和变更的段时，如果段种类为 P，就更改在该 #pragma 声明后记述的函数段名。
如果段种类为 C、 D 或者 B，就更改在该 #pragma 声明后定义了实体的全部段名。
如果只指定变更的段名，就更改该 #pragma 声明后的程序区、常数区、初始化数据区和未初始化数据区的全部段名。此时，各段被更改后的段名为在各默认段名后追加了 < 变更的段名 > 的字符串。
如果没有记述段种类和变更的段名，就将在该 #pragma 声明后的程序区、常数区、初始化数据区和未初始化数据区的全部段名恢复为默认段名。
如果指定 section 选项，各段种类的默认段名就遵循 section 选项的指定，否则就使用段的种类名。

例 1　指定段名和段种类的情况

```
#pragma section B Ba
int i;                          // 分配到 Ba 段。
void func(void)
{
( 省略 )
}

#pragma section B Bb
int j;                          // 分配到 Bb 段。
void sub(void)
{
( 省略 )
}
```

例 2 省略段种类的情况

```
#pragma section abc
int a;                          // 分配到 Babc 段。
const int c=1;                  // 分配到 Cabc 段。
void f(void)                    // 分配到 Pabc 段。
{
    a=c;
}

#pragma section
int b;                          // 分配到 B 段。
void g(void)                    // 分配到 P 段。
{
    b=c;
}
```

备 注 必须在函数定义的外面声明 #pragma section。
不能更改以下项目的段名，必须使用 section 选项。
(1) 字符串字面常量
(2) switch 语句的转移表
每个文件的 #pragma section 能指定的最多段数为 2045 个。
在指定静态类成员变量的段的情况下，需要给类的成员声明和实体的定义都指定 #pragma section。

例

```
 /*
 ** 类成员声明
 */

 class A
 {
            private:

           // 无初始值。
           #pragma section DATA
static int data_;
#pragma section
// 有初始值。
#pragma section TABLE
static int table_[2];
#pragma section
 };
```

```
/*
** 实体定义
*/

// 无初始值。
#pragma section DATA
int A::data_;
#pragma section

// 有初始值。
#pragma section TABLE
int A::table_[2]={0, 1};
#pragma section
```

堆栈段的建立

#pragma stacksize

格　式　　#pragma stacksize {si=< 常数 > | su=< 常数 >}

说　明　　当指定 si=< 常数 > 时，建立用作段名为 SI、长度为 < 常数 > 的堆栈的数据段。
当指定 su=< 常数 > 时，建立用作段名为 SU、长度为 < 常数 > 的堆栈的数据段。

例　　C 源：

```
#pragma stacksize si=100
#pragma stacksize su=200
```

代码的展开例子：

```
.SECTION    SI,DATA,ALIGN=4
.BLKB       100
.SECTION    SU,DATA,ALIGN=4
.BLKB       200
```

备　注　　只能在文件内各指定 1 次 si 和 su。
< 常数 > 必须指定 4 的倍数。

中断函数的建立

#pragma interrupt

格　式　　#pragma interrupt [()< 函数名 >[(＜中断规格＞ [,…]==[,…][])

说　明　　用 #pragma interrupt 声明中断函数。
能给函数名指定全局函数成员和静态函数成员。
中断规格一览表如**表 9.21** 所示。

表 9.21　中断规格一览表

	项目	格式	选项	指定的内容
1	向量表的指定	vect=	< 向量号 >	分配中断函数地址的向量号。
2	高速中断的指定	fint	无	指定用于高速中断的函数。 用 RTFI 指令返回。
3	中断函数寄存器的限制指定	save	无	限制中断函数内使用的寄存器个数，减少保存和恢复的次数。
4	多重中断允许的指定	enable	无	在函数的起始位置将 PSW 的 I 标志置 1，允许多重中断。

用 #pragma interrupt 声明的函数在处理函数前后保证全部寄存器 （在函数出口 / 入口，保存和恢复函数内使用的全部寄存器），通常用 RTE 指令返回。
当不指定中断规格时，作为单纯的中断函数进行处理。
当指定向量表 （vect=）时，将该函数地址设定到 C$VECT 段内指定的向量表号的位置。
当指定高速中断 （fint）时，用 RTFI 指令返回。当指定 fint_register 选项时，不保存和恢复选项指定的寄存器而在中断函数中使用。
当限制中断函数寄存器 （save）时，将中断函数内使用的寄存器个数限制在 R1 ～ R5 和 R14 ～ R15。因为 R6 ～ R13 不用于中断，所以不生成保存指令和恢复指令。
当允许多重中断 （enable）时，在中断函数的起始位置将 PSW 的 I 标志置 1，允许多重中断。
在定义中断函数时，能指定的函数是全局函数 （C/C++ 语言）成员和静态函数成员 （C++ 语言）。
函数返回值的型只有 void。不能指定 return 语句的返回值，否则就输出错误。

例 1　正确声明和错误声明的例子

```
#pragma interrupt (f1, f2)
void f1(){…}          // 这是正确的声明。
int f2(){…}           // 因为返回值的型不是 void，所以是错误的声明。
```

例 2 一般的中断函数例子

C 源：

```
#pragma interrupt func
void func(){ .... }
```

输出代码：

```
_func:
        PUSHM   R1-R3       ; 保存函数内使用的寄存器。
        ....
        ( 在函数内使用 R1、 R2 和 R3。 )
        ....
        POPM    R1-R3       ; 恢复在入口保存的寄存器。
        RTE
```

例 3 有函数调用的中断函数例子

除函数内使用的寄存器以外，还对函数调用前后不保证的寄存器，在中断函数的入口进行保存以及在出口进行恢复。

C 源：

```
#pragma interrupt func
void func(){
    ....
    sub();
    ....
  }
```

输出代码：

```
_func:
        PUSHM   R1-R5       ; 保存 R1 ～ R5。
        PUSHM   R14-R15     ; 保存 R14、 R15。
        ....
        MOV.L   #_sub,R15
        JSR     R15         ; 进行函数调用。
        ....
        POPM    R14-R15     ; 恢复 R14、 R15。
        POPM    R1-R5       ; 恢复 R1 ～ R5。
        RTE
```

例 4 使用中断规格 fint 时的例子

C 源：指定 fint_register=2 选项进行编译。

```
#pragma interrupt func(fint)
void func1(){ .... } // 中断函数
void func2(){ .... } // 一般函数
```

输出代码：

```
_func1:
        PUSHM   R1-R3       ; 保存函数内使用的寄存器。
        ....                ; （但是不保存 R12、 R13。）
        ....
        （在函数内使用 R1、 R2、 R3、 R12、 R13。）
        ....
        POPM    R1-R3       ; 恢复在入口保存的寄存器。
        RTE

_func2:
        ....                ; 对于不用 #pragma interrupt fint 指定的函数，
        ....                ; 生成不使用 R12、 R13 的代码。
```

备 注 因为有可能在优化时被删除，所以不能指定 static 函数。

函数的 inline 展开

#pragma inline, #pragma noinline

格　式　　#pragma inline [(]< 函数名 >[,…][)]
#pragma noinline [(]< 函数名 >[,…][)]

说　明　　#pragma inline 声明要进行 inline 展开的函数。
即使指定 noinline 选项， #pragma inline 指定的函数也为 inline 展开的对象。
#pragma noinline 声明要抑制 inline 选项指定的函数。
能给函数名指定全局函数成员和静态函数成员。
对于用 #pragma inline 指定函数名的函数以及指定函数说明符 inline （C++ 语言和 C(C99) 语言）的函数，在调用此函数的位置进行 inline 展开。

例　　源文件

```
#pragma inline(func)
static int func (int a, int b)
{
    return (a+b)/2;
}
int x;
main()
{
    x=func(10,20);
}
```

展开信息

```
int x;
main()
{
    int func_result;
    {
        int a_1=10, b_1=20;
        func_result=(a_1+b_1)/2;
    }
     x=func_result;
}
```

备　注　　即使指定 #pragma inline，在以下的任意情况下也不进行 inline 展开。
－有可变参数的函数。
－在函数内参照形式参数的地址。
－通过展开对象函数的地址进行调用。
#pragma inline 不保证 inline 展开。有可能通过考虑编译时间和存储器使用量增大的限制来抑制 inline 展开。在 inline 展开被抑制的情况下，如果指定 noscope 选项，就可能进行 inline 展开。
必须在定义函数前指定 #pragma inline。
对 #pragma inline 指定的函数也生成外部定义。如果在各源文件中记述了 inline 展开对象函数的实体，就必须在声明函数时指定 static。如果指定 static，就不生成外部定义。
指定 inline （C++ 语言和 C （C99）语言）的函数不生成外部定义。

汇编记述函数的 inline 展开

#pragma inline_asm

格　式　　#pragma inline_asm[(]< 函数名 >[,…][)]

说　明　　对 #pragma inline_asm 声明的汇编记述函数进行 inline 展开。
汇编程序的内部 inline 函数的调用规则和一般函数的调用规则相同。

例　　C 源：

```
#pragma inline_asm func
static int func(int a, int b){
        ADD     R2,R1       ; 汇编描述
}
main(int *p){
*p = func(10,20);
}
```

输出代码：

```
_main:
        PUSH.L  R6
        MOV.L   R1,R6
        MOV.L   #20,R2
        MOV.L   #10,R1
        ADD     R2,R1       ; inline 展开
        MOV.L   R1,[R6]
        POP     R6
        RTS
```

备　注　　必须在定义函数前指定 #pragma inline_asm。
对 #pragma inline_asm 指定的函数也生成外部定义。
如果在汇编程序的内部 inline 函数内并且在函数的出口 / 入口使用要保证的寄存器 （参照**表 8.2**），就需要在汇编程序的内部 inline 函数的开头和最后保存和恢复这些寄存器。
只能给汇编程序的内部 inline 函数记述 RX 族的指令和临时标号。
不能在汇编程序的内部 inline 函数的最后记述 RTS。
不能给函数名指定函数成员。
如果给 static 函数指定 #pragma inline_asm，就在 inline 展开后删除函数定义。
汇编记述为预处理器的处理对象。因此，在通过 #define 对和汇编语言使用的指令或者寄存器同名的宏 （例：“MOV”和 “R5”等）进行宏定义时必须注意。

入口函数的建立

#pragma entry

格　式　　#pragma entry[(]< 函数名 >[)]

说　明　　将用 < 函数名 > 指定的函数作为入口函数进行处理。
入口函数不建立任何寄存器的保存代码和恢复代码。
如果有 #pragma stacksize 声明，就在函数的开头输出堆栈指针的初始设定代码。
如果指定 base 选项，就设定选项指定的基址寄存器。

例　　C 源：指定 -base=rom=R13

```
#pragma stacksize su=100
#pragma entry INIT
void INIT() {
:
}
```

输出代码：

```
.SECTION    SU,DATA,ALIGN=4
.BLKB       100
.SECTION    P,CODE
_INIT:
MVTC        (TOPOF SU + SIZEOF SU),USP
MOV.L       #__ROM_TOP,R13
```

备　注　　必须在声明函数前指定 #pragma entry。
不能整个装入模块中指定多个入口函数。

函数单位选项的指定

#pragma option

格　式　　#pragma option [< 选项串 >]

说　明　　用 #pragma option 将选项串指定的选项设定为有效。
指定的选项适用于文件的结束或者没有 < 选项串 > 的设定 #pragma option 的部分。
如果指定 #pragma option < 选项串 >，就进行 < 选项串 > 指定的优化。能使用的优化如表 9.22 所示。有关各优化选项，请参照 “2. C/C++ 编译程序的选项”。

表 9.22　#pragma option 能使用的优化选项

	选项的指定方法	选项的解除方法
1	const_div	noconst_div
2	optimize = {0 | 1 | 2}	无
3	speed size	size speed
4	loop=n　（n 为 2 ～ 32 的整数）	loop=1
5	case={ ifthen | table | auto }	无
6	schedule	noschedule
7	scope	noscope

例　　C 源：未指定编译程序的选项 （默认值）

```
#pragma option speed
void func1(){ ... }          /* 默认值 + –speed 有效。                    */
#pragma option optimize=0
void func2(){ ... }          /* 默认值 + –speed + –optimize=0 有效。      */
#pragma option
void func3(){ ... }          /* 默认值有效。                              */
```

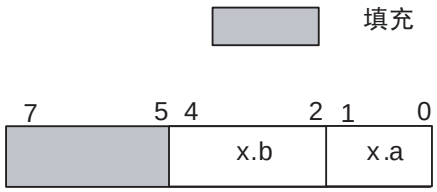
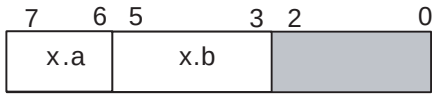
位域排列顺序的指定

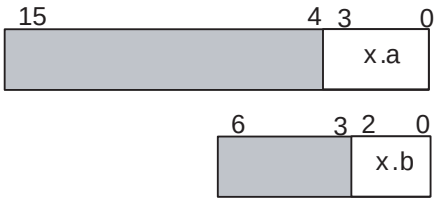
#pragma bit_order

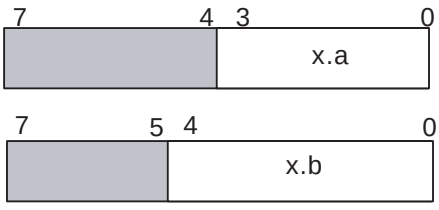
格　式　　#pragma bit_order [{left | right}]

说　明　　指定位域排列顺序的转换。
当指定 left 时，从高位开始分配成员；当指定 right 时，从低位开始分配成员。
默认设定是 right。
如果省略 left|right，以后就以选项为基准。

例

C 源	位的分配
#pragma bit_order right struct tbl_r { 　unsigned char a:2; 　unsigned char b:3; } x;	
#pragma bit_order left struct tbl_l { 　unsigned char a:2; 　unsigned char b:3;} y;	

// 不同长度成员的情况
#pragma bit_order right
struct tbl_r {
　unsigned short a:4;
　unsigned char b:3;
} x;
	

// 超过型长度的情况
#pragma bit_order right
struct tbl_r {
　unsigned char a:4;
　unsigned char b:5;
} x;
	

结构体成员和类成员调整数的指定

#pragma pack
#pragma unpack
#pragma packoption

格　式　　#pragma pack
#pragma unpack
#pragma packoption

说　明　　指定源程序中指定位置后的结构体成员和类成员的调整数。
在没有指定本扩展名时或者在 #pragma packoption 指定位置后声明的结构体成员和类成员的调整数都遵循 pack 选项的指定。 #pragma pack 扩展名和调整数的关系如**表 9.23** 所示。

表 9.23　#pragma pack 和成员的调整数

成员的型	#pragma pack	#pragma unpack	#pragma packoption 或者没有指定
(signed) char	1	1	1
(unsigned) short	1	2	遵循 pack 选项
(unsigned) int*、 (unsigned) long、 (unsigned) long long、浮点型、指针型	1	4	遵循 pack 选项

例　　#pragma pack

```
struct S1 {
    char a;          /* 字节偏移量 =0          */
    int b;           /* 字节偏移量 =1          */
    char c;          /* 字节偏移量 =5          */
} ST1;               /* 总容量为 6 字节。      */

#pragma unpack
struct S2 {
    char a;          /* 字节偏移量 =0          */
                     /* 3 字节空区域           */
    int b;           /* 字节偏移量 =4          */
    char c;          /* 字节偏移量 =8          */
                     /* 3 字节空区域           */
} ST2;               /* 总容量为 12 字节。     */
```

备　注　　不能用指针存取指定 #pragma pack 的结构体成员和类成员 （包括在使用指针的成员函数内的存取）。

例

```
#pragma pack
struct st {
    char x;
    int y;
} ST;
int *p=&ST.y; /* ST.y 的地址有可能为奇数。*/
void func(void) {
    ST.y=1; /* 能正确地存取。*/
    *p=1; /* 有可能无法正确地存取。*/
}
```

也能通过 pack 选项指定结构体、联合体和类成员的调整数。如果同时指定选项和 #pragma，#pragma 的指定就优先。

绝对地址的指定

#pragma address

格　式　　#pragma address [(]< 变量名 >=< 绝对地址 >[,…][)]

说　明　　将指定的变量分配到指定的地址。此时，编译程序按指定的变量设定段，在连接时分配到指定的绝对地址。如果给连续的地址指定变量，就将这些变量分配到同一个段。

例　　C 源：

```
#pragma address X=0x7f00
int X;
main(){
      X=0;
}
```

输出代码：

```
_main:
      MOV.L       #0,R5
      MOV.L       #32512,R14        ; 0x7f00
      MOV.L       R5,[R14]
      RTS
      .SECTION    $ADDR_B_7F00,DATA
      .ORG        7F00H
      .glb        _X
_X:                                 ; static: X
      .blkl       1
```

备　注　　必须在声明变量前指定 #pragma address。
如果指定结构体 / 联合体的成员或者变量以外的符号名，就发生错误。
如果对同一个变量多次指定 #pragma address，就发生错误。
如果对同一个变量同时指定 #pragma section，就发生错误。

初始值字节序的指定

#pragma endian

格　式　　#pragma endian [{big | little}]

说　明　　指定要保存静态目标的区域的字节序。
从记述 #pragma endian 的行到文件的末尾或者在记述下一个 #pragma endian 行前定义的内容为对象。
当指定 big 时，为 big endian。当指定 endian=little 选项时，将数据分配到段名之后加 _B 的段。
当指定 little 时，为 little endian。当指定 endian=big 选项时，将数据分配到段名之后加 _L 的段。
如果省略 big | little，以后就以选项为基准。

例　　当指定 endian=little 选项时 （默认值）
C 源：

```
#pragma endian big
int A=100;      /* D_B 段        */
#pragma endian
int B=200;      /* D 段 */
```

输出代码：

```
                .SECTION        D_B,ROMDATA,ALIGN=4
                .ENDIAN         BIG
                .glb            A
_A:                                             ; static: A
                .LWORD          00000064H
                .SECTION        D,ROMDATA,ALIGN=4
                .glb            _B
_B:                                             ; static: B
                .LWORD          000000C8H
```

备　注　　如果在不同于 endian 选项的 #pragma endian 对象的目标中包含 long long 型、 double 型 （指定 dbl_size=8 选项时）和 long double 型 （同）的区域，就不能使用地址和指针对这些区域进行间接存取，否则无法保证运行。
如果记述取得这些区域地址的代码，就显示警告。
如果在不同于 endian 选项的 #pragma endian 对象的目标中包含 long long 型位域，就不能写此区域，否则无法保证运行。
如果记述写这些区域的代码，就显示警告。
不能更改以下项目的字节序，必须使用 endian 选项。
(1) 字符串字面常量
(2) switch 语句的转移表
(3) 声明外部参照的目标 （没有初始化表达式而进行 extern 声明的目标）

指定长度的存取保证

__evenaccess

格　式　　__evenaccess < 型说明符 > < 变量名 >
< 型说明符 > __evenaccess < 变量名 >

说　明　　保证以变量的型长度进行存取。
保证的对象长度是不超过 4 字节的整数标量型 （signed char、 unsigned char、 signed short、 unsigned short、 signed int、 unsigned int、 signed long、 unsigned long）。

例　　C 源：

```
#pragma address A=0xff0178
unsigned long __evenaccess A;
void test(void)
{
        A &= ~0x20;
}
```

输出代码 （不指定 __evenaccess 时）：

```
_test:
  MOV.L #16712056,R1
  BCLR #5,[R1]        ; 进行 1 字节的存储器存取。
  RTS
```

输出代码 （指定 __evenaccess 时）：

```
_test:
  MOV.L #16712056,R1
  MOV.L [R1],R5       ; 进行 4 字节的存储器存取。
  BCLR  #5,R5
  MOV.L R5,[ R1]      ; 进行 4 字节的存储器存取。
  RTS
```

备　注　　如果指定为结构体或者联合体，就和给全部成员指定 __evenaccess 有相同的效果。此时，保证不超过 4 字节的整数标量型成员的存取长度，但是不保证以结构体或者联合体为单位的存取长度。

9.2.2 内部函数

作为内部函数，提供以下功能：

- 最大值、最小值
- 数据内字节序的变更
- 数据的交换
- 乘加运算
- 循环
- 特殊指令 （BRK、 WAIT、 INT、 NOP）
- BRK、 WAIT等的RX族特殊指令
- 控制寄存器的设定和参照

和一般函数一样，用函数调用格式记述内部函数。
内部函数一览表如**表 9.24** 所示。

表 9.24 内部函数一览表

	项目	规格	功能
1	最大值、最小值	signed long max(signed long data1, signed long data2)	选择最大值。
2		signed long min(signed long data1, signed long data2)	选择最小值。
3	字节序的变更	unsigned long revl(unsigned long data)	将长字数据进行字节颠倒。
4		unsigned long revw(unsigned long data)	将长字数据按字进行字节颠倒。
5	数据的交换	void xchg(signed long *data1, signed long *data2)	交换数据。
6	乘加运算	long long rmpab(long long init, unsigned long count, signed char *addr1, signed char *add2)	进行乘加运算 （字节）。
7		long long rmpaw(long long init, unsigned long count, short *addr1, short *add2)	进行乘加运算 （字）。
8		long long rmpal(long long init, unsigned long count, long *addr1, long *add2)	进行乘加运算 （长字）。
9	循环	unsigned long rolc(unsigned long data)	将包括进位在内的数据左循环 1 位。
10		unsigned long rorc(unsigned long data)	将包括进位在内的数据右循环 1 位。
11		unsigned long rotl(unsigned long data, unsigned long num)	左循环
12		unsigned long rotr (unsigned long data, unsigned long num)	右循环
13	特殊指令	void brk(void)	BRK 指令异常
14		void int_exception(unsigned long num)	INT 指令异常
15		void wait(void)	停止程序执行。
16		void nop(void)	展开为 NOP 指令。
17	处理器中断优先级（IPL）	void set_ipl(unsigned char level)	设定中断优先级。
18		unsigned char get_ipl(void)	参照中断优先级。
19	处理器状态字 （PSW）	void set_psw(unsigned long data)	设定 PSW。
20		unsigned long get_psw(void)	参照 PSW。
21	浮点状态字 （FPSW）	void set_fpsw(unsigned long data)	设定 FPSW。
22		unsigned long get_fpsw(void)	参照 FPSW。

	项目	规格	功能
23	用户堆栈指针 （USP）	void set_usp(unsigned long data)	设定 USP。
24		unsigned long get_usp(void)	参照 USP。
25	中断堆栈指针 （ISP）	void set_isp(unsigned long data)	设定 ISP。
26		unsigned long get_isp(void)	参照 ISP。
27	中断表寄存器 （INTB）	void set_intb (unsigned long data)	设定 INTB。
28		unsigned long get_intb(void)	参照 INTB。
29	备份 PSW （BPSW）	void set_bpsw(unsigned long data)	设定 BPSW。
30		unsigned long get_bpsw(void)	参照 BPSW。
31	备份 PC （BPC）	void set_bpc(unsigned long data)	设定 BPC。
32		unsigned long get_bpc(void)	参照 BPC。
33	高速中断向量寄存器 （FINTV）	void set_fintv(unsigned long data)	设定 FINTV。
34		unsigned long get_fintv(void)	参照 FINTV。

最大值的选择

signed long max(signed long data1, signed long data2)

说　明　　选择 2 个输入值中较大的值 （展开为 MAX 指令）。

头文件　　<machine.h>

参　数　　data1　　输入值 1
data2　　输入值 2

返回值　　data1 和 data2 中较大的值

例

```
#include < machine.h>
extern signed long ret,in1,in2;
void main(void)
{
        ret = max(in1,in2);             // 将 in1 和 in2 中较大的值设定到 ret。
}
```

最小值的选择

signed long min(signed long data1, signed long data2)

说　明　　选择 2 个输入值中较小的值 （展开为 MIN 指令）。

头文件　　<machine.h>

参　数　　data1　　输入值 1
data2　　输入值 2

返回值　　data1 和 data2 中较小的值

例

```
#include < machine.h>
extern signed long ret,in1,in2;
void main(void)
{
        ret = min(in1,in2);             // 将 in1 和 in2 中较小的值设定到 ret。
}
```

将长字数据进行字节颠倒

unsigned long revl(unsigned long data)

说　明　　颠倒 4 字节数据的字节排列顺序 （展开为 REVL 指令）。

头文件　　<machine.h>

参　数　　data　　要颠倒字节排列顺序的数据

返回值　　data 的字节排列顺序颠倒后的值

例

```
#include <machine.h>
extern unsigned long ret,indata=0x12345678;
void main(void)
{
        ret = revl(indata);    // ret=0x78563412
}
```

将长字数据按字进行字节颠倒

unsigned long revw(unsigned long data)

说　明　　将 4 字节数据的高位 2 字节和低位 2 字节分别进行字节排列顺序的颠倒 （展开为 REVW 指令）。

头文件　　<machine.h>

参　数　　data　　要颠倒字节排列顺序的数据

返回值　　data 的高位 2 字节和低位 2 字节分别进行字节排列顺序颠倒后的值

例

```
#include <machine.h>
extern unsigned long ret;indata=0x12345678;
void main(void)
{
        ret = revw(indata);   // ret=0x34127856
}
```

数据的交换

void xchg(signed long *data1, signed long *data2)

说　明　　交换参数指向的区域内容 （展开为 XCHG 指令）。

头文件　　<machine.h>

参　数　　*data1　　输入值 1
*data2　　输入值 2

例

```
#include <machine.h>
extern signed long *in1,*in2;
void main(void)
{
        xchg (in1,in2);       // 交换地址 in1 和地址 in2 的数据。

}
```

乘加运算 （字节）

long long rmpab(long long init, unsigned long count, signed char *addr1, signed char *add2)

说　明　　将初始值设定为 init，次数设定为 count，保存乘数的起始地址设定为 addr1 和 addr2，进行乘加运算 （展开为 RMPA.B 指令）。

头文件　　<machine.h>

参　数　　init　　初始值
count　　乘加运算的次数
*addr1　　乘数 1 的起始地址
*addr2　　乘数 2 的起始地址

返回值　　init + Σ(data1[n] * data2[n]) 的低 64 位的结果 （n=0、 1、…、 const–1）

例

```
#include <machine.h>
extern signed char data1[8],data2[8];
long long sum;
void main(void)
{
        sum=rmpab(0, 8, data1, data2);        // 初始值为 0，在将数组 data1 乘数组
                                              // data2 的结果相加后，将结果设定到 sum。
}
```

备注　　RMPA 指令在 80bit 范围内计算结果，但是本内部函数只在 64bit 范围内进行处理。

乘加运算（字）

long long rmpaw(long long init, unsigned long count, short *addr1, short *add2)

说　明　　将初始值设定为 init，次数设定为 count，保存乘数的起始地址设定为 addr1 和 addr2，进行乘加运算（展开为 RMPA.W 指令）。

头文件　　<machine.h>

参　数

init	初始值
count	乘加运算的次数
*addr1	乘数 1 的起始地址
*addr2	乘数 2 的起始地址

返回值　　init + Σ(data1[n] * data2[n]) 的低 64 位的结果（n=0、1、…、const–1）

例

```
#include <machine.h>
extern signed short data1[8],data2[8];
long long sum;
void main(void)
{
        sum=rmpaw(0, 8, data1, data2);        // 初始值为 0，在将数组 data1 乘数组
                                              // data2 的结果相加后，将结果设定到 sum。
}
```

备　注　　RMPA 指令在 80bit 范围内计算结果，但是本内部函数只在 64bit 范围内进行处理。

乘加运算（长字）

long long rmpal(long long init, unsigned long count, long *addr1, long *add2)

说　明　　将初始值设定为 init，次数设定为 count，保存乘数的起始地址设定为 addr1 和 addr2，进行乘加运算（展开为 RMPA.L 指令）。

头文件　　<machine.h>

参　数

init	初始值
count	乘加运算的次数
*addr1	乘数 1 的起始地址
*addr2	乘数 2 的起始地址

返回值　　init + Σ(data1[n] * data2[n]) 的低 64 位的结果（n=0、1、…、const–1）

例

```
#include <machine.h>
extern signed long data1[8],data2[8];
long long sum;
void main(void)
{
        sum=rmpaw(0, 8, data1, data2);      // 初始值为 0，在将数组 data1 乘数组
                                            // data2 的结果相加后，将结果设定到 sum。
}
```

将包括进位在内的数据左循环 1 位

unsigned long rolc(unsigned long data)

说　明　返回包括 C 标志在内的数据左循环 1 位的结果 （展开为 ROLC 指令）。
将操作数移出的位反映到 C 标志。

头文件　<machine.h>

参　数　data　要左循环的数据

返回值　将包括 C 标志在内的 data 左循环 1 位的结果

例

```
#include <machine.h>
extern unsigned long ret;indata;
void main(void)
{
        ret = rolc(indata);                    // 将包括 C 标志在内的 indata 左循环 1 位并且
                                               // 设定到 ret。
}
```

将包括进位在内的数据右循环 1 位

unsigned long rorc(unsigned long data)

说　明　返回包括 C 标志在内的数据右循环 1 位的结果 （展开为 RORC 指令）。
将操作数移出的位反映到 C 标志。

头文件　<machine.h>

参　数　data　要右循环的数据

返回值　将包括 C 标志在内的 data 右循环 1 位的结果

例

```
#include <machine.h>
extern unsigned long ret;indata;
void main(void)
{
        ret = rorc(indata);                    // 将包括 C 标志在内的 indata 右循环 1 位并且
                                               // 设定到 ret。
}
```

左循环

unsigned long rotl(unsigned long data, unsigned long num)

说　明　返回左循环任意位的结果（展开为 ROTL 指令）。
将操作数移出的位反映到 C 标志。

头文件　<machine.h>

参　数　data　要左循环的数据
num　循环的位数

返回值　将 data 左循环 num 位的结果

例

```
#include <machine.h>
extern unsigned long ret;indata;
void main(void)
{
        ret = rotl(indata, 31);                 // 将 indata 左循环 31 位并且
                                                // 设定到 ret。
}
```

右循环

unsigned long rotr (unsigned long data, unsigned long num)

说　明　返回右循环任意位的结果（展开为 ROTR 指令）。
将操作数移出的位反映到 C 标志。

头文件　<machine.h>

参　数　data　要右循环的数据
num　循环的位数

返回值　将 data 右循环 num 位的结果

例

```
#include <machine.h>
extern unsigned long ret;indata;
void main(void)
{
        ret = rotr(indata, 31);                 // 将 indata 右循环 31 位并且
                                                // 设定到 ret。
}
```

BRK 指令异常

void brk(void)

说　明　　展开为 BRK 指令。

头文件　　<machine.h>

例

```
#include <machine.h>
void main(void)
{
        brk();                          // BRK 指令
}
```

INT 指令异常

void int_exception(unsigned long num)

说　明　　展开为 INT num 指令。

头文件　　<machine.h>

参　数　　num　　INT 指令号

例

```
#include <machine.h>
void main(void)
{
        int_exception(10);              // INT #10 指令
}
```

备　注　　num 能设定的数只有 0 ～ 255 的整数。

程序的停止执行

void wait(void)

说　明　　展开 WAIT 指令。

头文件　　<machine.h>

例

```
#include <machine.h>
void main(void)
{
        wait();                 // WAIT 指令
}
```

展开为 NOP 指令

void nop(void)

说　明　　展开为 NOP 指令。

头文件　　<machine.h>

例

```
#include <machine.h>
void main(void)
{
        nop();                  // NOP 指令
}
```

中断优先级的设定

void set_ipl(unsigned long level)

说　明　　更改中断屏蔽优先级。

头文件　　<machine.h>

返回值　　level　　要设定的中断屏蔽优先级

例

```
#include <machine.h>
void main(void)
{
        set_ipl(7);                    // 给 PSW.IPL 设定 7。
}
```

备　注　　level 的默认值为 0 ～ 15 的值。如果指定 -patch=rx610，就能分别指定 0 ～ 7 的值。
在 level 为常数时，如果指定范围外的值，就发生错误。

中断优先级的参照

unsigned char get_ipl(void)

说　明　　参照中断屏蔽优先级。

头文件　　<machine.h>

返回值　　中断屏蔽优先级

例

```
#include <machine.h>
extern unsigned char level;
void main(void)
{
        level=get_ipl();               // 取得 PSW.IPL 的值并且设定到 level。
}
```

PSW 的设定

void set_psw(unsigned long data)

说　明　　设定 PSW。

头文件　　<machine.h>

参　数　　data　　设定值

例

```
#include <machine.h>
extern unsigned long data;
void main(void)
{
        set_psw(data);              // 将 data 的值设定到 PSW。
}
```

PSW 的参照

unsigned long get_psw(void)

说　明　　参照 PSW。

头文件　　<machine.h>

返回值　　PSW 的值

例

```
#include <machine.h>
extern unsigned long ret;
void main(void)
{
        ret=get_psw();              // 取得 PSW 的值并且设定到 ret。
}
```

备　注　　由于优化的作用，有可能在和 get_psw 的调用位置不同的时序取得 PSW 寄存器的值。如果在进行某些运算后记述利用本函数返回值中所包含的 C、 Z、 S 或者 O 标志的代码，就无法保证运行。

FPSW 的设定

void set_fpsw(unsigned long data)

说　明　　设定 FPSW。

头文件　　<machine.h>

参　数　　data　　设定值

例

```
#include <machine.h>
extern unsigned long data;
void main(void)
{
        set_fpsw(data);              // 将 data 的值设定到 FPSW。
}
```

FPSW 的参照

unsigned long get_fpsw(void)

说　明　　参照 FPSW。

头文件　　<machine.h>

返回值　　FPSW 的值

例

```
#include <machine.h>
extern unsigned long ret;
void main(void)
{
        ret=get_fpsw();              // 取得 FPSW 的值并且设定到 ret。
}
```

备　注　　由于优化的作用，有可能在和 get_fpsw 的调用位置不同的时序取得 FPSW 寄存器的值。如果在进行某些运算后记述利用本函数返回值中所包含的 CV、 CO、 CZ、 CU、 CX、 CE、 FV、 FO、 FZ、 FU、 FX 或者 FS 标志的代码，就无法保证运行。

USP 的设定

void set_usp(unsigned long data)

说　明　　设定 USP。

头文件　　<machine.h>

参　数　　data　　设定值

例

```
#include <machine.h>
extern unsigned long data;
void main(void)
{
        set_usp(data);              // 将 data 的值设定到 USP。
}
```

USP 的参照

unsigned long get_usp(void)

说　明　　参照 USP。

头文件　　<machine.h>

返回值　　USP 的值

例

```
#include <machine.h>
extern unsigned long ret;
void main(void)
{
        ret=get_usp();              // 取得 USP 的值并且设定到 ret。
}
```

ISP 的设定

void set_isp(unsigned long data)

说　明　　设定 ISP。

头文件　　<machine.h>

参　数　　data　　设定值

例

```
#include <machine.h>
extern unsigned long data;
void main(void)
{
        set_isp(data);          // 将 data 的值设定到 ISP。
}
```

ISP 的参照

unsigned long get_isp(void)

说　明　　参照 ISP。

头文件　　<machine.h>

返回值　　ISP 的值

例

```
#include <machine.h>
extern unsigned long ret;
void main(void)
{
        ret=get_isp();          // 取得 ISP 的值并且设定到 ret。
}
```

INTB 的设定

void set_intb (unsigned long data)

说　明　　设定 INTB。

头文件　　<machine.h>

参　数　　data　　设定值

例

```
#include <machine.h>
extern unsigned long data;
void main(void)
{
        set_intb (data);                // 将 data 的值设定到 INTB。
}
```

INTB 的参照

unsigned long get_intb(void)

说　明　　参照 INTB。

头文件　　<machine.h>

返回值　　INTB 的值

例

```
#include <machine.h>
extern unsigned long ret;
void main(void)
{
        ret=get_intb();                 // 取得 INTB 的值并且设定到 ret。
}
```

BPSW 的设定

void set_bpsw(unsigned long data)

说　明　　设定 BPSW。

头文件　　<machine.h>

参　数　　data　　设定值

例

```
#include <machine.h>
extern unsigned long data;
void main(void)
{
        set_bpsw (data);            // 将 data 的值设定到 BPSW。
}
```

BPSW 的参照

unsigned long get_bpsw(void)

说　明　　参照 BPSW。

头文件　　<machine.h>

返回值　　BPSW 的值

例

```
#include <machine.h>
extern unsigned long ret;
void main(void)
{
        ret=get_bpsw ();            // 取得 BPSW 的值并且设定到 ret。
}
```

BPC 的设定

void set_bpc(unsigned long data)

说　明　　设定 BPC。

头文件　　<machine.h>

参　数　　data 设定值

例

```
#include <machine.h>
extern unsigned long data;
void main(void)
{
        set_bpc(data);              // 将 data 的值设定到 BPC。
}
```

BPC 的参照

unsigned long get_bpc(void)

说　明　　参照 BPC。

头文件　　<machine.h>

返回值　　BPC 的值

例

```
#include <machine.h>
extern unsigned long ret;
void main(void)
{
        ret=get_bpc();              // 取得 BPC 的值并且设定到 ret。
}
```

FINTV 的设定

void set_fintv(unsigned long data)

说　明　　设定 FINTV。

头文件　　<machine.h>

参　数　　data 设定值

例

```
#include <machine.h>
extern unsigned long data;
void main(void)
{
        set_fintv(data);            // 将 data 的值设定到 FINTV。
}
```

FINTV 的参照

unsigned long get_fintv(void)

说　明　　参照 FINTV。

头文件　　<machine.h>

返回值　　FINTV 的值

例

```
#include <machine.h>
extern unsigned long ret;
void main(void)
{
        ret=get_fintv();            // 取得 FINTV 的值并且设定到 ret。
}
```

9.2.3 段地址运算符

段地址运算符一览表如**表 9.25** 所示。

表 9.25 段运算符一览表

	段运算符名	说明
1	__sectop("< 段名 >")	参照 __sectop 指定的 < 段名 > 的起始地址。
2	__secend("< 段名 >")	参照 __secend 指定的 < 段名 > 的末尾 +1 地址。
3	__secsize("< 段名 >")	生成 __secsize 指定的 < 段名 > 的长度。

段地址运算符

__sectop, __secend, __secsize

格 式 __sectop("< 段名 >")
__secend("< 段名 >")
__secsize("< 段名 >")

说 明 参照 __sectop 指定的 < 段名 > 的起始地址。
参照 __secend 指定的 < 段名 > 的末尾 +1 地址。
生成 __secsize 指定的 < 段名 > 的长度。

返回值的型 __sectop 返回值的型为 void *。
__secend 返回值的型为 void *。
__secsize 返回值的型为 unsigned long。

例 (1) __sectop, __secend

```
#include <machine.h>
#pragma section $DSEC
static const struct {
      void *rom_s; /* 初始化数据段的 ROM 的起始地址 */
      void *rom_e; /* 初始化数据段的 ROM 的结束地址 */
      void *ram_s; /* 初始化数据段的 RAM 的起始地址 */
} DTBL[]={__sectop("D"), __secend("D"), __sectop("R")};

#pragma section $BSEC
static const struct {
     void *b_s; /* 未初始化数据段的起始地址 */
     void *b_e; /* 未初始化数据段的结束地址 */
} BTBL[]={__sectop("B"), __secend("B")};
```

```
#pragma section
#pragma stacksize si=0x100
#pragma entry INIT
void main(void);
void INIT(void)
{
        _INITSCT();
        main();
        sleep();
}
```

(2) __secsize

```
/* size of section B */
unsigned int size_of_B = _ _secsize("B");
```

9.3 C/C++ 库

9.3.1 标准 C 库

(1) 库的概要

说明 C/C++ 语言中能标准利用的 C 库函数的规格。在此概述库的结构、本节的阅读方法和术语，以下根据库的结构说明各库函数的规格。

【注】 本节出现的 double 型和 long double 型都作为指定编译程序 dbl_size=8 选项的数据型进行说明。

(a) 库的种类

所谓库是指以 C/C++ 语言的函数形式实现输入 / 输出、字符串操作等标准处理的函数。能通过包含对应各处理单位的标准 include 文件使用这些库。

在标准 include 文件中定义了对应库的声明和使用这些库所需的宏名。

库的种类和对应的标准 include 文件如表 9.26 所示。

表 9.26 库的种类和对应的标准 include 文件

	库的种类	内容	标准 include 文件
1	用于程序诊断的库	这是输出程序诊断信息的库。	<assert.h>
2	用于字符操作的库	这是操作并且检查字符的库。	<ctype.h>
3	用于数值计算的库	这是计算三角函数等数值的库。	<math.h> <mathf.h>
4	用于程序控制移动的库	这是支持函数之间控制移动的库。	<setjmp.h>
5	用于可变个数实际参数存取的库	对于有可变个数实际参数的函数，是支持存取该实际参数的库。	<stdarg.h>
6	用于输入 / 输出的库	这是进行输入 / 输出操作的库。	<stdio.h>
7	用于标准处理的库	这是进行存储区管理等 C 程序中标准处理的库。	<stdlib.h>
8	用于字符串操作的库	这是进行字符串比较和复制等的库。	<string.h>
9	复数计算库	这是计算复数的库。	<complex.h>
10	浮点环境库	这是浮点环境的库。	<fenv.h>
11	整数型的格式转换	这是进行最大宽度的整数操作和转换的库。	<inttypes.h>
12	多字节字符库、宽字符库	这是进行多字节字符操作的库。	<wchar.h> <wctype.h>

为了提高程序建立作业的效率，除以上标准 include 文件以外，还有表 9.27 中所示的只由宏名的定义构成的标准 include 文件。

表 9.27 由宏名的定义构成的标准 include 文件

	标准 include 文件	内容
1	<stddef.h>	定义各标准 include 文件中通用的宏名。
2	<limits.h>	定义有关编译程序内部处理的各种限制值。
3	<errno.h>	在库函数中发生错误时定义 errno 的设定值。
4	<float.h>	定义有关浮点型界限的各种限制值。
5	<iso646.h>	定义替代的宏名。
6	<stdbool.h>	定义有关逻辑型和逻辑值的宏。
7	<stdint.h>	声明指定宽度的整数型并且定义宏。
8	<tgmath.h>	定义统称为宏的型。

(b) 库的说明格式

按标准 include 文件将库的各函数进行分类，说明各标准 include 文件。这些分类由标准 include 文件中定义的宏名、函数声明的说明格式（参照图 9.2）和各函数的说明格式（参照图 9.3）构成。

标准 include 文件和函数的说明格式分别如图 9.2 和图 9.3 所示。

项目号＜标准include文件名＞

- 说明本include文件所具有的全部功能概要。
- 将本include文件中定义和声明的名称分为名称种类（【型】、【常数】、【变量】、【函数】）进行说明。在为宏时，在名称种类的标题（【】内）或者名称的说明位置表述为（宏）。
- 在有实现定义的规格时，或者在本include文件中声明的函数具有共同的注意事项时，进行补充说明。

图 9.2 标准 include 文件的说明格式

表示功能的概要。

表示库函数的型（返回值和参数）。

说　明	说明库函数的功能。
头文件	这是声明源的标准include文件名。
返回值	正常：这是库函数正常结束时的值。 异常：这是库函数异常结束时的值。
参　数	说明参数的含义。
例	说明调用步骤。
错误条件	表示在库函数的处理中不能从返回值判断错误发生的条件。 在发生这样的错误时，由各编译程序定义的错误种类的对应值被设定到errno*。
备　注	这是补充说明或者使用时的注意事项。
实现定义的规格	这是本编译程序的处理方法。

图 9.3　函数的说明格式

【注】 *　errno 是保存库函数执行过程中发生的错误种类的变量，详细内容请参照 “9.3.1(2) <stddef.h>”。

(c)　库函数的说明中使用的术语

(a) 流输入 / 输出

在输入或者输出数据时，如果在每次按字符调用输入 / 输出函数时驱动输入 / 输出装置或者调用 OS 功能，就会降低效率。因此，通常准备称为缓冲区的存储区，将缓冲区内的数据进行一次性的输入 / 输出。

从程序的角度来看，按字符调用输入 / 输出函数的方法比较简单。

通过库函数自动管理缓冲区，在程序中不必在意缓冲区的状态，从而能高效率地进行以字符为单位的输入 / 输出。

这种为了实现高效率的数据输入 / 输出而不必在意手段，能将输入 / 输出作为 1 个数据流进行编程的功能称为流输入 / 输出。

(b) FILE 结构体和文件指针

将流输入 / 输出所需的缓冲区和其他信息存储在一个结构体中，此结构体在标准 include 文件 <stdio.h> 中被定义为 FILE。

在流输入 / 输出中，将文件全部作为有 FILE 结构体的数据结构文件进行处理。这样的文件称为流文件，指向此文件结构体的指针称为文件指针，用于指定输入 / 输出文件。

文件指针定义为

```
FILE *fp;
```

如果用 fopen 函数等打开文件，就能得到文件指针。如果打开处理失败，就返回 NULL。必须注意：如果将 NULL 指针指定给其他流输入 / 输出函数，该函数就异常结束。在打开文件时，必须检查文件指针的值。

(c) 函数和宏

库函数的实现方法有函数和宏两种。

函数有和一般用户建立的函数相同的接口，在连接时被取进来。

宏由 #define 语句定义在与该函数相关的标准 include 文件中。

有关宏，需要注意以下几点：

- 宏由预处理器自动展开，即使用户声明了同名的函数，也无法将宏置为无效。
- 如果将有副作用的表达式 （赋值表达式、递增、递减）指定为宏的参数，就无法保证其效果。
  例：
  将计算参数绝对值的MACRO进行如下的宏定义：
  在定义#define MACRO(a) ((a)>=0?(a):-(a))时
  如果在程序内有X=MACRO(a++)，
  就展开为X=((a++)>=0?(a++):-(a++))， a递增2次，而且结果值也和最初的a的值绝对值不同。

(d) EOF

在 getc、getchar、fgetc 等函数从文件输入数据时， EOF 是在文件结束 （End Of File）时的返回值，定义在标准 include 文件 <stdio.h> 中。

(e) NULL

NULL 是什么也没有指向的指针值，定义在标准 include 文件 <stddef.h> 中。

(f) 空字符

字符 “\0”表示 C/C++ 语言中字符串的结束。
库函数中的字符串参数也必须遵循此规定。
将表示字符串结束的字符 “\0”称为空字符。

(g) 返回码

在库函数中，用返回值判断被指定的处理是否成功等结果。
此时的返回值特称为返回码。

(h) 文本文件和二进制文件

大部分系统有用于保存数据的特殊文件格式。
为了支持数据的保存，库函数中有文本文件和二进制文件共 2 种文件格式。

- 文本文件
  文本文件是用于保存一般文本的文件，由行的集合构成。在输入文本文件时，输入换行字符 （“\n”），将行分开；在输出文本文件时，输出换行字符，结束当前行的输出。文本文件是用于保存各种处理的标准文本而进行文件输入/输出的文件。对于文本文件，库函数输入或者输出的字符并不一定对应文件中的物理数据的排列。
- 二进制文件
  二进制文件是由字节数据串构成的文件。库函数输入或者输出的数据对应文件中的物理数据的排列。

(i) 标准输入 / 输出文件

输入 / 输出的库函数不进行打开文件等准备就能标准使用的文件称为标准输入 / 输出文件。标准输入 / 输出文件中有标准输入文件 （stdin）、标准输出文件 （stdout）、标准错误输出文件 （stderr）。

- 标准输入文件 （stdin）
  输入到程序的标准文件。
- 标准输出文件 （stdout）
  从程序输出的标准文件。
- 标准错误输出文件 （stderr）
  从程序输出错误信息等的标准文件。

(j) 浮点型

浮点型是用近似实数的数据表示的数据型。在 C/C++ 语言的源程序中用 10 进制表示浮点型，但是在计算机内部通常用 2 进制数表示浮点型。

在 2 进制数的情况下，浮点型的表示如下：

$2^n \times m$ （n：整数， m：2 进制小数）

在此， n 称为浮点型的指数， m 称为尾数。要用固定的数据长度表示浮点型时，通常固定 n 和 m 的位数。

以下说明有关浮点型的术语：

- 基数

  这是指示用几进制数表示浮点型的整数值，通常基数为2。

- 舍入

  在精度高于浮点型的运算过程中，将中途结果保存为浮点型时进行舍入。舍入分为上舍入，舍去和四舍五入 （在2进制小数的情况下，为0舍1入）。

- 正规化

  在以$2^n \times m$的格式表示浮点型时，同一数值可能有不同的表示。

  例：

  $2^5 \times 1.0_{(2)}$ （$_{(2)}$表示2进制数）

  $2^6 \times 0.1_{(2)}$

  都为相同的值。

  为了确保有效位数，通常使用第一个位不为0的表示，这称为正规化浮点型。将浮点型转换为这样表示的操作称为正规化。

- 保护位

  在保存浮点型运算的中途结果时，为了进行舍入，通常需要准备比实际浮点型多1位的数据。但是，这种方法在位数减少时无法得到正确的结果，因此通过再增加1位来保存运算中途结果，此位称为保护位。

(k) 文件的存取模式

表示在打开文件时对文件进行何种处理的字符串。字符串的种类有 12 种，如**表 9.28** 所示。

表 9.28 文件存取模式的种类

	存取模式	含义
1	"r"	将文本文件打开为读模式。
2	"w"	将文本文件打开为写模式。
3	"a"	将文本文件打开为追加模式。
4	"rb"	将二进制文件打开为读模式。
5	"wb"	将二进制文件打开为写模式。
6	"ab"	将二进制文件打开为追加模式。
7	"r+"	将文本文件打开为读和更新模式。
8	"w+"	将文本文件打开为写和更新模式。
9	"a+"	将文本文件打开为追加和更新模式。
10	"r+b"	将二进制文件打开为读和更新模式。
11	"w+b"	将二进制文件打开为写和更新模式。
12	"a+b"	将二进制文件打开为追加和更新模式。

(l) 实现定义

定义因编译程序而不同。

(m) 错误指示符、文件结束指示符

按流文件保持错误指示符和文件结束指示符的数据，错误指示符表示在输入或者输出文件时是否发生错误，文件结束指示符表示输入文件是否结束。

能分别通过 ferror 函数和 feof 函数参照这些数据。

在处理流文件的函数中，只用该函数的返回值有可能无法得到错误发生信息和文件结束信息。在执行这样的函数后，能通过错误指示符和文件结束指示符调查文件的状态。

(n) 位置指示符

对于能从磁盘文件等文件中的任意位置进行读写的流文件，保持在文件中表示当前读写位置的数据，该数据称为位置指示符。对于不能更改终端设备等文件中读写位置的流文件，不使用位置指示符。

(d) 使用库时的注意事项

在库中定义的宏内容因各编译程序而不同。

在使用库时，如果重新定义这些宏的内容，就无法保证运行。

库并不是在任何时候都能检测到错误。如果不使用以下说明中所示的格式调用库函数，就无法保证运行。

(2) <stddef.h>

定义标准 include 文件中通用的宏名。

以下全部为实现定义：

种类	定义名	说明
型 （宏）	ptrdiff_t	这是 2 个指针的相减结果的型。
	size_t	这是 sizeof 运算符的运算结果的型。
常数 （宏）	NULL	这是什么也没有指向的指针值。 这是用相等运算符 （==）和 0 比较的结果为真的值。
变量 （宏）	errno	如果在库函数的处理过程中发生错误，就将各库中定义的错误代码设定到此 errno。 在调用库函数前将 errno 设定为 0，在库函数的处理结束后，能通过检查设定在 errno 的代码，查出库函数处理过程中发生的错误。
函数 （宏）	offsetof	以字节为单位，求从结构体成员的结构体起始位置开始的偏移值。
型 （宏）	wchar_t	这是表示扩展字符的型。

实现定义的规格

	项目	
1	宏 NULL 的值	0 （但是，为指向 void 型的指针型）
2	适合宏 ptrdiff_t 的型	long 型
3	适合 wchar_t 的型	short 型

(3) <assert.h>

在程序中附加诊断功能。

种类	定义名	说明
函数（宏）	assert	在程序中附加诊断功能。

为了使 <assert.h> 定义的诊断功能无效，必须在包含 <assert.h> 前用 #define 语句定义 NDEBUG 的宏名（#define NDEBUG）。

【注】 如果对 assert 的宏名使用 #undef 语句，就无法保证以后的 assert 调用效果。

诊断

void assert(long expression)

说　明　　在程序中附加诊断功能。

头文件　　<assert.h>

参　数　　expression　　要评价的表达式

例

```
#include <assert.h>
int expression;
assert (expression);
```

备　注　　当 expression 为真时， assert 宏不返回值而结束处理；当 expression 为伪时，以编译程序定义的格式，将诊断信息输出到标准错误文件，然后调用 abort 函数。诊断信息中含有参数的程序文本、源文件名和源行号等信息。

实现定义的规格　在 assert(expression) 中，当 expression 为伪时输出信息。
显示因编译时的 lang 选项而不同。

(1) 在没有指定 -lang=c99 时 （C(C89)、 C++、 EC++ 语言的情况）：
ASSERTION FAILED: Δ表达式Δ FILE Δ < 文件名 >,LINE Δ < 行号 >

(2) 在指定 -lang=c99 时 （C(C99) 语言的情况）：
ASSERTION FAILED: Δ表达式Δ FILE Δ < 文件名 >,LINE Δ < 行号 > Δ
FUNCNAME Δ < 函数名 >

(4) <ctype.h>

进行字符种类的判断和转换。

种类	定义名	说明
函数	isalnum	判断是否为英文字母或者 10 进制数字。
	isalpha	判断是否为英文字母。
	iscntrl	判断是否为控制字符。
	isdigit	判断是否为 10 进制数字。
	isgraph	判断是否为非空格字符的打印字符。
	islower	判断是否为英文小写字母。
	isprint	判断是否为含空格字符的打印字符。
	ispunct	判断是否为特殊字符。
	isspace	判断是否为空格类字符。
	isupper	判断是否为英文大写字母。
	isxdigit	判断是否为 16 进制数字。
	tolower	将英文大写字母转换为英文小写字母。
	toupper	将英文小写字母转换为英文大写字母。
	lsblank	判断是空格字符还是制表符。

在上述函数中，如果输入的参数值不在能用 unsigned char 型表示的范围内并且不是 EOF 时，就无法保证该函数的运行。

字符种类一览表如表 9.29 所示。

表 9.29 字符种类

	字符种类	内容
1	英文大写字母	为以下 26 个字母中的任意字母： 'A'、 'B'、 'C'、 'D'、 'E'、 'F'、 'G'、 'H'、 'I'、 'J'、 'K'、 'L'、 'M'、 'N'、 'O'、 'P'、 'Q'、 'R'、 'S'、 'T'、 'U'、 'V'、 'W'、 'X'、 'Y'、 'Z'
2	英文小写字母	为以下 26 个字母中的任意字母： 'a'、 'b'、 'c'、 'd'、 'e'、 'f'、 'g'、 'h'、 'i'、 'j'、 'k'、 'l'、 'm'、 'n'、 'o'、 'p'、 'q'、 'r'、 's'、 't'、 'u'、 'v'、 'w'、 'x'、 'y'、 'z'
3	英文字母	为英文大小写字母中的任意字母。
4	10 进制数字	为以下 10 个字符中的任意字符： '0'、 '1'、 '2'、 '3'、 '4'、 '5'、 '6'、 '7'、 '8'、 '9'
5	打印字符	显示器上显示的含空格 （' '）的字符。 对应 ASCII 码的 0x20 ～ 0x7E。
6	控制字符	为打印字符以外的字符。
7	空格类字符	为以下 6 个字符中的任意字符： 空格 （' '）、换页 （'\f'）、换行 （'\n'）、回车 （'\r'）、水平制表符 （'\t'）、垂直制表符 （'\v'）
8	16 进制数字	为以下 22 个字符中的任意字符： '0'、 '1'、 '2'、 '3'、 '4'、 '5'、 '6'、 '7'、 '8'、 '9'、 'A'、 'B'、 'C'、 'D'、 'E'、 'F'、 'a'、 'b'、 'c'、 'd'、 'e'、 'f'
9	特殊字符	为不含空格 （' '）、英文字母和 10 进制数字的任意打印字符。
10	空白字符	为以下 2 个字符中的任意字符： 空格 （' '）、水平制表符 （'\t'）

实现定义的规格

	项目	编译程序的规格
1	通过 isalnum 函数、 isalpha 函数、 iscntrl 函数、 islower 函数、 isprint 函数和 isupper 函数判断的字符集	这是能用 unsigned char 型表示的字符集。判断结果为真的字符如表 9.30 所示。

表 9.30 为真的字符集

	函数名	为真的字符
1	isalnum	'0' ～ '9'、 'A' ～ 'Z'、 'a' ～ 'z'
2	isalpha	'A' ～ 'Z'、 'a' ～ 'z'
3	iscntrl	'\x00' ～ '\x1f'、 '\x7f'
4	islower	'a' ～ 'z'
5	isprint	'\x20' ～ '\x7E'
6	isupper	'A' ～ 'Z'

英文字母和10 进制数字的判断

long isalnum(long c)

说　明　　判断字符是否为英文字母或者 10 进制数字。

头文件　　<ctype.h>

返回值　　当字符 c 为英文字母或者 10 进制数字时　　：非 0 值
当字符 c 不为英文字母或者 10 进制数字时　：0

参　数　　c　　要判断的字符

例

```
#include <ctype.h>
int c, ret;
   ret=isalnum(c);
```

英文字母的判断

long isalpha(long c)

说　明　　判断字符是否为英文字母。

头文件　　<ctype.h>

返回值　　当字符 c 为英文字母时　　：非 0 值
当字符 c 不为英文字母时　：0

参　数　　c　　要判断的字符

例

```
#include <ctype.h>
int c, ret;
   ret=isalpha(c);
```

控制字符的判断

long iscntrl(long c)

说　明　　判断字符是否为控制字符。

头文件　　<ctype.h>

返回值　　当字符 c 为控制字符时　　：非 0 值
　　当字符 c 不为控制字符时　：0

参　数　　c　　要判断的字符

例　　
```
#include <ctype.h>
int c, ret;
   ret=iscntrl(c);
```

10 进制数字的判断

long isdigit(long c)

说　明　　判断字符是否为 10 进制数字。

头文件　　<ctype.h>

返回值　　当字符 c 为 10 进制数字时　　：非 0 值
　　当字符 c 不为 10 进制数字时　：0

参　数　　c　　要判断的字符

例　　
```
#include <ctype.h>
int c, ret;
   ret=isdigit(c);
```

非空格字符的打印字符的判断

long isgraph(long c)

说　明　　判断字符是否为非空格字符（' '）的任意打印字符。

头文件　　<ctype.h>

返回值　　当字符 c 为非空格字符的打印字符时　　：非 0 值
当字符 c 不为非空格字符的打印字符时　：0

参　数　　c　　要判断的字符

例

```
#include <ctype.h>
int c, ret;
   ret=isgraph(c);
```

英文小写字母的判断

long islower(long c)

说　明　　判断字符是否为英文小写字母。

头文件　　<ctype.h>

返回值　　当字符 c 为英文小写字母时　　：非 0 值
当字符 c 不为英文小写字母时　：0

参　数　　c　　要判断的字符

例

```
#include <ctype.h>
int c, ret;
   ret=islower(c);
```

打印字符的判断

long isprint(long c)

说　明　　判断字符是否为含空格字符（' '）的打印字符。

头文件　　<ctype.h>

返回值　　当字符 c 为含空格字符的打印字符时　　：非 0 值
当字符 c 不为含空格字符的打印字符时　：0

参　数　　c　　要判断的字符

例

```
#include <ctype.h>
int c, ret;
   ret=isprint (c);
```

特殊字符的判断

long ispunct(long c)

说　明　　判断字符是否为特殊字符。

头文件　　<ctype.h>

返回值　　当字符 c 为特殊字符时　　：非 0 值
当字符 c 不为特殊字符时　：0

参　数　　c　　要判断的字符

例

```
#include <ctype.h>
int c, ret;
   ret=ispunct(c);
```

空格类字符的判断

long isspace(long c)

说　明　　判断字符是否为空格类字符。

头文件　　<ctype.h>

返回值　　当字符 c 为空格类字符时　　：非 0 值
当字符 c 不为空格类字符时　：0

参　数　　c　　要判断的字符

例

```
#include <ctype.h>
int c, ret;
   ret=isspace(c);
```

英文大写字母的判断

long isupper(long c)

说　明　　判断字符是否为英文大写字母。

头文件　　<ctype.h>

返回值　　当字符 c 为英文大写字母时　　：非 0 值
当字符 c 不为英文大写字母时　：0

参　数　　c　　要判断的字符

例

```
#include <ctype.h>
int c, ret;
   ret=isupper(c);
```

16 进制数字的判断

long isxdigit(long c)

说　明　　判断字符是否为 16 进制数字。

头文件　　<ctype.h>

返回值　　当字符 c 为 16 进制数字时　　：非 0 值
当字符 c 不为 16 进制数字时　：0

参　数　　c　　要判断的字符

例

```
#include <ctype.h>
int c, ret;
   ret=isxdigit(c);
```

英文小写字母的转换

long tolower(long c)

说　明　　将英文大写字母转换为对应的英文小写字母。

头文件　　<ctype.h>

返回值　　当字符 c 为英文大写字母时　　：字符 c 对应的英文小写字母
当字符 c 不为英文大写字母时　：字符 c

参　数　　c　　要转换的字符

例

```
#include <ctype.h>
int c, ret;
   ret=tolower(c);
```

英文大写字母的转换

long toupper(long c)

说　明　　将英文小写字母转换为对应的英文大写字母。

头文件　　<ctype.h>

返回值　　当字符 c 为英文小写字母时　：字符 c 对应的英文大写字母
当字符 c 不为英文小写字母时　：字符 c

参　数　　c　　要转换的字符

例

```
#include <ctype.h>
int c, ret;
   ret=toupper(c);
```

空白的判断

long isblank(long c)

说　明　　判断字符是否为空格字符或者制表符。

头文件　　<ctype.h>

返回值　　当字符 c 为空格字符或者制表符时　：非 0 值
当字符 c 既不为空格字符也不为制表符时　：0

参　数　　c　　要判断的字符

例

```
#include <ctype.h>
int c, ret;
   ret=isblank(c);
```

(5) <float.h>

定义有关浮点型内部表示的各种极限值。

以下全部为实现定义：

种类	定义名	定义值	说明
常数 （宏）	FLT_RADIX	2	这是指数表示中的基数。
	FLT_ROUNDS	1	表示是否舍入加法运算的结果。此宏定义的含义如下： • 当舍入运算结果时：正值 • 当舍去运算结果时：0 • 当不特别规定时：–1 舍入和舍去的方法为实现定义。
	FLT_GUARD	1	表示在乘法运算结果中是否使用保护位。此宏定义的含义如下： • 当使用保护位时：1 • 当不使用保护位时：0
	FLT_NORMALIZE	1	表示是否对浮点值进行正规化处理。此宏定义的含义如下： • 当进行正规化处理时：1 • 当不进行正规化处理时：0
	FLT_MAX	3.4028235677973364e+38F	作为浮点值，为 float 型能表示的最大值。
	DBL_MAX	1.7976931348623158e+308	作为浮点值，为 double 型能表示的最大值。
	LDBL_MAX	1.7976931348623158e+308	作为浮点值，为 long double 型能表示的最大值。
	FLT_MAX_EXP	127	作为浮点值，为 float 型能表示基数的乘方的最大值。
	DBL_MAX_EXP	1023	作为浮点值，为 double 型能表示基数的乘方的最大值。
	LDBL_MAX_EXP	1023	作为浮点值，为 long double 型能表示基数的乘方的最大值。
	FLT_MAX_10_EXP	38	作为浮点值，为 float 型能表示 10 的乘方的最大值。
	DBL_MAX_10_EXP	308	作为浮点值，为 double 型能表示 10 的乘方的最大值。
	LDBL_MAX_10_EXP	308	作为浮点值，为 long double 型能表示 10 的乘方的最大值。
	FLT_MIN	1.175494351e–38F	作为浮点值，为 float 型能表示正值的最小值。
	DBL_MIN	2.2250738585072014e–308	作为浮点值，为 double 型能表示正值的最小值。
	LDBL_MIN	2.2250738585072014e–308	作为浮点值，为 long double 型能表示正值的最小值。
	FLT_MIN_EXP	–149	作为正值，为 float 型能表示浮点值基数的乘方的最小值。
	DBL_MIN_EXP	–1074	作为正值，为 double 型能表示浮点值基数的乘方的最小值。
	LDBL_MIN_EXP	–1074	作为正值，为 long double 型能表示浮点值基数的乘方的最小值。

种类	定义名	定义值	说明
常数 （宏）	FLT_MIN_10_EXP	–44	作为正值，为 float 型能表示浮点值的 10 的乘方的最小值。
	DBL_MIN_10_EXP	–323	作为正值，为 double 型能表示浮点值的 10 的乘方的最小值。
	LDBL_MIN_10_EXP	–323	作为正值，为 long double 型能表示浮点值的 10 的乘方的最小值。
	FLT_DIG	6	这是 float 型浮点值的 10 进制精度的最大位数。
	DBL_DIG	15	这是 double 型浮点值的 10 进制精度的最大位数。
	LDBL_DIG	15	这是 long double 型浮点值的 10 进制精度的最大位数。
	FLT_MANT_DIG	24	这是按基数表示 float 型浮点值时的尾数的最大位数。
	DBL_MANT_DIG	53	这是按基数表示 double 型浮点值时的尾数的最大位数。
	LDBL_MANT_DIG	53	这是按基数表示 long double 型浮点值时的尾数的最大位数。
	FLT_EXP_DIG	8	这是按基数表示 float 型浮点值时的指数的最大位数。
	DBL_EXP_DIG	11	这是按基数表示 double 型浮点值时的指数的最大位数。
	LDBL_EXP_DIG	11	这是按基数表示 long double 型浮点值时的指数的最大位数。
	FLT_POS_EPS	5.9604648328104311e–8F	表示 float 型的 1.0+x≠1.0 的最小浮点值 x。
	DBL_POS_EPS	1.1102230246251567e–16	表示 double 型的 1.0+x≠1.0 的最小浮点值 x。
	LDBL_POS_EPS	1.1102230246251567e–16	表示 long double 型的 1.0+x≠1.0 的最小浮点值 x。
	FLT_NEG_EPS	2.9802324164052156e–8F	表示 float 型的 1.0–x≠1.0 的最小浮点值 x。
	DBL_NEG_EPS	5.5511151231257834e–17	表示 double 型的 1.0–x≠1.0 的最小浮点值 x。
	LDBL_NEG_EPS	5.5511151231257834e–17	表示 long double 型的 1.0–x≠1.0 的最小浮点值 x。
	FLT_POS_EPS_EXP	–23	表示 float 型的 1.0+(基数)^n≠1.0 的最小整数 n。
	DBL_POS_EPS_EXP	–52	表示 double 型的 1.0+(基数)^n≠1.0 的最小整数 n。
	LDBL_POS_EPS_EXP	–52	表示 long double 型的 1.0+(基数)^n≠1.0 的最小整数 n。
	FLT_NEG_EPS_EXP	–24	表示 float 型的 1.0–(基数)^n≠1.0 的最小整数 n。
	DBL_NEG_EPS_EXP	–53	表示 double 型的 1.0–(基数)^n≠1.0 的最小整数 n。
	LDBL_NEG_EPS_EXP	–53	表示 long double 型的 1.0–(基数)^n≠1.0 的最小整数 n。
	DECIMAL_DIG	10	这是浮点型数值的 10 进制精度的最大位数。
	FLT_EPSILON	1E–5	这是 float 型能表示的大于 1 的最小值和 1 的差。
	DBL_EPSILON	1E–9	这是 double 型能表示的大于 1 的最小值和 1 的差。
	LDBL_EPSILON	1E–9	这是 long double 型能表示的大于 1 的最小值和 1 的差。

(6) <limits.h>

定义有关整数型数据内部表示的各种极限值。

以下全部为实现定义：

种类	定义名	定义值	说明
常数 （宏）	CHAR_BIT	8	表示 char 型由几位构成。
	CHAR_MAX	127	作为值，为 char 型变量具有的最大值。
	CHAR_MIN	–128	作为值，为 char 型变量具有的最小值。
	SCHAR_MAX	127	作为值，为 signed char 型变量具有的最大值。
	SCHAR_MIN	–128	作为值，为 signed char 型变量具有的最小值。
	UCHAR_MAX	255U	作为值，为 unsigned char 型变量具有的最大值。
	SHRT_MAX	32767	作为值，为 short 型变量具有的的最大值。
	SHRT_MIN	–32768	作为值，为 short 型变量具有的最小值。
	USHRT_MAX	65535U	作为值，为 unsigned short 型变量具有的最大值。
	INT_MAX	2147483647	作为值，为 int 型变量具有的最大值。
		32767*	
	INT_MIN	–2147483647–1	作为值，为 int 型变量具有的最小值。
		–32768*	
	UINT_MAX	4294967295U	作为值，为 unsigned int 型变量具有的最大值。
		65535U*	
	LONG_MAX	2147483647L	作为值，为 long 型变量具有的最大值。
	LONG_MIN	–2147483647L–1L	作为值，为 long 型变量具有的最小值。
	ULONG_MAX	4294967295U	作为值，为 unsigned long 型变量具有的最大值。
	LLONG_MAX	9223372036854775807LL	作为值，为 long long 型变量具有的最大值。
	LLONG_MIN	–9223372036854775807LL–1LL	作为值，为 long long 型变量具有的最小值。
	ULLONG_MAX	18446744073709551615ULL	作为值，为 unsigned long long 型变量具有的最大值。

【注】 * 作为值，为指定 int_to_short 选项时的变量具有的值。

(7) <errno.h>

定义在库函数中发生错误时给 errno 的设定值。

以下全部为实现定义：

种类	定义名	说明
变量 （宏）	errno	这是 int 型变量。在库函数中发生错误时，设定错误号。
常数 （宏）	ERANGE	请参照 “11.3 C 标准库函数的错误信息”。
	EDOM	
	ESTRN	
	PTRERR	
	ECBASE	
	ETLN	
	EEXP	
	EEXPN	
	EFLOATO	
	EFLOATU	
	EDBLO	
	EDBLU	
	ELDBLO	
	ELDBLU	
	NOTOPN	
	EBADF	
	ECSPEC	
	EFIXEDO	
	EFIXEDU	
	EACCUMO	
	EACCUMU	
	ELFIXEDO	
	ELFIXEDU	
	ELACCUMO	
	ELACCUMU	
	EILSEQ	

(8) <math.h>

进行各种数值的计算。

以下常数 （宏）全部为实现定义：

种类	定义名	说明
常数 （宏）	EDOM	这是在输入给函数的参数值超出函数内定义值的范围时设定给 errno 的值。
	ERANGE	这是在函数的计算结果不能表示为 double 型的值或者发生上溢 / 下溢时设定给 errno 的值。
	HUGE_VAL HUGE_VALF HUGE_VALL	这是在函数的计算结果发生下溢时，作为函数的返回值返回的值。
	INFINITY	展开为表示正或者无符号的无穷大的 float 型常数表达式。
	NAN	这是在支持 float 型 qNaN 时被定义的值。
	FP_INFINITE FP_NAN FP_NORMAL FP_SUBNORMAL FP_ZERO	表示浮点数值的互斥种类。
	FP_FAST_FMA FP_FAST_FMAF FP_FAST_FMAFL	这是在以同等以上的速度对有 double 型操作数的 Fma 函数进行 1 次乘法运算和加法运算时被定义的值。
	FP_ILOGB0 FP_ILOGBNAN	在分别为 0 或者非数值时展开为以 ilogb 返回的值的整数常数表达式。
	MATH_ERRNO MATH_ERREXCEPT	分别展开为整数常数 1 和整数常数 2。
	math_errhandling	Int 型的值展开为以 MATH_ERRNO、 MATH_ERREXCEPT 位为单位的逻辑 “或”的表达式。
型	float_t double_t	分别是具有和 float 型和 double 型相同宽度的浮点型。
函数 （宏）	fpclassify	将实际参数的值分为非数值、无穷大、正规化数、非正规化数、 0。
	isfinite	判断实际参数是否为有限值。
	isinf	判断实际参数是否为无穷大。
	isnan	判断实际参数是否为非数值。
	isnormal	判断实际参数是否为正规化数。
	signbit	判断实际参数的符号是否为负。
	isgreater	判断第 1 个参数是否大于第 2 个参数。
	isgreaterequal	判断第 1 个参数是否大于等于第 2 个参数。
	isless	判断第 1 个参数是否小于第 2 个参数。
	Islessequal	判断第 1 个参数是否小于等于第 2 个参数。
	Islessgreater	判断第 1 个参数是否大于或者小于第 2 个参数。
	Isunordered	判断是否已排序。

种类	定义名	说明
函数	acos acosf acosl	计算浮点值的反余弦。
	asin asinf asinl	计算浮点值的反正弦。
	atan atanf atanl	浮点值的反正切。
	atan2 atan2f atan2l	计算浮点值除浮点值后的结果值的反正切。
	cos cosf cosl	计算浮点值的弧度值的余弦。
	sin sinf sinl	计算浮点值的弧度值的正弦。
	tan tanf tanl	计算浮点值的弧度值的正切。
	cosh coshf coshl	计算浮点值的双曲余弦。
	sinh sinhf sinhl	计算浮点值的双曲正弦。
	tanh tanhf tanhl	计算浮点值的双曲正切。
	exp expf expl	计算浮点值的指数函数。
	frexp frexpf frexpl	将浮点值分为 [0.5,1.0] 的值和 2 的乘方的积。
	ldexp ldexpf ldexpl	计算浮点值和 2 的乘方的积。
	log logf logl	计算浮点值的自然对数。
	log10 log10f log10l	计算以 10 为底的浮点值的对数。

种类	定义名	说明
函数	modf modff modfl	将浮点值分为整数和小数。
	pow powf powl	计算浮点值的乘方。
	sqrt sqrtf sqrtl	计算浮点值的正平方根。
	ceil ceilf ceill	求上舍入浮点值小数部分后的整数值。
	fabs fabsf fabsl	计算浮点值的绝对值。
	floor floorf floorl	求舍去浮点值小数部分后的整数值。
	fmod fmodf fmodl	计算浮点值除浮点值相后的结果的余数。
	acosh acoshf acoshl	计算浮点值的双曲反余弦。
	asinh asinhf asinhl	计算浮点值的双曲反正弦。
	atanh atanhf atanhl	计算浮点值的双曲反正切。
	exp2 exp2f exp2l	计算浮点值的 2 的 x 乘方。
	expm1 expm1f expm1l	计算自然对数的 x 乘方减去 1 后的值。
	ilogb ilogbf ilogbl	作为带符号的 int 值，抽出 x 的指数。
	log1p log1pf log1pl	计算实际参数加上 1 后的值的自然对数。
	log2 log2f log2l	计算以 2 为底的对数。

种类	定义名	说明
函数	logb logbf logbl	作为带符号的整数值，抽出 x 的指数。
	scalbn scalbnf scalbnl scalbln scalblnf scalblnl	计算 X×FLT_RADIXn。
	cbrt cbrtf cbrtl	计算浮点值的立方根。
	hypot hypotf hypotl	求各浮点值的参数的平方，并计算它们的和。
	erf erff erfl	计算误差函数。
	erfc erfcf erfcl	计算补余误差函数。
	lgamma lgammaf lgammal	计算 γ 函数的绝对值的自然对数。
	tgamma tgammaf tgammal	计算 γ 函数。
	nearbyint nearbyintf nearbyintl	根据浮点值的舍入方向，向浮点格式的整数值舍入。
	rint rintf rintl	对于 nearbyint，有可能生成浮点异常。
	lrint lrintf lrintl llrint llrintf llrintl	根据舍入方向，向最接近的整数值舍入。
	round roundf roundl	向浮点格式的最接近的整数值舍入。

种类	定义名	说明
函数	lround lroundf lroundl llround llroundf llroundl	向最接近的整数值舍入。
	trunc truncf truncl	向浮点格式的最接近的整数值舍入。
	remainder remainderf remainderl	计算 IEEE60559 的余数 x REM y。
	remquo remquof remquol	计算和 x/y 同符号的并且和被 2^n 整除的 x/y 商的绝对值。
	copysign copysignf copysignl	生成绝对值和符号相同的值。
	nan nanf nanl	nan("n 字符串 ") 和 strtod("NAN(n 字符串)", (char**)NULL) 等效。
	nextafter nextafterf nextafterl	在转换为函数的型后，求实数轴上的下一个能表示的值。
	nexttoward nexttowardf nexttowardl	当第 2 个参数型为 long double 并且参数和参数相等时，在将第 2 个参数转换为该函数的型后返回，其他和 nextafter 函数群相同。
	fdim fdimf fdiml	计算正的差分。
	fmax fmaxf fmaxl	求较大的值。
	fmin fminf fminl	求较小的值。
	fma fmaf fmal	作为 3 元运算，计算 (x×y)+z。

以下说明发生错误时的运行。

1. 定义域错误
   当输入给函数的参数值超出函数内定义值的范围时，发生定义域错误。此时，将EDOM的值设定到errno。函数的返回值为实现定义。
2. 范围错误
   当函数中的计算结果不能表示为double型的值时，发生范围错误。此时，将ERANGE的值设定到errno。当计算结果发生上溢时，将和正确计算时相同符号的HUGE_VAL、 HUGE_VALF或者HUGE_VALL的值作为返回值返回；当计算结果发生下溢时，将0作为返回值返回。

【注】 1. 在可能因 <math.h> 函数的调用而发生定义域错误时，直接使用结果值很危险，必须在检查 errno 后再使用。
例：

```
                .
                .
                .
1           x=asin(a);
2           if (errno==EDOM)
3               printf("error\n");
4           else
5               printf("result is : %lf\n",x);
                .
                .
                .
```

在第 1 行，使用 asin 函数求反正弦值。此时，如果实际参数 a 的值超出 asin 函数的定义域 [–1.0, 1.0] 的范围，就将 EDOM 的值设定到 errno。在第 2 行，判断是否发生定义域错误。如果发生定义域错误，就在第 3 行输出 error，否则就在第 5 行输出反正弦值。

2. 是否发生范围错误取决于编译程序规定的浮点型内部表示格式。例如，在采用能将无穷大作为值表示的内部表示格式时，不发生范围错误而能执行 <math.h> 的库函数。

实现定义的规格

	项目	编译程序的规格
1	输入给数学函数的实际参数超出范围时的数学函数的返回值	返回非数值，非数值的格式请参照 “9.1.3 浮点型的规格”。
2	在数学函数中发生下溢错误时，是否将宏 “ERANGE”的值设定到 “errno”。	不设定。
3	当 fmod 函数中的第 2 个实际参数的值为 0 时，是否发生范围错误。	发生范围错误。

反余弦

double acos(double d)
float acosf(float d)
long double acosl(long double d)

说　明　　计算浮点值的反余弦。

头文件　　<math.h>

返回值　　正常：d 的反余弦值
异常：在发生定义域错误时，返回非数值。

参　数　　d　　要计算反余弦的浮点值

例
```
#include <math.h>
double d, ret;
   ret=acos(d);
```

错误条件　　当 d 的值超出 [-1.0,1.0] 的范围时，发生定义域错误。

备　注　　acos 函数返回值的范围为 [0,π]。

反正弦

double asin(double d)
float asinf(float d)
long double asinl(long double)

说　明　　计算浮点值的反正弦。

头文件　　<math.h>

返回值　　正常：d 的反正弦值
异常：在发生定义域错误时，返回非数值。

参　数　　d　　要计算反正弦的浮点值

例
```
#include <math.h>
double d, ret;
   ret=asin(d);
```

错误条件　　当 d 的值超出 [-1.0,1.0] 的范围时，发生定义域错误。

备　注　　asin 函数返回值的范围为 [-π/2,π/2]。

反正切

double atan(double d)
float atanf(float d)
long double atanl(long double d)

说　明　　计算浮点值的反正切。

头文件　　<math.h>

返回值　　d 的反正切值

参　数　　d　　要计算反正切的浮点值

例

```
#include <math.h>
double d, ret;
   ret=atan(d);
```

备　注　　atan 函数返回值的范围为 (-π/2,π/2)。

除法运算后的反正切

double atan2(double y, double x)
float atan2f(float y, float x)
long double atan2l(long double y, long double x)

说　明　　计算浮点值除浮点值后的结果值的反正切。

头文件　　<math.h>

返回值　　正常：y 除 x 后的反正切值
异常：在发生定义域错误时，返回非数值。

参　数

x	除数
y	被除数

例

```
#include <math.h>
double x, y, ret;
   ret=atan2(y,x);
```

错误条件　　当 x 和 y 的值都是 0.0 时，发生定义域错误。

备　注　　atan2 函数返回值的范围为 (-π,π)。atan2 函数表示的含义如图 9.4 所示，atan2 函数的结果是求通过点 (x, y) 和原点的直线与 x 轴的角度。当 y=0.0 并且 x 为负时，结果为 π；当 x=0.0 时，根据 y 值的正负，结果为 ±π/2。

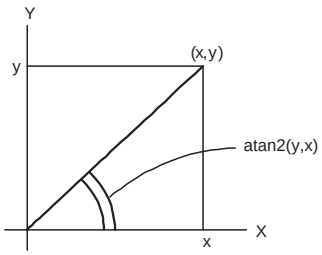


图 9.4　atan2 函数的含义

余弦

double cos(double d)
float cosf(float d)
long double cosl(long double d)

说　明　　计算浮点值的弧度值的余弦。

头文件　　<math.h>

返回值　　d 的余弦值

参　数　　d　　要计算余弦的弧度值

例

```
#include <math.h>
double d, ret;
   ret=cos(d);
```

正弦

double sin(double d)
float sinf(float d)
long double sinl(long double d)

说　明　　计算浮点值的弧度值的正弦。

头文件　　<math.h>

返回值　　d 的正弦值

参　数　　d　　要计算正弦的弧度值

例

```
#include <math.h>
double d, ret;
   ret=sin(d);
```

正切

double tan(double d)
float tanf(float d)
long double tanl(long double d)

说　明　　计算浮点值的弧度值的正切。

头文件　　<math.h>

返回值　　d 的正切值

参　数　　d　　要计算正切的弧度值

例

```
#include <math.h>
double d, ret;
   ret=tan(d);
```

双曲余弦

double cosh(double d)
float coshf(float d)
long double coshl(long double d)

说　明　　计算浮点值的双曲余弦。

头文件　　<math.h>

返回值　　d 的双曲余弦值

参　数　　d　　要计算双曲余弦的浮点值

例

```
#include <math.h>
double d, ret;
   ret=cosh(d);
```

双曲正弦

double sinh(double d)
float sinhf(float d)
long double sinhl(long double d)

说　明　　计算浮点值的双曲正弦。

头文件　　<math.h>

返回值　　d 的双曲正弦值

参　数　　d　　要计算双曲正弦的浮点值

例

```
#include <math.h>
double d, ret;
   ret=sinh(d);
```

双曲正切

double tanh(double d)
float tanhf(float d)
long double tanhl(long double d)

说　明　　计算浮点值的双曲正切。

头文件　　<math.h>

返回值　　d 的双曲正切值

参　数　　d　　要计算双曲正切的浮点值

例

```
#include <math.h>
double d, ret;
   ret=tanh(d);
```

指数函数

double exp(double d)
float expf(float d)
long double expl(long double d)

说　明　　计算浮点值的指数函数。

头文件　　<math.h>

返回值　　d 的指数函数值

参　数　　d　　要计算指数函数的浮点值

例

```
#include <math.h>
double d, ret;
   ret=exp(d);
```

将浮点值分为尾数和指数

double frexp(double value, long *exp)
float frexpf(float value, long *exp)
long double frexpl(long double value, long *exp)

说　明　　将浮点值分为 [0.5,1.0) 的值和 2 的乘方的积。

头文件　　<math.h>

返回值　　当 value 为 0.0 时　　：0.0
当 value 不为 0.0 时　：ret*$2^{exp\ 的指向区域的值}$ =value 定义的 ret 值

参　数　　value　　要分为 [0.5,1.0) 的值和 2 的乘方的积的浮点值
exp　　指向保存 2 的乘方值的存储区的指针

例

```
#include <math.h>
double ret, value;
long *exp;
   ret=frexp(value,exp);
```

备　注　　frexp 函数将 value 分为 [0.5,1.0) 的值和 2 的乘方的积。将分解后的结果的 2 的乘方值设定到 exp 指向的区域。
返回值 ret 的范围为 [0.5,1.0) 或者 0.0。
如果 value 为 0.0， exp 指向的 int 型存储区的内容和 ret 的值就为 0.0。

将尾数和指数转换为浮点值

double ldexp(double e, long f)
float ldexpf(float e, long f)
long double ldexpl(long double e, long f)

说　明　　计算浮点值和 2 的乘方的积。

头文件　　<math.h>

返回值　　$e*2^f$ 的运算结果的值

参　数　　e　　要计算 2 的乘方值的浮点值
　　f　　2 的乘方值

例
```
#include <math.h>
double ret, e;
int f;
   ret=ldexp(e, f);
```

自然对数

double log(double d)
float logf(float d)
long double logl(long double d)

说　明　　计算浮点值的自然对数。

头文件　　<math.h>

返回值　　正常：d 的自然对数的值
　　异常：在发生定义域错误时，返回非数值。

参　数　　d　　要计算自然对数的浮点值

例
```
#include <math.h>
double d, ret;
   ret=log(d);
```

错误条件　　当 d 的值为负时，发生定义域错误。
　　当 d 的值为 0.0 时，发生范围错误。

常用对数

double log10(double d)
float log10f(float d)
long double log10l(long double d)

说　明　　计算以 10 为底的浮点值的对数。

头文件　　<math.h>

返回值　　正常：以 10 为底的 d 的对数值
异常：在发生定义域错误时，返回非数值。

参　数　　d　　要计算以 10 为底的对数的浮点值

例
```
#include <math.h>
double d, ret;
   ret=log10(d);
```

错误条件　　当 d 的值为负值时，发生定义域错误。
当 d 的值为 0.0 时，发生范围错误。

将浮点值分为整数和小数

double modf(double a, double *b)
float modff(float a, float *b)
long double modfl(long double a, long double *b)

说　明　　将浮点值分为整数和小数。

头文件　　<math.h>

返回值　　a 的小数部分

参　数　　a　　要分为整数和小数的浮点值
b　　指向保存整数的存储区的指针

例
```
#include <math.h>
double a, *b, ret;
   ret=modf(a,b);
```

乘方

double pow(double x, double y)
float powf(float x, float y)
long double powl(long double x, long double y)

说　明　　计算浮点值的乘方。

头文件　　<math.h>

返回值　　正常：x 的 y 乘方的值
异常：在发生定义域错误时，返回非数值。

参　数　　x　　被乘方的值
y　　乘方的值

例

```
#include <math.h>
double x, y, ret;
   ret=pow(x,y);
```

错误条件　　当 x 的值为 0.0 并且 y 的值小于等于 0.0 或者 x 的值为负并且 y 的值不是整数值时，发生定义域错误。

平方根

double sqrt(double d)
float sqrtf(float d)
long double sqrtl(long double d)

说　明　　计算浮点值的正平方根。

头文件　　<math.h>

返回值　　正常：d 的正平方根的值
异常：在发生定义域错误时，返回非数值。

参　数　　d　　要计算正平方根的浮点值

例

```
#include <math.h>
double d, ret;
   ret=sqrt(d);
```

错误条件　　当 d 的值为负值时，发生定义域错误。

上舍入

double ceil(double d)
float ceilf(float d)
long double ceill(long double d)

说　明　　计算上舍入浮点值的小数部分后的整数值。

头文件　　<math.h>

返回值　　上舍入 d 的小数部分后的整数值

参　数　　d　　要上舍入小数部分的浮点值

例

```
#include <math.h>
double d, ret;
   ret=ceil(d);
```

备　注　　ceil 函数将大于等于 d 的最小整数值作为 double 型的值返回。因此，当 d 的值为负值时，返回舍去小数部分后的值。

绝对值

double fabs(double d)
float fabsf(float d)
long double fabsl(long double d)

说　明　　计算浮点值的绝对值。

头文件　　<math.h>

返回值　　d 的绝对值

参　数　　d　　要计算绝对值的浮点值

例

```
#include <math.h>
double d, ret;
   ret=fabs(d);
```

舍去

double floor(double d)
float floorf(float d)
long double floorl(long double d)

说　明　　计算舍去浮点值的小数部分后的整数值。

头文件　　<math.h>

返回值　　舍去 d 的小数部分后的整数值

参　数　　d　　要舍去小数部分的浮点值

例

```
#include <math.h>
double d, ret;
   ret=floor(d);
```

备　注　　floor 函数将不超过 d 的范围的最大整数值作为 double 型的值返回。
因此，当 d 的值为负值时，返回上舍入小数部分后的值。

余数

double fmod(double x, double y)
float fmodf(float x, float y)
long double fmodl(long double x, long double y)

说　明　　计算浮点值除浮点值后的结果的余数。

头文件　　<math.h>

返回值　　当 y 的值为 0.0 时　：x
当 y 的值不为 0.0 时　：x 除 y 的结果的余数

参　数　　x　　被除数
y　　除数

例

```
#include <math.h>
double x, y, ret;
   ret=fmod(x,y);
```

备　注　　在 fmod 函数中，参数 x、参数 y 和返回值 ret 有以下的关系：
x=y*i+ret （i 为整数值）
返回值 ret 的符号和 x 的符号相同。
如果无法表示 x/y 的商，就无法保证结果值。

双曲反余弦

double acosh(double d)
float acoshf(float d)
long double acoshl(long double d)

说　明　　计算双曲反余弦。

头文件　　<math.h>

返回值　　正常：d 的双曲反余弦值
异常：在发生定义域错误时，返回非数值。

参　数　　d　　要计算双曲反余弦的浮点值

例
```
#include <math.h>
double d, ret;
   ret=acosh(d);
```

错误条件　　当 d 的值小于 1.0 时，发生定义域错误。

备　注　　acosh 函数返回值的范围为 [0,+∞]。

双曲反正弦

double asinh(double d)
float asinhf(float d)
long double asinhl(long double d)

说　明　　计算双曲反正弦。

头文件　　<math.h>

返回值　　d 的双曲反正弦值

参　数　　d　　要计算双曲反正弦的浮点值

例
```
#include <math.h>
double d, ret;
   ret=asinh(d);
```

双曲反正切

double atanh(double d)
float atanhf(float d)
long double atanhl(long double d)

说　明　　计算双曲反正切。

头文件　　<math.h>

返回值　　正常：d 的双曲反正切值
异常：当发生定义域错误时，根据函数，返回值为 HUGE_VAL、 HUGE_VALF 或者 HUGE_VALL。
当发生范围错误时，返回值为非数值。

参　数　　d　　要计算双曲反正切的浮点值

例

```
#include <math.h>
double d, ret;
   ret=atanh(d);
```

错误条件　　当 d 的值不在 [-1,+1] 范围内时，发生定义域错误。
当 d 的值为 -1 或者 1 时，可能发生范围错误。

指数

double exp2(double d)
float exp2f(float d)
long double exp2l(long double d)

说　明　　计算 2 的 d 乘方。

头文件　　<math.h>

返回值　　正常：2 的指数函数值
异常：当发生范围错误时，返回值为 0，或者根据函数，返回值为 +HUGE_VAL、+HUGE_VALF 或者 +HUGE_VALL。

参　数　　d　　要计算指数函数的浮点数

例

```
#include <math.h>
double d, ret;
   ret=exp2(d);
```

错误条件　　当 d 的绝对值太大时，发生范围错误。

对数

double expm1(double d)
float expm1f(float d)
long double expm1l(long double d)

说　明　　计算自然对数的底 e 的 d 乘方减 1 后的值。

头文件　　<math.h>

返回值　　正常：自然对数的底 e 的 d 乘方减 1 后的值
异常：在发生范围错误时，根据函数，返回值为 -HUGE_VAL、 -HUGE_VALF 或者 -HUGE_VALL。

参　数　　d　　自然对数的底 e 的指数的值

例

```
#include <math.h>
double d, ret;
   ret=expm1(d);
```

错误条件　　当 d 的值太大时，发生范围错误。

备　注　　即使 d 的值接近 0，也能通过 exp(x)-1 正确地计算 expm1(d)。

指数的抽出

long ilogb(double d)
long ilogbf(float d)
long ilogbl(long double d)

说　明　　抽出 d 的指数。

头文件　　<math.h>

返回值　　正常：d 的指数函数值
　　当 d 为 ∞ 时，为 INT_MAX。
　　当 d 为非数值时，为 FP_ILOGBNAN。
　　当 d 为 0 时，为 FP_ILOGBNAN。
异常：当 d 为 0 并且发生范围错误时，为 FP_ILOGB0。

参　数　　d　　要抽出指数的值

例

```
#include <math.h>
double d;
int ret;
   ret = ilogb(d);
```

错误条件　　当 d 的值为 0 时，可能发生范围错误。

对数

double log1p(double d)
float log1pf(float d)
long double log1pl(long double d)

说　明　　计算以 e 为底的 d 加上 1 的自然对数。

头文件　　<math.h>

返回值　　正常：d 加上 1 的自然对数
异常：当发生定义域错误时，为非数值。
当发生范围错误时，根据函数，为 -HUGE_VAL、 -HUGE_VALF 或者 -HUGE_VALL。

参　数　　d　　要计算参数加上 1 的自然对数的值

例

```
#include <math.h>
double d;
double ret;
   ret = log1p(d);
```

错误条件　　当 d 的值小于 -1 时，发生定义域错误。
当 d 的值为 -1 时，发生范围错误。

备　注　　即使 d 的值接近 0，也能通过 log(1+d) 正确地计算 log1p(d)。

对数的抽出

double log2(double d)
float log2f(float d)
long double log2l(long double d)

说　明　　计算以 2 为底的 d 的对数。

头文件　　<math.h>

返回值　　正常：以 2 为底的 d 的对数
异常：当发生定义域错误时，为非数值。

参　数　　d　　要计算对数的值

例

```
#include <math.h>
double d;
int ret;
    ret = log2(d);
```

错误条件　　当 d 的值为负值时，发生定义域错误。

指数的抽出

double logb(double d)
float logbf(float d)
long double logbl(long double d)

说　明　　作为浮点值，抽出 d 的浮点数内部表示的指数。

头文件　　<math.h>

返回值　　正常：d 的带符号的指数
异常：当发生范围错误时，根据函数，为 -HUGE_VAL、-HUGE_VALF 或者 -HUGE_VALL。

参　数　　d　　要抽出指数的值

例

```
#include <math.h>
double d, ret;
    ret = logb(d);
```

错误条件　　当 d 的值为 0 时，可能发生范围错误。

备　注　　d 总是作为正规化的值进行处理。

浮点和 FLT_RADIX 的乘法运算

double scalbn(double d, long e)
float scalbnf(float d, long e)
long double scalbnl(long double d, long e)
double scalbln(double d, long e)
float scalblnf(float d, long int e)
long double scalblnl(long double d, long int e)

说　明　　计算浮点数中为整数的基数的乘方。

头文件　　<math.h>

返回值　　正常：与 d 乘 FLT_RADIX 后的值等效的值
异常：当发生范围错误时，根据函数，为 -HUGE_VAL、 -HUGE_VALF 或者 -HUGE_VALL。

参　数　　d　　要乘 (FLT_RADIX 的 e 乘方) 的值
e　　FLT_RADIX 乘方时的指数值

例
```
#include <math.h>
double d, ret;
int e;
   ret = scalbn(d,e);
```

错误条件　　当 d 的值为 0 时，可能发生范围错误。

备　注　　实际上不计算以 e 为指数的 FLT_RADIX 的乘方。

立方根

double cbrt(double d)
float cbrtf(float d)
long double cbrtl(long double d)

说　明　　计算浮点值的立方根。

头文件　　<math.h>

返回值　　d 的立方根值

参　数　　d　　要计算立方根的值

例
```
#include <math.h>
double d, ret;
   ret = cbrt(d);
```

欧几里得距离

double hypot(double d, double e)
float hypotf(float d, double e)
long double hypotl(long double d, double e)

说　明　　计算浮点值的平方和的平方根。

头文件　　<math.h>

返回值　　正常：d 平方加上 e 平方的平方根函数值
异常：当发生范围错误时，根据函数，为 HUGE_VAL、 HUGE_VALF 或者 HUGE_VALL。

参　数　　d、e　　要计算平方和的平方根的值

例
```
#include <math.h>
double d, e, ret;
    ret = hypot(d, e);
```

错误条件　　当结果发生上溢时，可能发生范围错误。

误差

double erf(double d)
float erff(float d)
long double erfl(long double d)

说　明　　计算浮点值的误差函数。

头文件　　<math.h>

返回值　　d 的误差函数值

参　数　　d　　要计算误差函数值的值

例
```
#include <math.h>
double d, ret;
    ret = erf(d);
```

补余误差

double erfc(double d)
float erfcf(float d)
long double erfcl(long double d)

说　明　　计算浮点值的补余误差函数。

头文件　　<math.h>

返回值　　d 的补余误差函数值

参　数　　d　　要计算补余误差函数值的值

例

```
#include <math.h>
double d, ret;
    ret = erfc(d);
```

错误条件　　当 d 的绝对值太大时，发生范围错误。

γ 函数的对数

double lgamma(double d)
float lgammaf(float d)
long double lgammal(long double d)

说　明　　计算浮点值的 γ 函数的对数。

头文件　　<math.h>

返回值　　正常：d 的 γ 函数的对数值
异常：当发生定义域错误时，为附加正确数学符号的 HUGE_VAL、 HUGE_VALF 或者 HUGE_VALL。
当发生范围错误时，为 +HUGE_VAL、 +HUGE_VALF 或者 +HUGE_VALL。

参　数　　d　　要计算 γ 函数的对数值的值

例

```
#include <math.h>
double d, ret;
    ret = lgamma(d);
```

错误条件　　当 d 的绝对值太大或者太小时，发生范围错误。
当 d 的值为负整数或者 0 并且无法表示计算结果时，发生定义域错误。

γ 函数

double tgamma(double d)
float tgammaf(float d)
long double tgammal(long double d)

说　明　　计算浮点值的 γ 函数。

头文件　　<math.h>

返回值　　正常：d 的 γ 函数值
异常：当发生定义域错误时，为附加和 d 相同符号的 HUGE_VAL、HUGE_VALF 或者 HUGE_VALL。
当发生范围错误时，为 0，或者根据函数，为附加正确数学符号的 +HUGE_VAL、+HUGE_VALF 或者 +HUGE_VALL。

参　数　　d　　要计算 γ 函数值的值

例

```
#include <math.h>
double d, ret;
    ret = tgamma(d);
```

错误条件　　当 d 的绝对值太大或者太小时，发生范围错误。
当 d 的值为负整数或者 0 并且无法表示计算结果时，发生定义域错误。

整数的转换

double nearbyint(double d)
float nearbyintf(float d)
long double nearbyintl(long double d)

说　明　　根据舍入方向，将浮点值向浮点格式的整数值舍入。

头文件　　<math.h>

返回值　　向 d 的浮点格式的整数值舍入后的值

参　数　　d　　要向浮点格式的整数值舍入的值

例

```
#include <math.h>
double d, ret;
    ret = nearbyint (d);
```

备　注　　nearbyint 函数群不生成“不正确结果”的浮点异常。

整数的转换

double rint(double d)
float rintf(float d)
long double rintl(long double d)

说　明　　根据舍入方向，将浮点值向浮点格式的整数值舍入。

头文件　　<math.h>

返回值　　向 d 的浮点格式的整数值舍入后的值

参　数　　d　　要向浮点格式的整数值舍入的值

例

```
#include <math.h>
double d, ret;
    ret = rint (d);
```

备　注　　rint 函数群生成“不正确结果”的浮点异常，只有这一点不同于 nearbyint 函数群。

整数的转换

long int lrint (double d)
long int lrintf(float d)
long int lrintl(long double d)
long long int llrint (double d)
long long int llrintf(float d)
long long int llrintl(long double d)

说　明　　根据舍入方向，将浮点值向最接近的整数值舍入。

头文件　　<math.h>

返回值　　正常：d 向整数值舍入后的值
异常：当发生范围错误时，为不定值。

参　数　　d　　要向整数舍入的值

例

```
#include <math.h>
double d;
long int ret;
   ret = lrint (d);
```

错误条件　　当 d 的绝对值太大时，可能发生范围错误。

备　注　　未规定舍入后的值超出返回值型范围时的返回值。

整数的转换

double round(double d)
float roundf(float d)
long double roundl(long double d)
long int lround(double d)
long int lroundf(float d)
long int lroundl(long double d)
long long int llround (double d)
long long int llroundf(float d)
long long int llroundl(long double d)

说　明　　将浮点值向最接近的整数值舍入。

头文件　　<math.h>

返回值　　正常：d 向整数值舍入后的值
异常：当发生范围错误时，为不定值。

参　数　　d　　要向整数舍入的值

例

```
#include <math.h>
double d;
long int ret;
   ret = lround(d);
```

错误条件　　当 d 的绝对值太大时，可能发生范围错误。

备　注　　当 d 的值在中间时，与当时的舍入方向无关， lround 函数群将 d 向离 0 远的方向舍入。
未规定舍入后的值超出返回值型范围时的返回值。

整数的转换

double trunc(double d)
float truncf(float d)
long double truncl(long double d)

说　明　　将浮点值向最接近的浮点格式的整数值舍入。

头文件　　<math.h>

返回值　　舍去 d 后的浮点格式的整数值

参　数　　d　　要向浮点格式的整数舍入的值

例

```
#include <math.h>
double d, ret;
    ret = trunc(d);
```

备　注　　trunc 函数群使舍入后的值的绝对值不大于 d 的绝对值。

浮点余数的计算

double remainder(double d1, double d2)
float remainderf(float d1, float d2)
long double remainderl(long double d1, long double d2)

说　明　　计算浮点值和浮点值的余数。

头文件　　<math.h>

返回值　　d1 和 d2 的余数值

参　数　　d1　　要计算余数的值
d2

例

```
#include <math.h>
double d1, d2, ret;
   ret = remainder(d1, d2);
```

备　注　　remainder 函数群的余数计算符合 IEEE 60559 的规定。

浮点余数的计算

double remquo(double d1, double d2, long *q)
float remquof(float d1, float d2, long *q)
long double remquol(long double d1, long double d2, long *q)

说　明　　将浮点值向最接近的整数值舍入。

头文件　　<math.h>

返回值　　d1 和 d2 的余数值

参　数　　d1　　要向整数舍入的值
d2
q　　指向保存余数计算结果的商的值

例
```
#include <math.h>
double d1, d2, ret;
long q;
   ret = remquo(d1, d2, &q);
```

备　注　　保存到 q 的值有和 x/y 相同符号的并且被 2n （n 为大于等于 3 的实现定义的整数值）整除的 x/y 的整数商。

符号的复制

double copysign(double d1, double d2)
float copysignf(float d1, float d2)
long double copysignl(long double d1, long double d2)

说　明　　生成绝对值和 d1 相等并且符号位和 d2 相等的值。

头文件　　<math.h>

返回值　　正常：d1 的绝对值、 d2 的符号值
异常：当发生范围错误时，为不定值。

参　数　　d1　　要生成的绝对值的值
d2　　要生成的符号

例
```
#include <math.h>
double d1, d2, ret;
   ret = copysign(d1, d2);
```

备　注　　当 d1 为非数值时， copysign 函数群生成和 d2 的符号位相同的非数值。

非数值

double nan(const char *c)
float nanf(const char *c)
long double nanl(const char *c)

说　明　　返回非数值。

头文件　　<math.h>

返回值　　有 c 指向的内容的 qNaN 或者 0（不支持 qNaN 的情况）

参　数　　c　　字符串的指针

例

```
#include <math.h>
double ret;
const char *c;
   ret = nan(c);
```

备　注　　nan("c 字符串 ") 的调用和 strtod("NAN(c 字符串)", (char**)NULL) 等效。Nanf 和 nanl 的调用分别与 strtof 和 strtold 的调用等效。

double nextafter(double d1, double d2)
float nextafter f(float d1, float d2)
long double nextafterl(long double d1, long double d2)

说　明　　从 d1 朝着 d2 的方向，计算实数轴上紧接在 d1 之后的下一个表示的浮点数。

头文件　　<math.h>

返回值　　正常：能表示的浮点值
异常：当发生范围错误时，根据函数，为附加正确数学符号的 HUGE_VAL、 HUGE_VALF 或者 HUGE_VALL。

参　数

d1	实数轴上的浮点值
d2	从 d1 开始能表示的浮点值存在方向的值

例

```
#include <math.h>
double d1, d2, ret;
    ret = nextafter(d1, d2);
```

错误条件　　在 d1 是能以该型表示的最大有限值并且返回值为无穷大时或者在不能以该型表示时，可能发生范围错误。

备　注　　当 d1 和 d2 相等时， nextafter 函数群返回 d2。

double nexttoward(double d1, long double d2)
float nexttowardf(float d1, long double d2)
long double nexttowardl(long double d1, long double d2)

说　明　　从 d1 朝着 d2 的方向，计算实数轴上紧接在 d1 之后的下一个表示的浮点数。

头文件　　<math.h>

返回值　　正常：能表示的浮点值。
异常：当发生范围错误时，根据函数，为附加正确数学符号的 HUGE_VAL、 HUGE_VALF 或者 HUGE_VALL。

参　数

d1	实数轴上的浮点值
d2	从 d1 开始能表示的浮点值存在方向的值

例

```
#include <math.h>
double d1, ret;
long double d2;
   ret = nexttoward(d1, d2);
```

错误条件　　在 d1 是能以该型表示的最大有限值并且返回值为无穷大时或者在不能以该型表示时，可能发生范围错误。

备　注　　当 d2 的值为 long double 并且 d1 和 d2 相等时， nexttoward 函数群在根据函数转换 d2 后返回，其他和 nextafter 函数群等效。

正的差分

double fdim(double d1, double d2)
float fdimf(float d1, float d2)
long double fdiml(long double d1, long double d2)

说　明　　计算 2 个参数之间的正差分。

头文件　　<math.h>

返回值　　正常：2 个参数之间的正差分
异常：当发生范围错误时，为 HUGE_VAL、 HUGE_VALF 或者 HUGE_VALL。

参　数　　d1　　要计算正差分的值
d2

例　　
```
#include <math.h>
double d1, d2, ret;
   ret = fdim(d1, d2);
```

错误条件　当返回值发生上溢时，可能发生范围错误。

最大值

double fmax(double d1, double d2)
float fmaxf(float d1, float d2)
long double fmaxl(long double d1, long double d2)

说　明　　计算 2 个参数中的较大值。

头文件　　<math.h>

返回值　　2 个参数中的较大值

参　数　　d1　　要比较大小的值
d2

例　　
```
#include <math.h>
double d1, d2, ret;
   ret = fmax(d1, d2);
```

备　注　　fmax 函数群将非数值识别为缺乏数据的值。如果一个参数为非数值而另一个参数为数值，就返回数值。

最小值

double fmin(double d1, double d2)
float fminf(float d1, float d2)
long double fminl(long double d1, long double d2)

说　明　　计算 2 个参数中的较小值。

头文件　　<math.h>

返回值　　2 个参数中的较小值

参　数　　d1　　要比较大小的值
d2

例
```
#include <math.h>
double d1, d2, ret;
    ret = fmin(d1, d2);
```

备　注　　fmin 函数群将非数值识别为缺乏数据的值。如果一个参数为非数值而另一个参数为数值，就返回数值。

积与和

double fma(double d1, double d2, double d3)
float fmaf(float d1, float d2, float d3)
long double fmal(long double d1, long double d2, long double d3)

说　明　　将 (d1*d2)+d3 作为一个 3 元运算进行计算。

头文件　　<math.h>

返回值　　将 (d1*d2)+d3 作为 3 项运算进行计算的结果

参　数　　d1、d2、d3　　浮点值

例
```
#include <math.h>
double d1, d2, ret;
    ret = fma(d1, d2);
```

备　注　　fma 函数群将计算结果作为无限精度的值进行计算，根据 FLT_ROUNDS 的值所示的舍入模式，只进行 1 次舍入。

(9) <mathf.h>

进行各种数值的计算。
在 <mathf.h> 中声明了 ANSI 规格规定外的单精度格式的数学函数并且进行了宏定义。
各函数接受 float 型的实际参数，返回 float 型的值。

以下常数 （宏）全部为实现定义：

种类	定义名	说明
常数 （宏）	EDOM	这是在输入给函数的参数值超出函数内定义值的范围时设定给 errno 的值。
	ERANGE	这是在函数的计算结果不能表示为 float 型的值或者发生上溢 / 下溢时设定给 errno 的值。
	HUGE_VALF	这是在函数的计算结果发生上溢时，作为函数的返回值返回的值。
函数	acosf	计算浮点值的反余弦。
	asinf	计算浮点值的反正弦。
	atanf	计算浮点值的反正切。
	atan2f	计算浮点值除浮点值后的结果值的反正切。
	cosf	计算浮点值的弧度值的余弦。
	sinf	计算浮点值的弧度值的正弦。
	tanf	计算浮点值的弧度值的正切。
	coshf	计算浮点值的双曲余弦。
	sinhf	计算浮点值的双曲正弦。
	tanhf	计算浮点值的双曲正切。
	expf	计算浮点值的指数函数。
	frexpf	将浮点值分为 [0.5, 1.0) 的值和 2 的乘方的积。
	ldexpf	计算浮点值和 2 的乘方的积。
	logf	计算浮点值的自然对数。
	log10f	计算以 10 为底的浮点值的对数。
	modff	将浮点值分为整数和小数。
	powf	计算浮点值的乘方。
	sqrtf	计算浮点值的正平方根。
	ceilf	求上舍入浮点值的小数部分后的整数值。
	fabsf	计算浮点值的绝对值。
	floorf	求舍去浮点值的小数部分后的整数值。
	fmodf	计算浮点值除浮点值后的结果的余数。

以下说明发生错误时的运行：

1. 定义域错误
   当输入给函数的参数值超出函数内定义值的范围时，发生定义域错误。此时，将EDOM的值设定到errno。函数的返回值为实现定义。
2. 范围错误
   当函数中的计算结果不能表示为float型的值时，发生范围错误。此时，将ERANGE的值设定到errno。当计算结果发生上溢时，将和正确计算时相同符号的HUGE_VALF的值作为返回值返回；当计算结果发生下溢时，将0作为返回值返回。

【注】 1. 在可能因 <mathf.h> 函数的调用而发生定义域错误时，直接使用结果值很危险，必须在检查 errno 后再使用。
例：

```
                .
                .
                .
1           x=asinf(a);
2           if (errno==EDOM)
3               printf("error\n");
4           else
5               printf("result is : %f\n",x);
                .
                .
                .
```

在第 1 行，使用 asinf 函数求反正弦值。此时，如果实际参数 a 的值超出 asinf 函数的定义域 [–1.0,1.0] 的范围，就将 EDOM 的值设定到 errno。在第 2 行，判断是否发生定义域错误，如果发生定义域错误，就在第 3 行输出 error，否则就在第 5 行输出反正弦值。

2. 是否发生范围错误取决于编译程序规定的浮点型内部表示格式。例如，在采用能将无穷大作为值表示的内部表示格式时，不发生范围错误而能执行 <mathf.h> 的库函数。

实现定义的规格

	项目	编译程序的规格
1	输入给数学函数的实际参数超出范围时的数学函数的返回值	返回非数值，非数值的格式请参照 “9.1.3 浮点型的规格”。
2	在数学函数中发生下溢错误时，是否将宏 “ERANGE” 的值设定到 “errno”。	不设定。
3	当 fmodf 函数中的第 2 个实际参数的值为 0 时，是否发生范围错误。	发生范围错误。

反余弦

float acosf(float f)

说　明　　计算浮点值的反余弦。

头文件　　<mathf.h>

返回值　　正常：f 的反余弦值
异常：在发生定义域错误时，返回非数值。

参　数　　f　　要计算反余弦的浮点值

例
```
#include <mathf.h>
float f, ret;
   ret=acosf(f);
```

错误条件　　当 f 的值超出 [-1.0,1.0] 的范围时，发生定义域错误。

备　注　　acosf 函数返回值的范围为 [0,π]。

反正弦

float asinf(float f)

说　明　　计算浮点值的反正弦。

头文件　　<mathf.h>

返回值　　正常：f 的反正弦值
异常：在发生定义域错误时，返回非数值。

参　数　　f　　要计算反正弦的浮点值

例
```
#include <mathf.h>
float f, ret;
   ret=asinf(f);
```

错误条件　　当 f 的值超出 [-1.0,1.0] 的范围时，发生定义域错误。

备　注　　asinf 函数返回值的范围为 [-π/2,π/2]。

反正切

float atanf(float f)

说　明　　计算浮点值的反正切。

头文件　　<mathf.h>

返回值　　f 的反正切值

参　数　　f　　要计算反正切的浮点值

例

```
#include <mathf.h>
float f, ret;
   ret=atanf(f);
```

备　注　　atanf 函数返回值的范围为 (-π/2,π/2)。

除法运算后的反正切

float atan2f(float y, float x)

说　明　　计算浮点值除浮点值后的结果值的反正切。

头文件　　<mathf.h>

返回值　　正常：y 除 x 的反正切值
异常：在发生定义域错误时，返回非数值。

参　数

x	除数
y	被除数

例

```
#include <mathf.h>
float x, y, ret;
   ret=atan2f(y,x);
```

错误条件　　当 x 和 y 的值都为 0.0 时，发生定义域错误。

备　注　　atan2f 函数返回值的范围为 (-π,π)。atan2f 函数表示的含义如图 9.5 所示，atan2f 函数的结果是求通过点 (x, y) 和原点的直线与 x 轴的角度。当 y=0.0 并且 x 为负时，结果为 π；当 x=0.0 时，根据 y 值的正负，结果为 ±π/2。

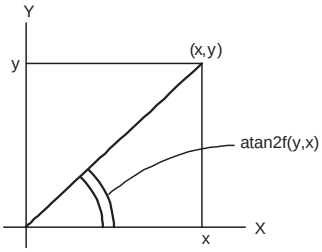


图 9.5　atan2f 函数的含义

余弦

float cosf(float f)

说　明　　计算浮点值的弧度值的余弦。

头文件　　<mathf.h>

返回值　　f 的余弦值

参　数　　f　　要计算余弦的弧度值

例

```
#include <mathf.h>
float f, ret;
   ret=cosf(f);
```

正弦

float sinf(float f)

说　明　　计算浮点值的弧度值的正弦。

头文件　　<mathf.h>

返回值　　f 的正弦值

参　数　　f　　要计算正弦的弧度值

例

```
#include <mathf.h>
float f, ret;
   ret=sinf(f);
```

正切

float tanf(float f)

说　明　　计算浮点值的弧度值的正切。

头文件　　<mathf.h>

返回值　　f 的正切值

参　数　　f　　要计算正切的弧度值

例
```
#include <mathf.h>
float f, ret;
   ret=tanf(f);
```

双曲余弦

float coshf(float f)

说　明　　计算浮点值的双曲余弦。

头文件　　<mathf.h>

返回值　　f 的双曲余弦值

参　数　　f　　要计算双曲余弦的浮点值

例
```
#include <mathf.h>
float f, ret;
   ret=coshf(f);
```

双曲正弦

float sinhf(float f)

说　明　　计算浮点值的双曲正弦。

头文件　　<mathf.h>

返回值　　f 的双曲正弦值

参　数　　f　　要计算双曲正弦的浮点值

例

```
#include <mathf.h>
float f, ret;
   ret=sinhf(f);
```

双曲正切

float tanhf(float f)

说　明　　计算浮点值的双曲正切。

头文件　　<mathf.h>

返回值　　f 的双曲正切值

参　数　　f　　要计算双曲正切的浮点值

例

```
#include <mathf.h>
float f, ret;
   ret=tanhf(f);
```

指数函数

float expf(float f)

说　明　　计算浮点值的指数函数。

头文件　　<mathf.h>

返回值　　f 的指数函数值

参　数　　f　　要计算指数函数的浮点值

例

```
#include <mathf.h>
float f, ret;
   ret=expf(f);
```

将浮点值分为尾数和指数

float frexpf(float value, long *exp)

说　明　　将浮点值分为 [0.5,1.0) 的值和 2 的乘方的积。

头文件　　<mathf.h>

返回值　　当 value 为 0.0 时　　：0.0
当 value 不为 0.0 时　：ret*2^exp 的指向的区域的值 = 用 value 定义的 ret 值

参　数　　value　　要分为 [0.5,1.0) 的值和 2 的乘方的积的浮点值
exp　　指向保存 2 的乘方值的存储区的指针

例

```
#include <mathf.h>
float ret, value;
long *exp;
   ret=frexpf(value,exp);
```

备　注　　frexpf 函数将 value 分为 [0.5,1.0) 的值和 2 的乘方的积。将分解后的结果的 2 的乘方值设定到 exp 指向的区域。
返回值 ret 的范围为 [0.5,1.0) 或者 0.0。
如果 value 为 0.0， exp 指向 int 型存储区的内容和 ret 的值就为 0.0。

将尾数和指数转换为浮点值

float ldexpf(float e, long f)

说　明　　计算浮点值和 2 的乘方的积。

头文件　　<mathf.h>

返回值　　e*2^f 运算结果的值

参　数　　e　　要计算 2 的乘方值的浮点值
f　　2 的乘方值

例
```
#include <mathf.h>
float ret, e;
int  f;
   ret=ldexpf(e, f);
```

自然对数

float logf(float f)

说　明　　计算浮点值的自然对数。

头文件　　<mathf.h>

返回值　　正常：f 的自然对数的值
异常：在发生定义域错误时，返回非数值。

参　数　　f　　要计算自然对数的浮点值

例
```
#include <mathf.h>
float f, ret;
   ret=logf(f);
```

错误条件　　当 f 的值为负时，发生定义域错误。
当 f 的值为 0.0 时，发生范围错误。

常用对数

float log10f(float f)

说　明　　计算以 10 为底的浮点值的对数。

头文件　　<mathf.h>

返回值　　正常：以 10 为底的 f 的对数值
异常：在发生定义域错误时，返回非数值。

参　数　　f　　要计算以 10 为底的对数的浮点值

例
```
#include <mathf.h>
float f, ret;
   ret=log10f(f);
```

错误条件　　当 f 的值为负值时，发生定义域错误。
当 f 的值为 0.0 时，发生范围错误。

将浮点值分为整数和小数

float modff(float a, float *b)

说　明　　将浮点值分为整数和小数。

头文件　　<mathf.h>

返回值　　a 的小数

参　数　　a　　要分为整数和小数的浮点值
b　　指向保存整数的存储区的指针

例
```
#include <mathf.h>
float a, *b, ret;
   ret=modff(a,b);
```

乘方

float powf(float x, float y)

说　明　　计算浮点值的乘方。

头文件　　<mathf.h>

返回值　　正常：x 的 y 乘方的值
异常：在发生定义域错误时，返回非数值。

参　数　　x　　被乘方的值
y　　乘方的值

例

```
#include <mathf.h>
float x, y, ret;
   ret=powf(x, y);
```

错误条件　　当 x 的值为 0.0 并且 y 的值小于等于 0.0 或者 x 的值为负值并且 y 的值不是整数值时，发生定义域错误。

平方根

float sqrtf(float f)

说　明　　计算浮点值的正平方根。

头文件　　<mathf.h>

返回值　　正常：f 的正平方根的值
异常：在发生定义域错误时，返回非数值。

参　数　　f　　要计算正平方根的浮点值

例

```
#include <mathf.h>
float f, ret;
   ret=sqrtf(f);
```

错误条件　　当 f 的值为负值时，发生定义域错误。

上舍入

float ceilf(float f)

说　明　　计算上舍入浮点值的小数部分后的整数值。

头文件　　<mathf.h>

返回值　　上舍入 f 的小数部分后的整数值

参　数　　f　　要上舍入小数部分的浮点值

例

```
#include <mathf.h>
float f, ret;
   ret=ceilf(f);
```

备　注　　ceilf 函数将大于等于 f 的最小整数值作为 float 型的值返回。因此，当 f 的值为负值时，返回舍去小数部分后的值。

绝对值

float fabsf(float f)

说　明　　计算浮点值的绝对值。

头文件　　<mathf.h>

返回值　　f 的绝对值

参　数　　f　　要计算绝对值的浮点值

例

```
#include <mathf.h>
float f, ret;
   ret=fabsf(f);
```

舍去

float floorf(float f)

说　明　　计算舍去浮点值的小数部分后的整数值。

头文件　　<mathf.h>

返回值　　舍去 f 的小数部分后的整数值

参　数　　f　　要舍去小数部分的浮点值

例
```
#include <mathf.h>
float f, ret;
   ret=floorf(f);
```

备　注　　floorf 函数将没有超出 f 值范围的最大整数值作为 float 型的值返回。因此，当 f 的值为负值时，返回上舍入小数部分后的值。

余数

float fmodf(float x, float y)

说　明　　计算浮点值除浮点值后的结果的余数。

头文件　　<mathf.h>

返回值　　当 y 的值为 0.0 时：x
当 y 的值不为 0.0 时：x 除 y 的结果的余数

参　数　　x　　被除数
y　　除数

例
```
#include <mathf.h>
float x, y, ret;
   ret=fmodf(x, y);
```

备　注　　在 fmodf 函数中，参数 x、参数 y 和返回值 ret 有以下的关系：
x=y*i+ret （i 为整数值）
返回值 ret 的符号和 x 的符号相同。
如果无法表示 x/y 的商，就无法保证结果值。

(10) <setjmp.h>

支持函数之间的控制移动。

以下宏为实现定义：

种类	定义名	说明
型 （宏）	jmp_buf	这是对应存储区的型名，用于预先保存能进行函数之间的控制移动的信息。
函数	setjmp	将当前正在执行的函数的执行环境 （由 jmp_buf 定义）保存到指定的存储区。
	longjmp	恢复 setjmp 函数保存的函数的执行环境，将控制移动到调用 setjmp 函数的程序位置。

setjmp 函数保存当前函数的执行环境。此后，能通过调用 longjmp 函数，返回到调用 setjmp 函数的程序位置。

使用 setjmp 函数和 longjmp 函数支持函数之间的控制移动的例子如下所示：

```
 1          #include <stdio.h>
 2          #include <setjmp.h>
 3          jmp_buf env;
 4          void sub();
 5          void main()
 6          {
 7
 8              if (setjmp(env)!=0){
 9                  printf("return from longjmp\n");
10                  exit(0);
11              }
12              sub();
13          }
14
15          void sub()
16          {
17              printf("subroutine is running \n");
18              longjmp(env,1);
19          }
```

【说明】

在第 8 行调用 setjmp 函数。此时，将调用 setjmp 函数的环境保存到 jmp_buf 型变量 env。此时返回值为 0，所以接着调用下一个函数 sub。

通过在函数 sub 中被调用的 longjmp 函数，恢复保存在变量 env 的环境。结果是程序犹如从第 8 行的 setjmp 函数返回。但是，此时的返回值为 longjmp 函数的第 2 个实际参数指定的值 (1)。

其结果是从第 9 行继续执行。

全局 goto 的跳转目标的设定

long setjmp(jmp_buf env)

说　明　　将当前正在执行的函数的执行环境保存到指定的存储区。

头文件　　<setjmp.h>

返回值　　当调用 setjmp 函数时　　：0
当从 longjmp 函数返回时　：非 0 值

参　数　　env　　指向要保存执行环境的存储区的指针

例　　
```
#include <setjmp.h>
int ret;
jmp_buf env;
   ret=setjmp(env);
```

备　注　　longjmp 函数使用由 setjmp 函数保存的执行环境。
调用 setjmp 函数时的返回值为 0，但是从 longjmp 函数返回时的返回值为 longjmp 函数指定的第 2 个参数的值。
在从复杂的表达式中调用 setjmp 函数时，有可能丢失一部分当前执行环境（表达式的评价中途结果等），所以只能在 setjmp 函数的结果和常数表达式比较的情况下使用 setjmp 函数，尽量不要在复杂的表达式中调用不能使用指向 setjmp 函数的指针进行间接调用。

全局 goto

void longjmp(jmp_buf env, long ret)

说　明　　恢复 setjmp 函数保存的函数的执行环境，并且将控制移动到调用 setjmp 函数的程序位置。

头文件　　<setjmp.h>

参　数　　env　　指向要保存执行环境的存储区的指针
ret　　返回给 setjmp 函数的返回码

例

```
#include <setjmp.h>
int ret;
jmp_buf env;
    longjmp(env,ret);
```

备　注　　longjmp 函数从第 1 个参数 env 指定的存储区，恢复在同一个程序中最后被调用的 setjmp 函数所保存的函数执行环境，并且将控制移动到调用该 setjmp 函数的程序位置。此时，将 longjmp 函数的第 2 个参数 ret 作为 setjmp 函数的返回值返回。但是，在 ret 为 0 时，返回给 setjmp 函数的返回值为 1。
在没有调用 setjmp 函数或者调用 setjmp 函数的函数已执行 return 语句时，无法保证 longjmp 函数的运行。

(11) <stdarg.h>

有可变个数参数的函数能参照该参数。

以下全部为实现定义：

种类	定义名	说明
型 （宏）	va_list	这是 va_start、 va_arg、 va_end 宏通用的变量型，用于参照可变个数的参数。
函数 （宏）	va_start	为了参照可变个数的参数而进行初始设定处理。
	va_arg	对于有可变个数参数的函数，能参照当前正在参照的参数的下一个参数。
	va_end	对于有可变个数参数的函数，结束参数的参照。
	va_copy	复制可变个数的参数。

使用本标准 include 文件定义的宏的程序例子如下所示：

```
1         #include <stdio.h>
2         #include <stdarg.h>
3
4         extern void prlist(int count,...);
5
6         void main()
7         {
8            prlist(1, 1);
9            prlist(3, 4, 5, 6);
10           prlist(5, 1, 2, 3, 4, 5);
11        }
12
13        void prlist(int count,...)
14        {
15           va_list ap;
16           int i;
17
18           va_start(ap,count);
19           for(i=0;i<count;i++)
20               printf("%d",va_arg(ap,int));
21           putchar('\n');
22           va_end(ap);
23        }
```

【说明】

在此例中，给第 1 个参数指定要输出的数据个数，执行函数 prlist，输出第 1 个参数指定个数的后面参数。

在第 18 行，用 va_start 对可变个数参数的参照进行初始化。然后，每输出一个参数时，就根据 va_arg 宏参照下一个参数 （第 20 行）。 va_arg 宏给第 2 个参数指定参数的型名 （此时为 int 型）。

一旦参数的参照结束，就调用 va_end 宏 （第 22 行）。

开始可变个数参数的取出

void va_start(va_list ap, parmN)

说　明　　为了参照可变个数的实际参数而进行初始设定处理。

头文件　　<stdarg.h>

参　数　　ap　　要存取可变个数参数的变量
parmN　　最右侧参数的标识符

例

```
#include <stdarg.h>
void func(int count,...)
{
   va_list ap;
     va_start(ap,count);
}
```

备　注　　va_start 宏对 va_arg 宏和 va_end 宏使用的 ap 进行初始化。
给 parmN 指定外部函数定义中参数排列的最右侧参数的标识符，即 “,...” 前一个标识符。
要参照没有名称的可变个数的参数时，需要最先执行 va_start 宏的调用。

可变个数参数的取出

type va_arg(va_list ap, type)

说　明　　对于有可变个数的实际参数的函数，可参照当前正在参照的参数的下一个参数。

头文件　　<stdarg.h>

返回值　　参数值

参　数　　ap　　要存取可变个数参数的变量
type　　要存取的参数型

例

```
#include <stdarg.h>
va_list ap;
int ret;
   ret=va_arg(ap,int);
```

备　注　　将由 va_start 宏初始化的 va_list 型变量指定为第 1 个参数。
在每次使用 va_arg 时更新 ap 的值，其结果是依次将可变个数的参数作为该宏的返回值返回。
必须给第 2 个参数 type 指定要参照的型。
ap 必须是由 va_start 初始化的 Ap。
对于 type，如果在指定函数的参数时指定了因型转换而改变长度的型（如 char 型、unsigned char 型、short 型、unsigned short 型、float 型），就不能正确地参照参数。如果指定这样的型，就无法保证运行。

结束可变个数参数的取出

void va_end(va_list ap)

说　明　　对于有可变个数的实际参数的函数，结束参数的参照。

头文件　　<stdarg.h>

参　数　　ap　　要参照可变个数参数的变量

例

```
#include <stdarg.h>
va_list ap;
   va_end(ap);
```

备　注　　ap 必须是由 va_start 初始化的 ap。
如果在函数 return 前不调用 va_end 宏，就无法保证该函数的运行。

可变个数参数的复制

void va_copy(va_list dest, va_list src)

说　明　　对于有可变个数的实际参数的函数，复制当前正在参照的参数。

头文件　　<stdarg.h>

参　数　　dest　　复制要参照可变个数参数的变量
src　　要参照可变个数参数的变量

例

```
#include <stdarg.h>
va_list ap, ap_sub;
   va_copy(ap_sub,ap);
```

备　注　　对于有由 va_start 宏初始化并且由 va_arg 使用的可变个数参数状态的第 2 个参数 src，复制到第 1 个参数 dest。
src 必须是由 va_start 初始化的 src。
能将 dest 用作以后的 va_arg 宏表示的可变个数的参数。

(12) <stdio.h>

进行有关流输入 / 输出文件的输入 / 输出的处理。

以下常数 （宏）全部为实现定义：

种类	定义名	说明
常数 （宏）	FILE	这是结构体的型，用于预先保存流输入 / 输出处理所需的缓冲区指针、错误指示符和结束指示符等各种控制信息。
	_IOFBF	作为缓冲区的使用方法，指示输入 / 输出处理要使用的全部缓冲区。
	_IOLBF	作为缓冲区的使用方法，指示输入 / 输出处理以行为单位使用的缓冲区。
	_IONBF	作为缓冲区的使用方法，指示输入 / 输出处理不使用的缓冲区。
	BUFSIZ	这是输入 / 输出处理所需的缓冲区大小。
	EOF	表示文件的结束 （End Of File），即从文件没有更多的输入内容。
	L_tmpnam*	这是保存 tmpnam 函数生成的临时文件名的字符串所需的数组长度。
	SEEK_CUR	表示将文件的当前读写位置从当前位置移动到指定的偏移位置。
	SEEK_END	表示将文件的当前读写位置从文件的结束位置移动到偏移位置。
	SEEK_SET	表示将文件的当前读写位置从文件的起始位置移动到偏移位置。
	SYS_OPEN*	这是本实现定义保证能同时打开的文件数。
	TMP_MAX*	这是 tmpnam 函数生成的固有文件名个数的最大值。
	stderr	这是指向标准错误文件的文件指针。
	stdin	这是指向标准输入文件的文件指针。
	stdout	这是指向标准输出文件的文件指针。
函数	fclose	关闭流输入 / 输出文件。
	fflush	将流输入 / 输出文件的缓冲区的内容输出到文件。
	fopen	用指定的文件名打开流输入 / 输出文件。
	freopen	关闭当前打开的流输入出文件，用指定的文件名重新打开新的文件。
	setbuf	在用户程序中定义和设定流输入 / 输出的缓冲区。
	setvbuf	在用户程序中定义和设定流输入 / 输出的缓冲区。
	fprintf	按格式将数据输出到流输入 / 输出文件。
	vfprintf	按格式将可变个数的参数列表输出到指定的流输入 / 输出文件。

【注】 * 在本实现定义中，没有被定义。

种类	定义名	说明
函数	printf	按格式转换数据并且输出到标准输出文件 （stdout）。
	vprintf	按格式将可变个数的参数列表输出到标准输出文件 （stdout）。
	sprintf	按格式转换数据并且输出到指定的区域。
	sscanf	从指定的存储区输入数据并且按格式进行转换。
	snprintf	按格式转换数据并且写到数组。
	vsnprintf	和用 va_list 替换可变个数的实际参数排列后的 snprintf 等效。
	vfscanf	和用 va_list 替换可变个数的实际参数排列后的 fscanf 等效。
	vscanf	和用 va_list 替换可变个数的实际参数排列后的 scanf 等效。
	vsscanf	和用 va_list 替换可变个数的实际参数排列后的 sscanf 等效。
	fscanf	从流输入 / 输出文件输入数据并且按格式进行转换。
	scanf	从标准输入文件 （stdin）输入数据并且按格式进行转换。
	vsprintf	按格式将可变个数的参数列表输出到指定的区域。
	fgetc	从流输入 / 输出文件输入 1 个字符。
	fgets	从流输入 / 输出文件输入字符串。
	fputc	将 1 个字符输出到流输入 / 输出文件。
	fputs	将字符串输出到流输入 / 输出文件。
	getc	（宏）从流输入 / 输出文件输入 1 个字符。
	getchar	（宏）从标准输入文件输入 1 个字符。
	gets	从标准输入文件输入字符串。
	putc	（宏）将 1 个字符输出到流输入 / 输出文件。
	putchar	（宏）将 1 个字符输出到标准输出文件。
	puts	将字符串输出到标准输出文件。
	ungetc	给流输入 / 输出文件返回 1 个字符。
	fread	将数据从流输入 / 输出文件输入到指定的存储区。
	fwrite	将数据从存储区输出到流输入 / 输出文件。
	fseek	移动流输入 / 输出文件的当前读写位置。
	ftell	取得流输入 / 输出文件的当前读写位置。
	rewind	将流输入 / 输出文件的当前读写位置移动到文件的起始位置。
	clearerr	清除流输入 / 输出文件的错误状态。
	feof	判断流输入 / 输出文件是否结束。
	ferror	判断流输入 / 输出文件是否为错误状态。
	perror	将与错误号对应的错误信息输出到标准错误文件 （stderr）。
型	fpos_t	这是能指定文件中任意位置的型。
常数 （宏）	FOPEN_MAX	这是能同时打开的文件数。
	FILENAME_MAX	这是能保持的文件名的最大长度。

实现定义的规格

	项目	编译程序的规格
1	是否需要表示输入文本最后行结束的换行字符	未规定，取决于低级接口例程的规格。
2	在读时，是否读换行字符前的空格字符。	
3	给二进制文件的写数据附加的空字符数	
4	追加模式中的文件位置说明符的初始值	
5	是否因文本文件的输出而丢失以后的文件数据	
6	文件缓冲规格	
7	是否存在长度为 0 的文件	
8	合法文件名的构成规则	
9	是否能同时打开相同的文件	
10	fprintf 函数中的 %p 格式转换的输出格式	为 16 进制数输出。
11	fscanf 函数中的 %p 格式转换的输入格式 fscanf 函数中的转换字符 “—” 的含义	为 16 进制数输入。 如果 “—” 不在开头、末尾或者 “^” 的后面，就表示前一个字符和后一个字符的范围。
12	fgetpos、 ftell 函数设定的 errno 的值	不支持 fgetpos 函数。 对 ftell 函数没有规定。 取决于低级接口例程的规格。
13	perror 函数生成的信息输出格式	信息的输出格式如 (a) 所示。

(a) perror 函数的输出格式：

< 字符串 > : < 设定给 error 的错误号所对应的错误信息 >

(b) 用 printf 函数和 fprintf 函数显示浮点的无穷大和非数值时的格式如表 9.31 所示。

表 9.31 无穷大和非数值的显示格式

	值	显示格式
1	正无穷大	++++++
2	负无穷大	------
3	非数值	******

对流输入 / 输出文件进行一系列输入 / 输出处理的程序例子如下所示：

```
1        #include <stdio.h>
2
3        void main()
4        {
5             int c;
6             FILE *ifp, *ofp;
7
8             if ((ifp=fopen("INPUT.DAT","r"))==NULL){
9                fprintf(stderr,"cannot open input file\n");
10               exit(1);
11       }
12            if ((ofp=fopen("OUTPUT.DAT","w"))==NULL){
13               fprintf(stderr,"cannot open output file\n");
14               exit(1);
15            }
16            while ((c=getc(ifp))!=EOF)
17               putc(c, ofp);
18            fclose(ifp);
19            fclose(ofp);
20       }
```

【说明】

这是将文件 INPUT.DAT 的内容复制到文件 OUTPUT.DAT 的程序。

通过第 8 行的 fopen 函数打开输入文件 INPUT.DAT，通过第 12 行的 fopen 函数打开输出文件 OUTPUT.DAT。如果打开失败，就将 NULL 作为 fopen 函数的返回值返回，然后输出错误信息，结束程序。

在 fopen 函数正常结束时，返回指向被打开文件的信息保存数据 （FILE 型）的指针，将它们设定到变量 ifp 和变量 ofp。

如果打开成功，就使用这些 FILE 型数据进行输入和输出。

一旦文件处理结束，就通过 fclose 函数关闭文件。

文件的关闭

long fclose(FILE *fp)

说　明　　关闭流输入 / 输出文件。

头文件　　<stdio.h>

返回值　　正常：0
异常：非 0 值

参　数　　fp　　文件指针

例

```
#include <stdio.h>
FILE *fp;
int ret;
   ret=fclose(fp);
```

备　注　　fclose 函数关闭文件指针 fp 指向的流输入 / 输出文件。
如果流输入 / 输出文件的输出文件为打开状态并且在缓冲区中存在未输出的数据， fclose 函数就将其输出到文件后关闭。
在系统自动分配输入 / 输出缓冲区时，释放此区域。

缓冲区的转储清除

long fflush(FILE *fp)

说　明　　将流输入 / 输出文件的缓冲区内容输出到文件。

头文件　　<stdio.h>

返回值　　正常：0
异常：非 0 值

参　数　　fp　　文件指针

例

```
#include  <stdio.h>
FILE *fp;
int ret;
   ret=fflush(fp);
```

备　注　　在流输入 / 输出文件的输出文件为打开状态时， fflush 函数将文件指针 fp 指向的流输入 / 输出文件缓冲区中未输出的内容输出到文件。在输入文件为打开状态时，将 ungetc 函数的指定设定为无效。

文件的打开

FILE *fopen(const char *fname, const char *mode)

说　明　　用指定的文件名打开流输入 / 输出文件。

头文件　　<stdio.h>

返回值　　正常：指向被打开文件信息的文件指针
异常：NULL

参　数　　fname　　指向表示文件名的字符串的指针
mode　　指向表示文件存取模式的字符串的指针

例

```
#include <stdio.h>
FILE *ret;
const char *fname, *mode;
   ret=fopen(fname,mode);
```

备　注　　fopen 函数打开以 fname 指向的字符串为文件名的流输入 / 输出文件。在用写模式或者追加模式打开不存在的文件时，尽量建立新的文件；在用写模式打开已存在的文件时，从文件的起始位置进行写操作，删除以前写的文件内容。
对于用追加模式打开的文件，从该文件的结束位置开始写处理；对于用更新模式打开的文件，能对此文件进行输入处理和输出处理。
但是在进行输出处理后，如果不执行 fflush、fseek、rewind 函数，就无法继续进行输入处理。
同样，在进行输入处理后，如果不执行 fflush、fseek、rewind 函数，就无法继续进行输出处理。
另外，还能在表示文件存取模式的字符串后面附加指示打开方法的字符。

文件的重新打开

FILE *freopen(const char *fname, const char *mode, FILE *fp)

说　明　　关闭当前打开的流输入 / 输出文件，用指定的文件名重新打开新文件。

头文件　　<stdio.h>

返回值　　正常：fp
异常：NULL

参　数

fname	指向表示新文件名的字符串的指针
mode	指向表示文件存取模式的字符串的指针
fp	当前打开的流输入 / 输出文件的文件指针

例

```
#include <stdio.h>
const char *fname, *mode;
FILE *ret, *fp;
   ret=freopen(fname,mode,fp);
```

备　注　　freopen 函数首先关闭文件指针 fp 指向的流输入 / 输出文件 （即使不能正确地进行此关闭处理，也继续进行以下处理）。
接着，重新使用该 fp 指向的 FILE 结构体，将用文件名 fname 表示的文件打开为流输入 / 输出文件。
freopen 函数在限制同时打开的文件数时有效。
freopen 函数通常返回和 fp 相同的值，但是在发生错误时，返回 NULL。

缓冲区的设定

void setbuf(FILE *fp, char buf[BUFSIZ])

说　明　　在用户程序中定义和设定流输入 / 输出缓冲区。

头文件　　<stdio.h>

参　数

fp	文件指针
buf	指向缓冲区的指针

例

```
#include <stdio.h>
FILE *fp;
char buf[BUFSIZ];
setbuf(fp,buf);
```

备　注　　对于文件指针 fp 指向的流输入 / 输出文件，setbuf 函数将 buf 指向的存储区定义为输入 / 输出缓冲区。结果是使用容量为 BUFSIZ 的缓冲区进行输入 / 输出处理。

缓冲控制

long setvbuf(FILE *fp, char *buf, long type, size_t size)

说　明　在用户程序中定义和设定流输入 / 输出缓冲区。

头文件　<stdio.h>

返回值　正常：0
异常：非 0 值

参　数

fp	文件指针
buf	指向缓冲区的指针
type	缓冲区的管理方式
size	缓冲区的容量

例

```
#include <stdio.h>
FILE *fp;
char *buf;
int type, ret;
size_t size;
   ret=setvbuf(fp,buf,type,size);
```

备　注　对于文件指针 fp 指向的流输入 / 输出文件， setvbuf 函数将 buf 指向的存储区定义为输入 / 输出缓冲区。
此缓冲区有以下三种使用方法：

(a) 给 type 指定 _IOFBF 时
使用全部缓冲区进行输入 / 输出处理。

(b) 给 type 指定 _IOLBF 时
以行为单位，使用缓冲区进行输入 / 输出处理。即在写换行字符或者缓冲区满或者请求输入时，从缓冲区取输入 / 输出数据。

(c) 给 type 指定 _IONBF 时
不使用缓冲区进行输入 / 输出处理。

setvbuf 函数通常返回 0，但是，在给 type 或者 size 设定不正确的值时或者在不接受缓冲区的使用方法等请求时，返回非 0 值。

不能在关闭被打开的流输入 / 输出文件前释放缓冲区，必须在打开流输入 / 输出文件后并且在进行输入 / 输出处理前使用 setvbuf 函数。

带格式的文件输出

long fprintf(FILE *fp, const char *control [, arg] …)

说　明　　按格式将数据输出到流输入 / 输出文件。

头文件　　<stdio.h>

返回值　　正常：转换并输出的字符数
异常：负值

参　数

fp	文件指针
control	指向表示格式的字符串的指针
arg,…	按格式输出的数据排列

例

```
#include <stdio.h>
FILE *fp;
const char *control="%s";
int ret;
char buffer[]="Hello World\n";
   ret=fprintf(fp,control,buffer);
```

备　注　　fprintf 函数根据 control 指向的表示格式的字符串，对参数 arg 进行转换和编辑，将转换后的内容输出到文件指针 fp 指向的流输入 / 输出文件。
fprintf 函数通常返回转换并输出的数据个数，但是在发生错误时，返回负值。
格式的规格如下：

【格式概要】
表示格式的字符串由 2 种字符串构成。

- 一般字符
  直接输出以下转换规格字符串以外的字符。
- 转换规格
  转换规格用 % 开始的字符串指定后面参数的转换方法。转换规格的格式遵循以下规则：

对于此转换规格，如果没有实际的输出参数，就无法保证运行。如果实际输出的参数个数多于转换规格，就忽视多余的全部参数。

【转换规格的说明】
(a)　标志
指定对带符号等的输出数据的限定。能指定的标志种类和含义如**表 9.32** 所示。

表 9.32 标志的种类和含义

	种类	含义
1	–	当转换后的数据字符数小于指定的字段宽度时，在字段内将该数据向左靠紧输出。
2	+	在转换为带符号的数据时，根据该数据的符号，在转换后的数据的前头附加正号或者负号。
3	空格字符	在转换为带符号数据时，如果转换后的数据的前头没有符号，就在该数据的前头附加空格字符。 在和 “+” 一起使用时，忽视此标志。
4	#	根据表 9.34 说明的转换种类，对转换后的数据进行限定： 1. 当进行 c、 d、 i、 s、 u 转换时 忽视此标志。 2. 当进行 o 转换时 在转换后的数据的前头附加 0。 3. 当进行 x （或者 X）转换时 在转换后的数据的前头附加 0x （或者 0X）。 4. 当进行 e、 E、 f、 g、 G 转换时 即使转换后的数据中没有小数，也输出小数点。 在进行 g、 G 转换时，删除附加在转换后的数据后面的 0。

(b) 字段宽度

用任意的 10 进制数指定要输出的转换后的数据字符数。

当转换后的数据字符数小于字段宽度时，在该数据的前面附加空格字符 （如果指定 ‘-’ 标志，就在数据的后面附加空格）。

如果转换后的数据字符数大于字段宽度，就将字段宽度扩展到能输出转换结果的宽度。

另外，如果字段宽度的指定以 0 开头，就在输出数据的前头附加字符 “0” 而不是空格字符。

(c) 精度

根据表 9.34 说明的转换种类，指定转换后的数据精度。

以句点 （.）后连续 10 进制整数的格式指定精度。如果省略 10 进制整数，就假设指定 0。

在指定的精度和指定的字段宽度发生矛盾时，字段宽度的指定无效。

各转换种类和精度指定的含义如下所示：

- 当进行 d、 i、 o、 u、 x、 X 转换时
  表示转换后的数据的最小位数。
- 当进行 e、 E、 f 转换时
  表示转换后的数据小数的位数。
- 当进行 g、 G 转换时
  表示转换后的数据的最大有效位数。
- 当进行 s 转换时
  表示打印的最大字符数。

(d) 参数长度的指定

在进行 d、 i、 o、 u、 x、 X、 e、 E、 f、 g、 G 转换时（参照表 9.34），指定要转换的数据长度 （short 型、 long 型、 long long 型、 long double 型）。在进行其他转换时，忽视此指定。长度指定的种类及其含义如表 9.33 所示。

表 9.33 参数长度指定的种类及其含义

	种类	含义
1	h	在进行 d、i、o、u、x、X 转换时，将要转换的数据指定为 short 型或者 unsigned short 型。
2	l	在进行 d、i、o、u、x、X 转换时，将要转换的数据指定为 long 型、unsigned long 型或者 double 型。
3	L	在进行 e、E、f、g、G 转换时，将要转换的数据指定为 long double 型。
4	ll	在进行 d、i、o、u、x、X 转换时，将要转换的数据指定为 long long 型或者 unsigned long long 型。在进行 n 转换时，将要转换的数据指定为指向 long long 型的指针型。

(e) 转换字符

指定将要转换的数据转换为何种格式。

当要转换的数据为结构体、数组型或者指向这些型的指针时，除了对字符数组进行 s 转换以及对指针进行 p 转换以外，无法保证运行。

转换字符和转换方式如**表 9.34** 所示。如果将此表中未记述的英文字母指定为转换字符，就无法保证运行。另外，在指定英文字母以外的字符时，运行因编译程序而不同。

表 9.34 转换字符和转换方式

	转换字符	转换种类	转换方式	转换对象的数据型	有关精度的注意事项
1	d	d 转换	将 int 型数据转换为带符号的 10 进制数的字符串。d 转换和 i 转换的规格相同。	int 型	精度的指定表示至少输出的字符数。如果转换后的字符数小于精度值，就在字符串的前面附加 0。在省略精度时，假设为 1。另外，对于值为 0 的数据，如果指定精度为 0，就不输出任何内容。
2	i	i 转换		int 型	
3	o	o 转换	将 int 型数据转换为无符号的 8 进制数的字符串。	int 型	
4	u	u 转换	将 int 型数据转换为无符号的 10 进制数的字符串。	int 型	
5	x	x 转换	将 int 型数据转换为无符号的 16 进制数。16 进制字符使用 a、b、c、d、e、f。	int 型	
6	X	X 转换	将 int 型数据转换为无符号的 16 进制数。16 进制字符使用 A、B、C、D、E、F。	int 型	
7	f	f 转换	将 double 型数据转换为“[-]ddd.ddd”格式的 10 进制数的字符串。	double 型	精度的指定表示小数点后的位数。在有小数时，必须在小数点前至少输出 1 位数字。在省略精度时，假设为 6。如果指定的精度为 0，就不输出小数点和小数，舍入要输出的数据。

	转换字符	转换种类	转换方式	转换对象的数据型	有关精度的注意事项
8	e	e 转换	将 double 型数据转换为“[-]d.ddde ± dd”格式的 10 进制数的字符串。至少输出 2 位指数。	double 型	精度的指定表示小数点后的位数。对于转换后的字符，在小数点前输出 1 位数字，而在小数后输出和精度相等位数的数字。在省略精度时，假设为 6。如果指定的精度为 0，就不输出小数，舍入要输出的数据。
9	E	E 转换	将 double 型数据转换为“[-]d.dddE ± dd”格式的 10 进制数的字符串。至少输出 2 位指数。	double 型	
10	g	g 转换 （或者 G 转换）	根据指定转换值和有效位数的精度值，决定是以 f 转换格式还是以 e 转换 （或者 E 转换）格式输出 double 型数据。如果被转换的数据指数小于 –4 或者大于指定的有效位数的精度，就转换为 e 转换（或者 E 转换）格式。	double 型	精度的指定表示转换后的数据的最大有效位数。
11	G			double 型	
12	c	c 转换	将 int 型数据作为 unsigned char 型数据，转换为该数据的对应字符。	int 型	精度的指定无效。
13	s	s 转换	对于指向 char 型的指针型数据所指向的字符串，输出表示字符串结束的空字符之前的字符或者由精度指定的字符数 （但是，不输出空字符，而且转换字符串中不含空格字符、水平制表符、换行字符）。	指向 char 型的指针型	精度的指定表示要输出的字符数。如果省略精度，就输出数据指向的字符串的空字符之前的字符 （但是，不输出空字符，而且转换字符串中不含空格字符、水平制表符、换行字符）。
14	p	p 转换	将数据视为指针，转换为各编译程序定义的能打印的字符串。	指向 void 型的指针	精度的指定无效。
15	n	不转换数据。	将数据视为指向 int 型的指针型，将以前输出的数据字符数设定到此数据指向的存储区。	指向 int 型的指针型	
16	%	不转换数据。	输出 %。	无	

(f) 用 “*”指定字段宽度或者精度

能用 “*”指定字段宽度或者精度的值。此时，将对应此转换规格的参数值用作字段宽度或者精度的值。当此参数有负的字段宽度时，解释为给正的字段宽度指定了标志 “-”；当有负的精度时，解释为省略精度。

带格式的字符串输出

long snprintf(char *restrict s, size_t n, const char *restrict control [, arg] …)

说　明　　按格式转换数据并且输出到指定的区域。

头文件　　<stdio.h>

返回值　　转换的字符数

参　数

s	指向要输出数据的存储区的指针
n	要输出的字符数
control	指向表示格式的字符串的指针
arg,…	按格式输出的数据

例

```
#include <stdio.h>
char *s;
size_t n;
const char *control="%s";
int ret;
char buffer[]="Hello World\n";
ret=snprintf(s,n,control,buffer);
```

备　注　　snprintf 函数根据表示 control 指向格式的字符串，对参数 arg 进行转换和编辑，然后输出到 s 指向的存储区。
在转换并输出的字符串的最后附加空字符，而作为返回值输出的字符数中不含此空字符。格式规格的详细内容请参照 fprintf 函数。

带可变个数参数格式的字符串输出

long vsnprintf(char *restrict s, size_t n, const char *restrict control, va_list arg)

说　明　　按格式转换数据并且输出到指定的区域。

头文件　　<stdarg.h>、<stdio.h>

返回值　　转换后的字符数

参　数

s	指向要输出数据的存储区的指针
n	要输出的字符数
control	指向表示格式的字符串的指针
arg	参数列表

例

```
#include <stdarg.h>
#include <stdio.h>
char *s;
size_t n;
const char *control="%d";
int ret;

void prlist(int count ,...)
{
   va_list ap;
   int i;
   va_start(ap, count);
   for(i=0;i<count;i++) {
      ret=vsnprintf(s,control,ap);
      va_arg(ap,int);
      s += ret;
   }
}
```

备　注　　vsnprintf 函数和用 arg 替换可变个数参数后的 snprintf 等效。
必须在调用 vsnprintf 函数前通过 va_start 宏对 arg 进行初始化。
vsnprintf 函数不调用 va_end 宏。

带格式的文件输入

long fscanf(FILE *fp, const char *control [, ptr] …)

说　明　　从流输入 / 输出文件输入数据并且按格式进行转换。

头文件　　<stdio.h>

返回值　　正常：输入转换成功的数据个数
异常：在进行输入数据的转换前输入数据结束时：EOF

参　数

fp	文件指针
control	指向表示格式的字符串的指针
ptr,…	指向保存输入数据的存储区的指针

例

```
#include <stdio.h>
FILE *fp;
const char *control="%d";
int ret,buffer[10];
   ret=fscanf(fp,control,buffer);
```

备　注　　fscanf 函数从文件指针 fp 指向的流输入 / 输出文件输入数据，并且按 control 指向的格式字符串对输入的数据进行转换和编辑，将结果保存到 ptr 指向的存储区。
用于输入数据的格式规格如下所示：

【格式概要】
表示格式的字符串由以下 3 种字符构成：
- 空格字符
  如果指定空格 （' '）、水平制表符 （'\t'）或者换行字符 （'\n'），就跳读输入数据中的空格类字符。
- 一般字符
  如果指定既不是以上的空格字符也不是%的字符，就输入1个字符的数据。在此输入的字符必须和表示格式的字符串中指定的字符相同。
- 转换规格
  转换规格是以%开头的字符串指定输入数据的转换方法和保存方法 （保存到表示格式的字符串之后的参数所指向的区域）。转换规格的格式遵循以下规则：
  %[*][字段宽度][转换后的数据长度] 转换字符

对于格式中的转换规格，如果没有指向保存输入数据的存储区的指针，就无法保证运行。在格式结束后，如果还有指向保存输入数据的存储区的指针，就忽视该指针。

【转换规格的说明】
- “*”指定
  抑制将输入数据保存到参数指向的存储区。
- 字段宽度
  用10进制数字指定输入数据的最大字符数。
- 转换后的数据长度

在进行d、i、o、u、x、X、e、E、f转换时（参照表9.36），指定转换后的数据长度（short型、long型、long long型、long double型）。在进行其他转换时，忽视此指定。长度指定的种类及其含义如表9.35所示。

表 9.35 转换后的数据长度指定的种类及其含义

	种类	含义
1	h	在进行 d、i、o、u、x、X 转换时，将转换后的数据指定为 short 型。
2	l	在进行 d、i、o、u、x、X 转换时，将转换后的数据指定为 long 型。 在进行 e、E、f 转换时，将转换后的数据指定为 double 型。
3	L	在进行 e、E、f 转换时，将转换后的数据指定为 long double 型。
4	ll	在进行 d、i、o、u、x、X 转换时，将转换后的数据指定为 long long 型。

- 转换字符

  按各转换字符指定的转换种类，转换各输入数据。如果读到空格类字符或者不是转换对象的字符，或者转换字符超过指定的字段宽度，就结束处理。

表 9.36 转换字符和转换内容

	转换字符	转换种类	转换方式	对应的参数型名
1	d	d 转换	将 10 进制数字的字符串转换为整数型数据。	整数型
2	i	i 转换	将前头带符号的 10 进制数字的字符串或者最后带 u（U）或者 l（L）的 10 进制数字的字符串转换为整数型数据。将以 0x（或者 0X）开头的字符串解释为 16 进制数字，并且转换为 int 型数据；将以 0 开头的的字符串解释为 8 进制数字，并且转换为 int 型数据。	整数型
3	o	o 转换	将 8 进制数字的字符串转换为整数型数据。	整数型
4	u	u 转换	将无符号的 10 进制数字的字符串转换为整数型数据。	整数型
5	x	x 转换	将 16 进制数字的字符串转换为整数型数据。 x 转换和 X 转换的含义相同。	整数型
6	X	X 转换		
7	s	s 转换	将空格、水平制表符、换行字符之前的的内容作为 1 个字符串进行转换。在字符串的最后附加空字符（设定转换后数据的字符串需要能保存含空字符的容量）。	字符型
8	c	c 转换	输入 1 个字符。此时，即使输入的字符为空格类字符，也不跳读。 如果只读取空格类字符以外的字符，就必须指定 %1s。如果指定字段宽度，就读取其指定宽度的字符，此时，需要给用于保存转换后数据的存储区指定容量。	char 型
9	e	e 转换	将表示浮点型的字符串转换为浮点型数据。e 转换和 E 转换、g 转换和 G 转换的含义相同。输入格式为能用 strtod 函数表示的浮点型。	浮点型
10	E	E 转换		
11	f	f 转换		
12	g	g 转换		
13	G	G 转换		
14	p	p 转换	在 fprintf 函数中，将通过 p 转换进行格式转换的字符串转换为指针型数据。	指向 void 型的指针型
15	n	不转换数据。	不输入数据，设定以前输入的数据字符数。	整数型

	转换字符	转换种类	转换方式	对应的参数型名
16	[	[转换	指定 “[字符集]”。此字符集定义构成字符串的字符集。如果字符集的第一个字符不是 “^”，就在输入数据是此字符集中没有的字符时将前面的内容作为一个字符串进行输入；如果第一个字符是 “^”，就在最初读到的输入数据是此字符集中 “^” 字符除外的字符时将前面的内容作为一个字符串进行输入。在输入字符串的最后自动附加空字符（设定转换后数据的字符串需要能保存含空字符的容量）。	字符型
17	%	不转换数据。	读 %。	无

如果转换字符不是表 9.36 所示的英文字符，就无法保证运行。如果是其他字符，其运行就为实现定义。

带格式的输出

long printf(const char *control [, arg] …)

说　明　　按格式转换数据并且输出到标准输出文件（stdout）。

头文件　　<stdio.h>

返回值　　正常：转换并输出的字符数
异常：负值

参　数　　control　　指向表示格式的字符串的指针
arg,…　　按格式输出的数据

例
```
#include <stdio.h>
const char *control="%s";
int ret;
char buffer[]="Hello World\n";
   ret=printf(control,buffer);
```

备　注　　printf 函数根据表示 control 指向格式的字符串，对参数 arg 进行转换和编辑，然后输出到标准输出文件（stdout）。
格式规格的详细内容请参照 fprintf 函数。

带可变个数参数格式的文件输入

long vfscanf(FILE *restrict fp, const char *restrict control , va_list arg)

说　明　　从流输入 / 输出文件输入数据并且按格式进行转换。

头文件　　<stdarg.h>, <stdio.h>

返回值　　正常：输入转换成功的数据个数
异常：在转换输入数据前输入数据结束时：EOF

参　数

fp	文件指针
control	指向表示格式的宽字符串的指针
arg	参数列表

例

```
#include <stdarg.h>
#include <stdio.h>

FILE *fp;
const char *control="%d";
int ret;

void prlist(int count ,...)
{
   va_list ap;
   int i;
   va_start(ap, count);
   for(i=0;i<count;i++)
      ret=vfscanf(fp, control, ap);
   va_end(ap);
}
```

备　注　　vfscanf 函数和用 arg 替换可变个数参数排列后的 fscanf 等效。
必须在调用 vfscanf 函数前通过 va_start 宏对 arg 进行初始化。
vfscanf 函数不调用 va_end 宏。

带格式的输入

long scanf(const char *control [, ptr] …)

说　明　　从标准输入文件（stdin）输入数据并且按格式进行转换。

头文件　　<stdio.h>

返回值　　正常：输入转换成功的数据个数
异常：EOF

参　数

control	指向表示格式的字符串的指针
ptr,…	指向保存输入转换数据的存储区的指针

例

```
#include <stdio.h>
const char *control="%d";
int ret,buffer[10];
   ret=scanf(control, buffer);
```

备　注　　scanf 函数从标准输入文件（stdin）输入数据，并且根据表示 control 指向格式的字符串，对该数据进行转换和编辑，然后将结果保存到 ptr 指向的存储区。
scanf 函数将输入转换成功的数据个数作为返回值返回。如果在开始转换前标准输入文件结束，就返回 EOF。
格式规格的详细内容请参照 fscanf 函数。
在进行 %e 转换时，用 l 指定 double 型，用 L 指定 long double 型。默认的数据型为 float 型。

带可变个数参数格式的文件输入

long vscanf(const char *restrict control , va_list arg)

说　明　　从指定的存储区输入数据并且按格式进行转换。

头文件　　<stdarg.h>, <stdio.h>

返回值　　正常：输入转换成功的数据个数
异常：在转换输入数据前输入数据结束时：EOF

参　数　　control　　指向表示格式的字符串的指针
arg　　参数列表

例

```
#include <stdarg.h>
#include <stdio.h>

FILE *fp;
const char *control="%d";
int ret;

void prlist(int count ,...)
{
    va_list ap;
    int i;
    va_start(ap, count);
    for(i=0;i<count;i++)
       ret=vscanf(control, ap);
    va_end(ap);
}
```

备　注　　vscanf 函数和用 arg 替换可变个数参数后的 scanf 等效。
必须在调用 vscanf 函数前通过 va_start 宏对 arg 进行初始化。
vscanf 函数不调用 va_end 宏。

带格式的字符串输出

long sprintf(char *s, const char *control [, arg] …)

说　明　　按格式转换数据并且输出到指定的区域。

头文件　　<stdio.h>

返回值　　转换后的字符数

参　数

s	指向输出数据的存储区的指针
control	指向表示格式的字符串的指针
arg,…	按格式输出的数据

例

```
#include <stdio.h>
char *s;
const char *control="%s";
int ret;
char buffer[]="Hello World\n";
   ret=sprintf(s,control,buffer);
```

备　注　　sprintf 函数根据表示 control 指向格式的字符串，对参数 arg 进行转换和编辑，然后输入到 s 指向的存储区。
在转换并输出的字符串的最后附加空字符，而作为返回值输出的字符数中不包含此空字符。
格式规格的详细内容请参照 fprintf 函数。

带格式的字符串输入

long sscanf(const char *s, const char *control [, ptr] …)

说　明　　从指定的存储区输入数据并且按格式进行转换。

头文件　　<stdio.h>

返回值　　正常：输入转换成功的数据个数
异常：EOF

参　数

s	有输入数据的存储区
control	指向表示格式的字符串的指针
ptr,…	指向保存输入转换数据的存储区的指针

例

```
#include <stdio.h>
const char *s, *control="%d";
int ret,buffer[10];
   ret=sscanf(s,control,buffer);
```

备　注　　sscanf 函数从 s 指向的存储区输入数据，并且根据表示 control 指向格式的字符串，对该数据进行转换和编辑，然后将结果保存到 ptr 指向的存储区。
sscanf 函数返回输入转换成功的数据个数。如果在开始转换前输入数据结束，就返回 EOF。
格式规格的详细内容请参照 fscanf 函数。

带可变个数参数格式的文件输入

long vsscanf(const char *restrict s, const char *restrict control , va_list arg)

说　明　　从指定的存储区输入数据并且按格式进行转换。

头文件　　<stdarg.h>, <stdio.h>

返回值　　正常：输入转换成功的数据个数
异常：在转换输入数据前输入数据结束时：EOF。

参　数

s	有输入数据的存储区
control	指向表示格式的字符串的指针
arg	参数列表

例

```
#include <stdarg.h>
#include <stdio.h>

const char *s, *control="%d";
int ret;

void prlist(int count ,...)
{
   va_list ap;
   int i;
   va_start(ap, count);
   for(i=0;i<count;i++)
      ret=vsscanf(control, ap);
   va_end(ap);
}
```

备　注　　vsscanf 函数和用 arg 替换可变个数参数后的 sscanf 等效。
必须在调用 vsscanf 函数前通过 va_start 宏对 arg 进行初始化。
vsscanf 函数不调用 va_end 宏。

可变个数参数的文件输出

long vfprintf(FILE *fp, const char *control, va_list arg)

说　明　　按格式将可变个数的参数列表输出到指定的流输入 / 输出文件。

头文件　　<stdio.h>

返回值　　正常：转换并输出的字符数
异常：负值

参　数

fp	文件指针
control	指向表示格式的字符串的指针
arg	参数列表

例

```
#include <stdarg.h>
#include <stdio.h>
FILE *fp;
const char *control="%d";
int  ret;

void prlist(int count ,...)
{
   va_list ap;
   int i;
   va_start(ap, count);
   for(i=0;i<count;i++)
      ret=vfprintf(fp, control, ap);
   va_end(ap);
}
```

备　注　　vfprintf 函数根据表示 control 指向格式的字符串，依次对可变个数的参数列表进行转换和编辑，然后输出到 fp 指向的流输入 / 输出文件。
vfprintf 函数返回转换并输出的数据个数，但是在发生输出错误时，返回负值。
vfprintf 函数不调用 va_end 宏。
格式规格的详细内容请参照 fprintf 函数。
必须通过 va_start（和随后的 va_arg 宏）对表示参数列表的 arg 进行初始化。

可变个数参数的输出

long vprintf(const char *control, va_list arg)

说　明　　按格式将可变个数的参数列表输出到标准输出文件（stdout）。

头文件　　<stdio.h>

返回值　　正常：转换并输出的字符数
异常：负值

参　数　　control　　指向表示格式的字符串的指针
arg　　参数列表

例

```
#include <stdarg.h>
#include <stdio.h>
FILE *fp;
const char *control="%d";
int ret;

void prlist(int count ,...)
{
   va_list ap;
   int i;
   va_start(ap, count);
   for(i=0;i<count;i++)
      ret=vprintf(control, ap);
   va_end(ap);
}
```

备　注　　vprintf 函数根据表示 control 指向格式的字符串，依次对可变个数的参数列表进行转换和编辑，然后输出到标准输出文件。
vprintf 函数返回转换并输出的数据个数，但是在发生输出错误时，返回负值。
vprintf 函数不调用 va_end 宏。
格式规格的详细内容请参照 fprintf 函数。
必须通过 va_start（和随后的 va_arg 宏）对表示参数列表的 arg 进行初始化。

可变个数参数的字符串输出

long vsprintf(char *s, const char *control, va_list arg)

说　明　　按格式将可变个数的参数列表输出到指定的存储区。

头文件　　<stdio.h>

返回值　　正常：转换后的字符数
异常：负值

参　数

s	指向要输出数据的存储区的指针
control	指向表示格式的字符串的指针
arg	参数列表

例

```
#include <stdarg.h>
#include <stdio.h>
char *s;
const char *control="%d";
int ret;

void prlist(int count ,...)
{
   va_list ap;
   int i;
   va_start(ap, count);
   for(i=0;i<count;i++) {
      ret=vsprintf(s,control,ap);
      va_arg(ap,int);
      s += ret;
   }
}
```

备　注　　vsprintf 函数根据表示 control 指向格式的字符串，依次对可变个数的参数列表进行转换和编辑，然后输出到 s 指向的存储区。
在转换并输出的字符串的最后附加空字符，而作为返回值输出的字符数中不包含此空字符。
格式规格的详细内容请参照 fprintf 函数。
必须通过 va_start（和随后的 va_arg 宏）对表示参数列表的 arg 进行初始化。

从文件输入 1 个字符

long fgetc(FILE *fp)

说　明　　从流输入 / 输出文件输入 1 个字符。

头文件　　<stdio.h>

返回值　　正常：　当文件结束时　：EOF
　　当文件未结束时　：输入的字符
　　异常：　EOF

参　数　　fp　　文件指针

例

```
#include <stdio.h>
FILE *fp;
int ret;
   ret=fgetc(fp);
```

错误条件　　如果发生读错误时，就设定该文件的错误指示符。

备　注　　fgetc 函数从文件指针 fp 指向的流输入 / 输出文件输入 1 个字符。
fgetc 函数通常返回输入的 1 个字符，但是在文件结束或者发生错误时，返回 EOF。在文件结束时，给该文件设定文件结束指示符。

从文件输入字符串

char *fgets(char *s, long n, FILE *fp)

说　明　　从流输入 / 输出文件输入字符串。

头文件　　<stdio.h>

返回值　　正常：　当文件结束时　：NULL
　　当文件未结束时　：s
　　异常：　NULL

参　数

s	指向要输入字符串的存储区的指针
n	要输入字符串的存储区的字节数
fp	文件指针

例

```
#include <stdio.h>
char *s, *ret;
int n;
FILE *fp;
   ret=fgets(s,n,fp);
```

备　注　　fgets 函数从文件指针 fp 指向的流输入 / 输出文件，将字符串输入到指针 s 指向的存储区。
fgets 函数输入 n-1 字符或者换行字符前的字符或者文件结束前的字符，在输入字符串的最后附加空字符。
fgets 函数通常返回指向要输入字符串的存储区的指针 s，但是在文件结束时或者发生错误时，返回 NULL。
在文件结束时，s 指向的存储区的内容不变，但是在发生错误时，无法保证 s 指向的存储区的内容。

将 1 个字符输出到文件

long fputc(long c, FILE *fp)

说　明　　将 1 个字符输出到流输入 / 输出文件。

头文件　　<stdio.h>

返回值　　正常：输出的字符
异常：EOF

参　数

c	要输出的字符
fp	文件指针

例

```
#include <stdio.h>
FILE *fp;
int c, ret;
   ret=fputc(c,fp);
```

错误条件　　如果发生写错误，就对该文件设定错误指示符。

备　注　　fputc 函数将字符 c 输出到文件指针 fp 指向的流输入 / 输出文件。
fputc 函数通常返回输出的字符 c，但是在发生错误时，返回 EOF。

将字符串输出到文件

long fputs(const char *s, FILE *fp)

说　明　　将字符串输出到流输入 / 输出文件。

头文件　　<stdio.h>

返回值　　正常：0
异常：非 0 值

参　数

s	指向要输出的字符串的指针
fp	文件指针

例

```
#include <stdio.h>
const char *s;
int ret;
FILE *fp;
   ret=fputs(s,fp);
```

备　注　　fputs 函数将 s 指向的空字符前的字符串输入到文件指针 fp 指向的流输入 / 输出文件。此时，不输出表示字符串结束的空字符。
fputs 函数通过返回 0，但是在发生错误时，返回非 0 值。

从文件输入 1 个字符

long getc(FILE *fp)

说　明　　从流输入 / 输出文件输入 1 个字符。

头文件　　<stdio.h>

返回值　　正常：　当文件结束时　：EOF
　　当文件未结束时　：输入的字符
　　异常：　EOF

参　数　　fp　　文件指针

例

```
#include <stdio.h>
FILE *fp;
int ret;
   ret=getc(fp);
```

错误条件　　如果发生读错误，就对该文件设定错误指示符。

备　注　　getc 函数从文件指针 fp 指向的流输入 / 输出文件输入 1 个字符。
getc 函数通常返回输入的 1 个字符，但是在文件结束或者发生错误时，返回 EOF。在文件结束时，对该文件设定文件结束指示符。

1 个字符的输入

long getchar(void)

说　明　　从标准输入文件 （stdin）输入 1 个字符。

头文件　　<stdio.h>

返回值　　正常：　当文件结束时　：EOF
　　当文件未结束时　：输入的字符
　　异常：　EOF

例

```
#include <stdio.h>
int ret;
   ret=getchar();
```

错误条件　　如果发生读错误，就对该文件设定错误指示符。

备　注　　getchar 函数从标准输入文件 （stdin）输入 1 个字符。
getchar 函数通常返回输入的 1 个字符，但是在文件结束或者发生错误时，返回 EOF。在文件结束时，对该文件设定文件结束指示符。

字符串的输入

char *gets(char *s)

说　明　　从标准输入文件（stdin）输入字符串。

头文件　　<stdio.h>

返回值　　正常：　当文件结束时　：NULL
　　当文件未结束时　：s
　　异常：　NULL

参　数　　s　　指向要输入字符串的存储区的指针

例　　
```
#include <stdio.h>
char *ret, *s;
   ret=gets(s);
```

备　注　　gets 函数从标准输入文件（stdin）将字符串输入到以 s 开头的存储区。
gets 函数输入文件结束前的字符或者换行字符前的字符，代替换行字符附加空字符。
gets 函数通常返回指向要输入的字符串的存储区指针 s，但是在标准输入文件结束或者发生错误时，返回 NULL。
在标准输入文件结束时，s 指向的存储区的内容不变。但是在发生错误时，无法保证 s 指向的存储区的内容。

将 1 个字符输出到文件

long putc(long c, FILE *fp)

说　明　　将 1 个字符输出到流输入 / 输出文件。

头文件　　<stdio.h>

返回值　　正常：输出的字符
异常：EOF

参　数　　c　　要输出的字符
fp　　文件指针

例

```
#include <stdio.h>
FILE *fp;
int c, ret;
ret=putc(c,fp);
```

错误条件　　如果发生写错误，就对该文件设定错误指示符。

备　注　　putc 函数将字符 c 输出到文件指针 fp 指向的流输入 / 输出文件。
putc 函数通常返回输出的字符 c，但是在发生错误时，返回 EOF。

1 个字符的输出

long putchar(long c)

说　明　　将 1 个字符输出到标准输出文件 （stdout）。

头文件　　<stdio.h>

返回值　　正常：输出的字符
异常：EOF

参　数　　c　　要输出的字符

例

```
#include <stdio.h>
int c, ret;
   ret=putchar(c);
```

错误条件　　如果发生写错误，就对该文件设定错误指示符。

备　注　　putchar 函数将字符 c 输出到标准输出文件 （stdout）。 putchar 宏通常返回输出的字符 c，但是在发生错误时，返回 EOF。

字符串的输出

long puts(const char *s)

说　明　　将字符串输出到标准输出文件（stdout）。

头文件　　<stdio.h>

返回值　　正常：0
异常：非 0 值

参　数　　s　　指向要输出的字符串的指针

例
```
#include <stdio.h>
const char *s;
int ret;
   ret=puts(s);
```

备　注　　puts 函数将 s 指向的字符串输出到标准输出文件（stdout）。此时，不输出表示字符串结束的字符，而输出换行字符。
puts 函数通常返回 0，但是在发生错误时，返回非 0 值。

将 1 个字符返回到文件

long ungetc(long c, FILE *fp)

说　明　　将 1 个字符返回到流输入 / 输出文件。

头文件　　<stdio.h>

返回值　　正常：返回的字符
异常：EOF

参　数　　c　　要返回的字符
fp　　文件指针

例
```
#include <stdio.h>
int c, ret;
FILE *fp;
   ret=ungetc(c,fp);
```

备　注　　ungetc 函数将字符 c 返回到文件指针 fp 指向的流输入 / 输出文件。
在此，如果不调用 fflush、 fseek、 rewind 函数，返回的字符就为下一个输入数据。
ungetc 函数通常返回已被返回的字符 c，但是在发生错误时，返回 EOF。
对于 ungetc 函数，如果不执行 fflush、 fseek、 rewind 函数，就无法保证调用至少 2 次时的运行。如果执行 ungetc 函数，文件的当前位置指示符就回移一个位置。如果此位置指示符已处于文件的起始位置，就无法保证位置指示符。

读文件

size_t fread(void *ptr, size_t size, size_t n, FILE *fp)

说　明　　从流输入 / 输出文件将数据输入到指定的存储区。

头文件　　<stdio.h>

返回值　　当 size 或者 n 为 0 时　：0
当 size 和 n 都不为 0 时 ：输入成功的成员个数

参　数

ptr	指向要输入数据的存储区的指针
size	1 个成员的字节数
n	要输入的成员个数
fp	文件指针

例

```
#include <stdio.h>
void *ptr;
size_t size;
size_t n, ret;
FILE *fp;
   ret=fread(ptr,size,n,fp);
```

备　注　　fread 函数将 size 指定的字节数作为 1 个成员，从文件指针 fp 指向的流输入 / 输出文件将 n 个成员的数据输入到 ptr 指向的存储区。此时，文件的位置指示符向前移了输入数据的字节数。
fread 函数返回实际输入成功的成员个数，所以通常和 n 的值相同。但是在文件结束或者发生错误时，返回以前输入成功的成员个数，所以返回值小于 n，必须使用 ferror 函数和 feof 函数来区分是文件结束还是发生错误。
当 size 或者 n 为 0 时，返回值为 0， ptr 指向的存储区的内容不变。如果发生错误或者只能输入部分成员，就无法保证该文件的位置指示符。

写文件

size_t fwrite(const void *ptr, size_t size, size_t n, FILE *fp)

说　明　　从存储区将数据输出到流输入 / 输出文件。

头文件　　<stdio.h>

返回值　　输出成功的成员个数

参　数

ptr	指向保存要输出数据的存储区的指针
size	1 个成员的字节数
n	要输出的成员个数
fp	文件指针

例

```
#include <stdio.h>
const void *ptr;
size_t size;
size_t n, ret;
FILE *fp;
   ret=fwrite(ptr,size,n,fp);
```

备　注　　fwrite 函数将 size 指定的字节数作为 1 个成员，从 ptr 指向的存储区将 n 个成员的数据输出到文件指针 fp 指向的流输入 / 输出文件。
此时，文件的位置指示符向前移了输出数据的字节数。
fwrite 函数返回实际输出成功的成员个数，所以通常和 n 的值相同。但是在发生错误时，返回以前输出成功的成员个数，所以返回值小于 n。
在发生错误时，无法保证该文件的位置指示符。

文件读写位置的移动

long fseek(FILE *fp, long offset, long type)

说　明　　移动流输入 / 输出文件的当前读写位置。

头文件　　<stdio.h>

返回值　　正常：0
异常：非 0 值

参　数

fp	文件指针
offset	从偏移种类指定的位置开始的偏移量
type	偏移种类

例

```
#include <stdio.h>
FILE *fp;
long offset;
int type, ret;
   ret=fseek(fp,offset,type);
```

备　注　　fseek 函数将文件指针 fp 指向的流输入 / 输出文件的当前读写位置从偏移种类 type 指定的位置移动到 offset 字节指定的位置。
偏移种类如表 9.37 所示。
fseek 函数通常返回 0，但是在出现异常时，返回非 0 值。

表 9.37　偏移种类

	偏移种类	含义
1	SEEK_SET	从文件的起始位置移动到 offset 字节指定的位置。此时， offset 指定的值必须为 0 或者正值。
2	SEEK_CUR	从文件的当前位置移动到 offset 字节指定的位置。此时，如果 offset 指定的值为正，就向文件的后方移动；如果为负，就向文件的前方移动。
3	SEEK_END	从文件的结束位置移动到 offset 字节指定的位置。此时， offset 指定的值必须为 0 或者负值。

对于文本文件，偏移种类必须为 SEEK_SET，并且 offset 必须为 0 或者由 ftell 函数给该文件返回的值。必须注意：fseek 函数调用会取消 ungetc 函数的效果。

文件读写位置的取得

long ftell(FILE *fp)

说　明　　取得流输入 / 输出文件的当前读写位置。

头文件　　<stdio.h>

返回值　　当前位置指示符的位置 （文本文件）
从文件起始位置到当前位置的字节数 （二进制文件）

参　数　　fp　　文件指针

例

```
#include <stdio.h>
FILE *fp;
long ret;
   ret=ftell(fp);
```

备　注　　ftell 函数取得文件指针 fp 指向的流输入 / 输出文件的当前读写位置。
对于二进制文件， ftell 函数返回从文件的起始位置到当前位置的字节数；对于文本文件，返回位置指示符的位置，此位置是 fseek 函数能使用的实现定义值。
在对文本文件使用 2 次 ftell 函数后，其返回值的差并不表示实际文件中的距离。

移动到文件的起始位置

void rewind(FILE *fp)

说　明　　将流输入 / 输出文件的当前读写位置移动到文件的起始位置。

头文件　　<stdio.h>

参　数　　fp　　文件指针

例

```
#include <stdio.h>
FILE *fp;
   rewind(fp);
```

备　注　　rewind 函数将文件指针 fp 指向的流输入 / 输出文件的当前读写位置移动到文件的起始位置。
rewind 函数清除该文件的结束指示符和错误指示符。
必须注意：rewind 函数的调用会取消 ungetc 函数的效果。

错误状态的清除

void clearerr(FILE *fp)

说　明　　清除流输入 / 输出文件的错误状态。

头文件　　<stdio.h>

参　数　　fp　　文件指针

例

```
#include <stdio.h>
FILE *fp;
   clearerr(fp);
```

备　注　　clearerr 函数清除文件指针 fp 指向的流输入 / 输出文件的错误指示符和结束指示符。

文件结束的判断

long feof(FILE *fp)

说　明　　判断流输入 / 输出文件是否结束。

头文件　　<stdio.h>

返回值　　当文件结束时　　：非 0 值
当文件未结束时　　：0

参　数　　fp　　文件指针

例

```
#include <stdio.h>
FILE *fp;
int ret;
   ret=feof(fp);
```

备　注　　feof 函数判断文件指针 fp 指向的流输入 / 输出文件是否结束。
feof 函数调查所指定的流输入 / 输出文件的文件结束指示符，如果文件结束指示符被设定，就返回非 0 值，表示文件结束，否则就返回 0，表示文件未结束。

文件错误状态的判断

long ferror(FILE *fp)

说　明　　判断流输入 / 输出文件是否为错误状态。

头文件　　<stdio.h>

返回值　　当文件为错误状态时　　：非 0 值
当文件不为错误状态时　：0

参　数　　fp　　文件指针

例
```
#include <stdio.h>
FILE *fp;
int ret;
   ret=ferror(fp);
```

备　注　　ferror 函数判断文件指针 fp 指向的流输入 / 输出文件是否为错误状态。
ferror 函数调查所指定的流输入 / 输出文件的错误指示符，如果错误状态被设定，就返回非 0 值，表示文件为错误状态，否则就返回 0，表示文件不为错误状态。

错误信息的输出

void perror(const char *s)

说　明　　将错误号对应的错误信息输出到标准错误文件 （stderr）。

头文件　　<stdio.h>

参　数　　s　　指向错误信息的指针

例
```
#include <stdio.h>
const char *s;
   perror(s);
```

备　注　　perror 函数使 s 指向的错误信息和 errno 相对应，并且输出到标准错误文件 （stderr）。
如果 s 不为 NULL 并且 s 指向的字符串不为空字符，就以 s 指向的字符串、冒号、空格字符、实现定义的错误信息、换行字符的格式输出信息。

(13) <stdlib.h>

定义 C 程序中进行标准处理的函数。

以下宏为实现定义：

种类	定义名	说明
型 （宏）	onexit_t	这是通过 onexit 函数注册的函数返回型以及 onexit 函数返回值的型。
	div_t	这是 div 函数返回值的结构体型。
	ldiv_t	这是 ldiv 函数返回值的结构体型。
	lldiv_t	这是 lldiv 函数返回值的结构体型。
常数 （宏）	RAND_MAX	这是 rand 函数生成的伪随机数的整数最大值。
	EXIT_SUCCESS	表示成功结束状态。
函数	atof	将表示数的字符串转换为 double 型浮点值。
	atoi	将表示 10 进制数的字符串转换为 int 型整数值。
	atol	将表示 10 进制数的字符串转换为 long 型整数值。
	atoll	将表示 10 进制数的字符串转换为 long long 型整数值。
	strtod	将表示数的字符串转换为 double 型浮点值。
	strtof	将表示数的字符串转换为 float 型浮点值。
	strtold	将表示数的字符串转换为 long double 型浮点值。
	strtol	将表示数的字符串转换为 long 型整数值。
	strtoul	将表示数的字符串转换为 unsigned long 型整数值。
	strtoll	将表示数的字符串转换为 long long 型整数值。
	strtoull	将表示数的字符串转换为 unsigned long long 型整数值。
	rand	生成 0 ～ RAND_MAX 范围的伪随机数整数。
	srand	设定 rand 函数生成的伪随机数串的初始值。
	calloc	分配存储区，将已分配的全部存储区初始化为 0。
	free	释放指定的存储区。
	malloc	分配存储区。
	realloc	将存储区的容量更改为指定的容量。
	bsearch	进行 2 分检索。
	qsort	进行排序。
	abs	计算 int 型整数的绝对值。
	div	计算 int 型整数除法运算的商和余数。
	labs	计算 long 型整数的绝对值。
	ldiv	计算 long 型整数除法运算的商和余数。
	llabs	计算 long long 型整数的绝对值。
	lldiv	计算 long long 型整数除法运算的商和余数。

实现定义的规格

	项目	编译程序的规格
1	calloc、 malloc、 realloc 函数中长度为 0 时的操作	返回 NULL。

将字符串转换为 double 型

double atof(const char *nptr)

说　明　　将表示数的字符串转换为 double 型浮点值。

头文件　　<stdlib.h>

返回值　　转换后的 double 型浮点值

参　数　　nptr　　要转换的表示数的字符串指针

例

```
#include <stdlib.h>
const char *nptr;
double ret;
   ret=atof(nptr);
```

错误条件　　如果转换后的值发生上溢或者下溢，就设定 errno。

备　注　　对不符合浮点型格式的最初字符前的字符串进行转换。
如果 atof 函数发生上溢等错误，就无法保证结果值。如果想得到发生错误时能保证的值，就必须使用 strtod 函数。

将字符串转换为 int 型

long atoi(const char *nptr)

说　明　　将表示 10 进制数的字符串转换为 int 型整数值。

头文件　　<stdlib.h>

返回值　　转换后的 int 型整数值

参　数　　nptr　　要转换的表示数的字符串指针

例

```
#include <stdlib.h>
const char *nptr;
int ret;
   ret=atoi(nptr);
```

错误条件　　如果转换后的值发生上溢，就设定 errno。

备　注　　对不符合 10 进制数格式的最初字符前的字符串进行转换。
如果 atoi 函数发生上溢等错误，就无法保证结果值。如果想得到发生错误时能保证的值，就必须使用 strtol 函数。

将字符串转换为 long 型

long atol(const char *nptr)

说　明　　将表示 10 进制数的字符串转换为 long 型整数值。

头文件　　<stdlib.h>

返回值　　转换后的 long 型整数值

参　数　　nptr　　要转换的表示数的字符串指针

例

```
#include <stdlib.h>
const char *nptr;
long ret;
   ret=atol(nptr);
```

错误条件　　如果转换后的值发生上溢，就设定 errno。

备　注　　对不符合 10 进制数格式的最初字符前的字符串进行转换。
如果 atol 函数发生上溢等错误，就无法保证结果值。如果想得到发生错误时能保证的值，就必须使用 strtol 函数。

将字符串转换为 long long 型

long long atoll(const char *nptr)

说　明　　将表示 10 进制数的字符串转换为 long long 型整数值。

头文件　　<stdlib.h>

返回值　　转换后的 long long 型整数值

参　数　　nptr　　要转换的表示数的字符串指针

例

```
#include <stdlib.h>
const char *nptr;
long long ret;
   ret=atoll(nptr);
```

错误条件　　如果转换后的值发生上溢，就设定 errno。

备　注　　对不符合 10 进制数格式的最初字符前的字符串进行转换。
如果 atoll 函数发生上溢等错误，就无法保证结果值。如果想得到发生错误时能保证的值，就必须使用 strtoll 函数。

将字符串转换为 double 型

double strtod(const char *nptr, char **endptr)

说　明　　将表示数的字符串转换为 double 型浮点值。

头文件　　<stdlib.h>

返回值　　正常：　当 nptr 指向的字符串以不构成浮点型的字符开始时：0
　　当 nptr 指向的字符串以构成浮点型的字符开始时：转换后的 double 型浮点值
　　异常：　当转换后的值发生上溢时：有和要转换的字符串相同符号的 HUGE_VAL
　　当转换后的值发生下溢时：0

参　数　　nptr　　指向要转换的表示数的字符串指针
　　endptr　　指向存储区的指针，该存储区保存最初不构成浮点值的字符指针。

例

```
#include <stdlib.h>
const char *nptr;
char **endptr;
double ret;
   ret=strtod(nptr,endptr);
```

错误条件　　如果转换后的值发生上溢或者下溢，就设定 errno。

备　注　　strtod 函数根据“9.1.3(4) 浮点运算的规格”的规则，将从最初的数字或者小数点到不构成浮点值的字符前的字符串转换为 double 型浮点值。如果既不出现指数也不出现小数点，就假设小数点在字符串的最后数字的后面。将指向最初不构成浮点型的字符指针设定到 endptr 指向的区域。如果在读数字前出现不构成浮点型的字符，就设定 nptr 的值，但是在 endptr 为 NULL 时，不进行此设定。

将字符串转换为 float 型

float strtof(const char *nptr, char **endptr)

说　明　　将表示数的字符串转换为 float 型浮点值。

头文件　　<stdlib.h>

返回值　　正常：　当 nptr 指向的字符串以不构成浮点型的字符开始时：0
　　当 nptr 指向的字符串以构成浮点型的字符开始时：转换后的 float 型浮点值
　　异常：　当转换后的值发生上溢时：有和要转换的字符串相同符号的 HUGE_VALF
　　当转换后的值发生下溢时：0

参　数　　nptr　　指向要转换的表示数的字符串指针
　　endptr　　指向存储区的指针，该存储区保存最初不构成浮点值的字符指针。

例　　
```
#include <stdlib.h>
const char *nptr;
char **endptr;
float ret;
   ret=strtof(nptr,endptr);
```

错误条件　　如果转换后的值发生上溢或者下溢，就设定 errno。

备　注　　strtof 函数根据“9.1.3(4) 浮点运算的规格”的规则，将从最初的数字或者小数点到不构成浮点值的字符前的字符串转换为 float 型浮点值。如果既不出现指数也不出现小数点，就假设小数点在字符串的最后数字的后面。将指向最初不构成浮点型的字符指针设定到 endptr 指向的区域。如果在读数字前出现不构成浮点型的字符，就设定 nptr 的值，但是在 endptr 为 NULL 时，不进行此设定。

将字符串转换为 long double 型

long double strtold(const char *nptr, char **endptr)

说　明　　将表示数的字符串转换为 long double 型浮点值。

头文件　　<stdlib.h>

返回值　　正常：　当 nptr 指向的字符串以不构成浮点型的字符开始时：0
当 nptr 指向的字符串以构成浮点型的字符开始时：转换后的 long double 型浮点值
异常：　当转换后的值发生上溢时：有和要转换的字符串相同符号的 HUGE_VALL
当转换后的值发生下溢时：0

参　数　　nptr　　指向要转换的表示数的字符串指针
endptr　　指向存储区的指针，该存储区保存最初不构成浮点值的字符指针。

例

```
#include <stdlib.h>
const char *nptr;
char **endptr;
long double ret;
   ret=strtold(nptr,endptr);
```

错误条件　　如果转换后的值发生上溢或者下溢，就设定 errno。

备　注　　strtold 函数根据“9.1.3(4) 浮点运算的规格”的规则，将从最初的数字或者小数点到不构成浮点值的字符前的字符串转换为 long double 型浮点值。如果既不出现指数也不出现小数点，就假设小数点在字符串的最后数字的后面。将指向最初不构成浮点型的字符指针设定到 endptr 指向的区域。如果在读数字前出现不构成浮点型的字符，就设定 nptr 的值，但是在 endptr 为 NULL 时，不进行此设定。

将字符串转换为 long 型

long strtol(const char *nptr, char **endptr, long base)

说　明　　将表示数的字符串转换为 long 型整数值。

头文件　　<stdlib.h>

返回值　　正常：　当 nptr 指向的字符串以不构成整数的字符开始时：0
　　当 nptr 指向的字符串以构成整数的字符开始时：转换后的 long 型整数值
　　异常：　当转换后的值发生上溢时：LONG_MAX 或者 LONG_MIN（取决于要转换的字符串符号）

参　数

nptr	指向要转换的表示数的字符串指针
endptr	指向存储器的指针，该存储器保存最初不构成整数的字符指针
base	转换基数（0 或者 2 ～ 36）

例

```
#include <stdlib.h>
long ret;
const char *nptr;
char **endptr;
int base;
   ret=strtol(nptr,endptr,base);
```

错误条件　　如果转换后的值发生上溢，就设定 errno。

备　注　　strtol 函数将从最初的数字到最初不构成整数的字符前的字符串转换为 long 型整数值。将指向最初不构成整数的字符指针设定到 endptr 指向的存储区。如果在读最初的数字前出现不构成整数的字符，就设定 nptr 的值，但是在 endptr 为 NULL 时，不进行此设定。
当 base 的值为 0 时，根据“9.1.1(4) 整数”的规则进行转换。当 base 的值在 2 ～ 36 之间时，表示转换时的基数。要转换的字符串中的 a（或者 A）～ z（或者 Z）的字符对应 10 ～ 35 的值。如果在要转换的字符串中存在大于等于 base 值的字符，就结束转换处理。在转换时，忽视符号后的 0，并且忽视 base 为 16 时的 0x（或者 0X）。

将字符串转换为 unsigned long 型

unsigned long strtoul(const char *nptr, char **endptr, long base)

说　明　　将表示数的字符串转换为 unsigned long 型整数值。

头文件　　<stdlib.h>

返回值　　正常：　当 nptr 指向的字符串以不构成整数的字符开始时：0
　　当 nptr 指向的字符串以构成整数的字符开始时：转换后的 unsigned long 型整数值
　　异常：　当转换后的值发生上溢时：ULONG_MAX

参　数

nptr	指向要转换的表示数的字符串指针
endptr	指向存储区的指针，该存储器保存最初不构成整数的字符指针
base	转换基数（0 或者 2 ～ 36）

例

```
#include <stdlib.h>
unsigned long ret;
const char *nptr;
char **endptr;
int base;
   ret=strtoul(nptr,endptr,base);
```

错误条件　　如果转换后的值发生上溢，就设定 errno。

备　注　　strtoul 函数将从最初的数字到最初不构成整数的字符前的字符串转换为 unsigned long 型整数值。
将指向最初不构成整数的字符指针设定到 endptr 指向的存储区。如果在读最初的数字前出现不构成整数的字符，就设定 nptr 的值，但是在 endptr 为 NULL 时，不进行此设定。
当 base 的值为 0 时，根据“9.1.1(4) 整数”的规则进行转换。当 base 的值在 2 ～ 36 之间时，表示转换时的基数。要转换的字符串中的 a（或者 A）～ z（或者 Z）的字符对应 10 ～ 35 的值。如果在要转换的字符串中存在大于等于 base 值的字符，就结束转换处理。在转换时，忽视符号后的 0，并且忽视 base 为 16 时的 0x（或者 0X）。

将字符串转换为 long long 型

long long strtoll(const char *nptr, char **endptr, long base)

说　明　　将表示数的字符串转换为 long long 型整数值。

头文件　　<stdlib.h>

返回值　　正常：　当 nptr 指向的字符串以不构成整数的字符开始时：0
　　当 nptr 指向的字符串以构成整数的字符开始时：转换后的 long long 型整数值
　　异常：　当转换后的值发生上溢时：LLONG_MAX 或者 LLONG_MIN（取决于要转换的字符串符号）

参　数

nptr	指向要转换的表示数的字符串指针
endptr	指向存储区的指针，该存储区保存最初不构成整数的字符指针
base	转换基数（0 或者 2 ～ 36）

例

```
#include <stdlib.h>
long long ret;
const char *nptr;
char **endptr;
int base;
   ret=strtoll(nptr,endptr,base);
```

错误条件　　如果转换后的值发生上溢，就设定 errno。

备　注　　strtoll 函数将从最初的数字到最初不构成整数的字符前的字符串转换为 long long 型整数值。
将指向最初不构成整数的字符指针设定到 endptr 指向的存储区。如果在读最初的数字前出现不构成整数的字符，就设定 nptr 的值，但是在 endptr 为 NULL 时，不进行此设定。
当 base 的值为 0 时，根据“9.1.1(4) 整数”的规则进行转换。当 base 的值在 2 ～ 36 之间时，表示转换时的基数。要转换的字符串中的 a（或者 A）～ z（或者 Z）的字符对应 10 ～ 35 的值。如果在要转换的字符串中存在大于等于 base 值的字符，就结束转换处理。在转换时，忽视符号后的 0，并且忽视 base 为 16 时的 0x（或者 0X）。

将字符串转换为 unsigned long long 型

unsigned long long strtoull(const char *nptr, char **endptr, long base)

说　明　　将表示数的字符串转换为 unsigned long long 型整数值。

头文件　　<stdlib.h>

返回值　　正常：　当 nptr 指向的字符串以不构成整数的字符开始时：0
　　当 nptr 指向的字符串以构成整数的字符开始时：转换后的 unsigned long long 型整数值
　　异常：　当转换后的值发生上溢时：ULLONG_MAX

参　数

nptr	指向要转换的表示数的字符串指针
endptr	指向存储区的指针，该存储区保存最初不构成整数的字符指针
base	转换基数（0 或者 2 ～ 36）

例

```
#include <stdlib.h>
unsigned long long ret;
const char *nptr;
char **endptr;
int base;
   ret=strtoull(nptr,endptr,base);
```

错误条件　　如果转换后的值发生上溢，就设定 errno。

备　注　　strtoull 函数将从最初的数字到最初不构成整数的字符前的字符串转换为 unsigned long long 型整数值。
将指向最初不构成整数的字符指针设定到 endptr 指向的存储区。如果在读最初的数字前出现不构成整数的字符，就设定 nptr 的值，但是在 endptr 为 NULL 时，不进行此设定。
当 base 的值为 0 时，根据“9.1.1(4) 整数”的规则进行转换。当 base 的值在 2 ～ 36 之间时，表示转换时的基数。要转换的字符串中的 a（或者 A）～ z（或者 Z）的字符对应 10 ～ 35 的值。如果在要转换的字符串中存在大于等于 base 值的字符，就结束转换处理。在转换时，忽视符号后的 0，并且忽视 base 为 16 时的 0x（或者 0X）。

伪随机数的生成

long rand(void)

说　明　　生成 0 ～ RAND_MAX 范围的伪随机数的整数。

头文件　　<stdlib.h>

返回值　　伪随机数的整数值

例
```
#include <stdlib.h>
int ret;
   ret=rand();
```

伪随机数串的初始设定

void srand(unsigned long seed)

说　明　　设定 rand 函数生成的伪随机数串的初始值。

头文件　　<stdlib.h>

参　数　　seed　　伪随机数串生成的初始值

例
```
#include <stdlib.h>
unsigned int seed;
   srand(seed);
```

备　注　　为了 rand 函数生成伪随机数串， srand 函数设定初始值。因此，当 rand 函数正在生成伪随机数值时，如果再次使用 srand 函数设定相同的初始值，就重复生成伪随机数串。
如果在调用 srand 函数前调用 rand 函数，就将生成伪随机数串的初始值设定为 1。

带初始化的存储区分配

void *calloc(size_t nelem, size_t elsize)

说　明　　分配存储区，将已分配的全部存储区初始化为 0。

头文件　　<stdlib.h>

返回值　　正常：已分配存储区的起始地址
异常：当不能分配存储区或者某个参数为 0 时：NULL

参　数　　nelem　　元素的个数
elsize　　一个元素占用的字节数

例

```
#include <stdlib.h>
size_t nelem, elsize;
void *ret;
   ret=calloc(nelem,elsize);
```

备　注　　将以 elsize 字节为单位的存储区分配为 nelem 个存储区，并且将已分配存储区的全部位初始化为 0。

存储区的释放

void free(void *ptr)

说　明　　释放所指定的存储区。

头文件　　<stdlib.h>

参　数　　ptr　　要释放的存储区地址

例

```
#include <stdlib.h>
void *ptr;
   free(ptr);
```

备　注　　释放 ptr 指向的存储区，能重新分配并且使用被释放的存储区。如果 ptr 为 NULL，就不进行任何操作。
对于不是由 calloc、 malloc、 realloc 函数分配的存储区或者已由 free、 realloc 函数释放的存储区，无法保证运行。无法保证参照释放后的存储区时的运行。

存储区的分配

void *malloc(size_t size)

说　明　　分配存储区。

头文件　　<stdlib.h>

返回值　　正常：已分配存储区的起始地址
异常：当不能分配存储区或者 size 为 0 时：NULL

参　数　　size　　要分配的存储区的字节数

例

```
#include <stdlib.h>
size_t size;
void *ret;
   ret=malloc(size);
```

备　注　　只分配 size 表示的字节长度的存储区。

存储区分配容量的变更

void *realloc(void *ptr, size_t size)

说　明　　将存储区容量更改为指定的容量。

头文件　　<stdlib.h>

返回值　　正常：更改后的存储区的起始地址
异常：当不能分配存储区或者 size 为 0 时：NULL

参　数　　ptr　　要更改的存储区的起始地址
size　　更改后的存储区的字节数

例

```
#include <stdlib.h>
size_t size;
void *ptr, *ret;
   ret=realloc(ptr,size);
```

备　注　　将 ptr 指向的存储区容量更改为 size 表示的字节长度的存储区。如果新分配的存储区容量小于更改前的存储区容量，新分配的存储区容量为止的内容就不变。
在 ptr 不是指向 calloc、 malloc、 realloc 函数分配的存储区的指针或者 ptr 是指向已被 free、 realloc 函数释放的存储区的指针时，不进行任何操作。

二分检索

void *bsearch(const void *key, const void *base, size_t nmemb, size_t size,
int (*compar)(const void *, const void *))

说　明　　进行二分检索。

头文件　　<stdlib.h>

返回值　　当能检测到相同的成员时　　：指向相同成员的指针
当无法检测到相同的成员时　　：NULL

参　数

key	指向要检索的数据指针
base	指向检索对象表的指针
nmemb	检索对象的成员个数
size	检索对象的成员字节数
compar	指向要比较的函数指针

例

```
#include <stdlib.h>
const void *key, *base;
size_t nmemb, size;
int (*compar)(const void *, const void *);
void *ret;
   ret=bsearch(key,base,nmemb,size,compar);
```

备　注　　根据二分检索法，在 base 指向的表中检索和 key 指向的数据相同的成员。要进行比较的函数必须取得指向 2 个比较数据的指针 p1（第 1 个参数）和 p2（第 2 个参数），并且根据以下规格返回结果：

当 *p1<*p2 时，返回负值。
当 *p1==*p2 时，返回 0。
当 *p1>*p2 时，返回正值。

检索对象的各成员需要按升序进行排列。

排序

void qsort(const void *base, size_t nmemb, size_t size, int (*compar)(const void *, const void *))

说　明　　进行排序。

头文件　　<stdlib.h>

参　数

base	指向排序对象表的指针
nmemb	排序对象的成员个数
size	排序对象的成员字节数
compar	指向要比较的函数指针

例

```
#include <stdlib.h>
const void *base;
size_t nmemb, size;
int (*compar)(const void *, const void *);
   qsort(base, nmemb, size, compar);
```

备　注　　对 base 指向的表数据进行排序。用指向比较函数的指针指定数据的排列顺序。此函数必须取得 2 个比较数据的指针 p1（第 1 个参数）和 p2（第 2 个参数），并且根据以下规格返回结果：

当 *p1<*p2 时，返回负值。
当 *p1==*p2 时，返回 0。
当 *p1>*p2 时，返回正值。

绝对值

long abs(long i)

说　明　　计算 int 型整数的绝对值。

头文件　　<stdlib.h>

返回值　　i 的绝对值

参　数

i	要计算绝对值的整数

例

```
#include <stdlib.h>
int i, ret;
   ret=abs(i);
```

备　注　　当 i 的绝对值结果不能表示为 int 型整数值时，无法保证运行。

商和余数

div_t div(long numer, long denom)

说　明　　计算 int 型整数除法运算的商和余数。

头文件　　<stdlib.h>

返回值　　numer 除 denom 的商和余数

参　数　　numer　　被除数
　　denom　　除数

例

```
#include <stdlib.h>
int numer, denom;
div_t ret;
   ret=div(numer,denom);
```

绝对值

long labs(long j)

说　明　　计算 long 型整数的绝对值。

头文件　　<stdlib.h>

返回值　　j 的绝对值

参　数　　j　　要计算绝对值的整数

例

```
#include <stdlib.h>
long j;
long ret;
   ret=labs(j);
```

备　注　　当 j 的绝对值结果不能表示为 long 型整数值时，无法保证运行。

商和余数

ldiv_t ldiv(long numer, long denom)

说　明　　计算 long 型整数除法运算的商和余数。

头文件　　<stdlib.h>

返回值　　numer 除 denom 的商和余数

参　数　　numer　　被除数
　　denom　　除数

例

```
#include <stdlib.h>
long numer, denom;
ldiv_t ret;
   ret=ldiv(numer,denom);
```

绝对值

long long llabs(long long j)

说　明　　计算 long long 型整数的绝对值。

头文件　　<stdlib.h>

返回值　　j 的绝对值

参　数　　j　　要计算绝对值的整数

例

```
#include <stdlib.h>
long long j;
long long ret;
   ret=llabs(j);
```

备　注　　当 j 的绝对值结果不能表示为 long long 型整数值时，无法保证运行。

商和余数

lldiv_t lldiv(long long numer,long long denom)

说　明　　计算 long long 型整数除法运算的商和余数。

头文件　　<stdlib.h>

返回值　　numer 除 denom 的商和余数

参　数　　numer　　被除数
　　denom　　除数

例

```
#include <stdlib.h>
long long numer, denom;
lldiv_t ret;
   ret=lldiv(numer,denom);
```

(14) <string.h>

定义字符数组的操作所需的各种函数。

种类	定义名	说明
函数	memcpy	将指定容量的源存储区的内容复制到目标存储区。
	strcpy	将包含空字符的源字符串的内容复制到目标存储区。
	strncpy	将指定字符数的源字符串复制到目标存储区。
	strcat	将字符串连接在字符串的后面。
	strncat	将指定字符数的字符串连接到字符串的后面。
	memcmp	比较指定的 2 个存储区。
	strcmp	比较指定的 2 个字符串。
	strncmp	对指定的 2 个字符串进行指定字符数的比较。
	memchr	在指定的存储区中检索指定的字符最初出现的位置。
	strchr	在指定的字符串中检索指定的字符最初出现的位置。
	strcspn	从头开始调查指定的字符串，取得另外指定的字符串中的字符最初出现前的字符数。
	strpbrk	在指定的字符串中检索另外指定的字符串中的字符最初出现的位置。
	strrchr	在指定的字符串中检索指定的字符最后出现的位置。
	strspn	从头开始调查指定的字符串，取得另外指定的字符串中的字符从头连续出现的字符数。
	strstr	在指定的字符串中检索另外指定的字符串最初出现的位置。
	strtok	将指定的字符串分成若干个字句。
	memset	从指定的存储区的开头设定指定字符数的指定字符。
	strerror	设定错误信息。
	strlen	计算字符串的字符数。
	memmove	将指定容量的源存储区的内容复制到目标存储区。即使源存储区和目标存储区重叠，也能正常复制。

实现定义的规格

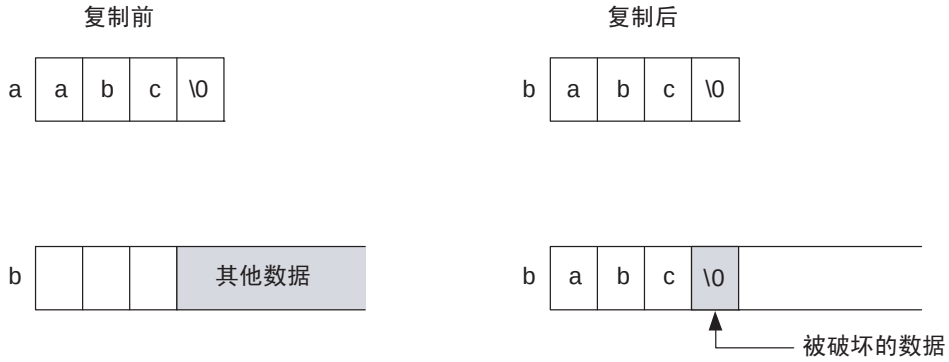
	项目	编译程序的规格
1	strerror 函数返回的错误信息内容	请参照 “11.3 C 标准库函数的错误信息”。

在使用本标准 inlcude 文件中定义的函数时，需要注意以下 2 个事项：

1. 必须注意：在复制字符串时，如果复制目标区域小于复制源区域，就无法保证运行。
   例：

```
char a[]="abc";
char b[3];
          :
          :
strcpy(b,a);
```

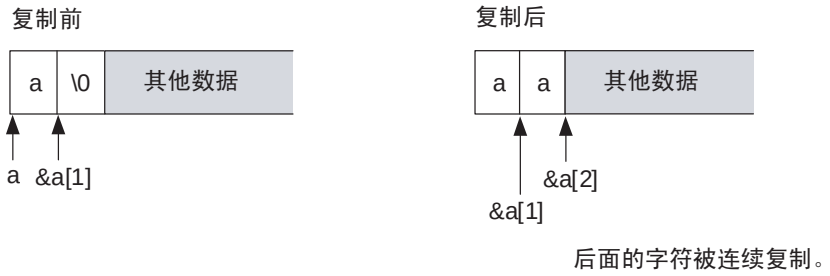
   此时，数组a的长度 （包括空字符）为4字节。如果通过strcpy函数进行复制，就改写数组b以外区域的数据。



2. 必须注意：在复制字符串时，如果复制源区域和复制目标区域重叠，就无法保证运行。
   例：

```
int a[]="a";
          :
          :
strcpy(&a[1], a);
          :
          :
```

   此时，在读到复制源字符串中的空字符前，字符'a'覆盖空字符。因此，复制源字符串数据后面的数据被改写。



存储区的复制

void *memcpy(void *s1, const void *s2, size_t n)

说　明　　将指定容量的源存储区内容复制到目标存储区。

头文件　　<string.h>

返回值　　s1 的值

参　数

s1	指向复制目标存储区的指针
s2	指向复制源存储区的指针
n	要复制的字符数

例

```
#include <string.h>
void *ret, *s1;
const void *s2;
size_t n;
   ret=memcpy(s1,s2,n);
```

字符串的复制

char *strcpy(char *s1, const char *s2)

说　明　　将包含空字符的源字符串内容复制到目标存储区。

头文件　　<string.h>

返回值　　s1 的值

参　数

s1	指向复制目标存储区的指针
s2	指向复制源字符串的指针

例

```
#include <string.h>
char *s1, *ret;
const char *s2;
   ret=strcpy(s1,s2);
```

字符串的复制

char *strncpy(char *s1, const char *s2, size_t n)

说　明　　将指定字符数的源字符串复制到目标存储区。

头文件　　<string.h>

返回值　　s1 的值

参　数

s1	指向复制目标存储区的指针
s2	指向复制源字符串的指针
n	要复制的字符数

例

```
#include <string.h>
char *s1, *ret;
const char *s2;
size_t n;
   ret=strncpy(s1,s2,n);
```

备　注　　将 s2 指定的字符串的前 n 个字符（最多）复制到 s1 指定的存储区。如果 s2 指定的字符串的字符数小于 n 个字符时，就附加空字符，使其增加到 n 个字符；如果 s2 指定的字符串的字符数大于 n 个字符时，复制到 s1 中的字符串就不以空字符结束。

字符串的连接

char *strcat(char *s1, const char *s2)

说　明　　将字符串连接到字符串的后面。

头文件　　<string.h>

返回值　　s1 的值

参　数

s1	指向被连接的字符串的指针
s2	指向要连接的字符串的指针

例

```
#include <string.h>
char *s1, *ret;
const char *s2;
   ret=strcat(s1,s2);
```

备　注　　将 s2 指定的字符串连接到 s1 指定的字符串的末尾。此时，也复制 s2 指定的字符串末尾的空字符，并且用 s2 的起始字符替换 s1 指定的字符串末尾的空字符。

字符串的连接

char *strncat(char *s1, const char *s2, size_t n)

说　明　　将指定字符数的字符串连接到字符串。

头文件　　<string.h>

返回值　　s1 的值

参　数

s1	指向被连接的字符串的指针
s2	指向要连接的字符串的指针
n	要连接的字符数

例

```
#include <string.h>
char *s1, *ret;
const char *s2;
size_t n;
   ret=strncat(s1,s2,n);
```

备　注　　将 s2 指定的字符串的前 n 个字符 （最多）附加到 s1 指定的字符串的末尾。用 s2 的起始字符替换 s1 指定的字符串末尾的空字符。
在连接后的字符串的末尾一定附加空字符。

存储区的比较

long memcmp(const void *s1, const void *s2, size_t n)

说　明　　比较指定的 2 个存储区的内容。

头文件　　<string.h>

返回值

当 s1 指定的存储区	>	s2 指定的存储区时：正值
当 s1 指定的存储区	==	s2 指定的存储区时：0
当 s1 指定的存储区	<	s2 指定的存储区时：负值

参　数

s1	指向被比较的存储区指针
s2	指向要比较的存储区指针
n	要比较的存储区的字符数

例

```
#include <string.h>
const void *s1, *s2;
size_t n;
int ret;
   ret=memcmp(s1,s2,n);
```

备　注　　比较 s1 指定的存储区和 s2 指定的存储区的前 n 个字符的内容。
此比较为实现定义。

字符串的比较

long strcmp(const char *s1, const char *s2)

说　明　　比较指定的 2 个字符串的内容。

头文件　　<string.h>

返回值　　当 s1 指定的字符串　>　　s2 指定的字符串时：正值
当 s1 指定的字符串　==　　s2 指定的字符串时：0
当 s1 指定的字符串　<　　s2 指定的字符串时：负值

参　数　　s1　　指向被比较的字符串的指针
s2　　指向要比较的字符串的指针

例

```
#include <string.h>
const char *s1, *s2;
int ret;
   ret=strcmp(s1,s2);
```

备　注　　比较 s1 指定的字符串和 s2 指定的字符串内容，将结果设定为返回值。
此比较为实现定义。

字符串的比较

long strncmp(const char *s1, const char *s2, size_t n)

说　明　　对指定的 2 个字符串进行指定字符数的比较。

头文件　　<string.h>

返回值　　当 s1 指定的字符串　>　　s2 指定的字符串时：正值
当 s1 指定的字符串　==　　s2 指定的字符串时：0
当 s1 指定的字符串　<　　s2 指定的字符串时：负值

参　数　　s1　　指向被比较的字符串的指针
s2　　指向要比较的字符串的指针
n　　要比较的字符数的最大值

例

```
#include <string.h>
const char *s1, *s2;
size_t n;
int ret;
   ret=strncmp(s1,s2,n);
```

备　注　　比较 s1 指定的字符串和 s2 指定的字符串的前 n 个字符的内容。
此比较为实现定义。

存储区内的字符检索

void *memchr(const void *s, long c, size_t n)

说　明　在指定的存储区中检索指定的字符最初出现的位置。

头文件　<string.h>

返回值　当找到要检索的字符时　：指向找到的字符指针
当未找到要检索的字符时 ：NULL

参　数

s	指向被检索的存储区的指针
c	要检索的字符
n	要检索的字符数

例

```
#include <string.h>
const void *s;
int c;
size_t n;
void *ret;
   ret=memchr(s,c,n);
```

备　注　在 s 指定的存储区的前 n 个字符中，将指向最初出现的和 c 字符相同字符的位置指针作为返回值返回。

最初字符的位置

char *strchr(const char *s, long c)

说　明　在指定的字符串中检索指定的字符最初出现的位置。

头文件　<string.h>

返回值　当找到要检索的字符时　：指向找到的字符指针
当未找到要检索的字符时 ：NULL

参　数

s	指向被检索的字符串的指针
c	要检索的字符

例

```
#include <string.h>
const char *s;
int c;
char *ret;
   ret=strchr(s,c);
```

备　注　在 s 指定的字符串中，将指向最初出现的和 c 字符相同字符的指针作为返回值返回。
检索对象也包括表示 s 指定的字符串结束的空字符。

指定字符集最初出现前的字符数

size_t strcspn(const char *s1, const char *s2)

说　明　　从头开始调查指定的字符串，取得另外指定的字符串中的字符最初出现前的字符数。

头文件　　<string.h>

返回值　　从 s1 的开头开始， s2 指定的字符串中的字符最初出现前的字符数。

参　数　　s1　　指向被调查的字符串的指针
　　s2　　指向用于调查 s1 的字符串的指针

例

```
#include <string.h>
const char *s1, *s2;
size_t ret;
   ret=strcspn(s1,s2);
```

备　注　　从头开始调查 s1 指定的字符串，取得 s2 指定的字符串中的字符最初出现前的字符数，将该字符串的字符数作为返回值返回。
不将表示 s2 指定的字符串结束的空字符视为 s2 指定的字符串的一部分。

指定字符集最初出现的位置

char *strpbrk(const char *s1, const char *s2)

说　明　　在指定的字符串中检索另外指定的字符串中的字符最初出现的位置。

头文件　　<string.h>

返回值　　当找到要检索的字符时　：指向找到的字符指针
　　当未找到要检索的字符时 ：NULL

参　数　　s1　　指向被检索的字符串的指针
　　s2　　指向表示在 s1 内要检索字符的字符串的指针

例

```
#include <string.h>
const char *s1, *s2;
char *ret;
   ret=strpbrk(s1,s2);
```

备　注　　在 s1 指定的字符串中，检索 s2 指定的字符串中某个字符最初出现的位置，将指向该位置的指针作为返回值返回。

最后的字符位置

char *strrchr(const char *s, long c)

说　明　在指定的字符串中检索指定的字符最后出现的位置。

头文件　<string.h>

返回值　当找到要检索的字符时　：指向找到的字符指针
当未找到要检索的字符时 ：NULL

参　数

s	指向被检索的字符串的指针
c	要检索的字符

例

```
#include <string.h>
const char *s;
int c;
char *ret;
   ret=strrchr(s,c);
```

备　注　在 s 指定的字符串中，将指向 c 指定的字符最后出现的位置指针作为返回值返回。
检索对象也包括表示 s 指定的字符串结束的空字符。

指定字符集连续部分的长度

size_t strspn(const char *s1, const char *s2)

说　明　从头开始调查指定的字符串，取得另外指定的字符串中的字符从头连续出现的字符数。

头文件　<string.h>

返回值　s2 指定的字符串中的字符从 s1 的开头连续出现的字符数

参　数

s1	指向被调查的字符串的指针
s2	指向用于调查 s1 的字符串的指针

例

```
#include <string.h>
const char *s1, *s2;
size_t ret;
   ret=strspn(s1,s2);
```

备　注　从头开始调查 s1 指定的字符串，取得 s2 指定的字符串中的字符连续出现的字符数，将该字符串的字符数作为返回值返回。

最初的字符串位置

char *strstr(const char *s1, const char *s2)

说　明　　在指定的字符串中检索另外指定的字符串最初出现的位置。

头文件　　<string.h>

返回值　　当找到要检索的字符时　：指向找到的字符指针
当未找到要检索的字符时 ：NULL

参　数

s1	指向被检索的字符串的指针
s2	指向要检索的字符串的指针

例

```
#include <string.h>
const char *s1, *s2;
char *ret;
   ret=strstr(s1,s2);
```

备　注　　在 s1 指定的字符串中检索 s2 指定的字符串最初出现的位置，将指向该位置的指针作为返回值返回。

字句的拆分

char *strtok(char *s1, const char *s2)

说　明　　将指定的字符串拆成若干个字句。

头文件　　<string.h>

返回值　　当拆成字句时　　：指向已拆成字句的起始位置的指针
当未拆成字句时　　：NULL

参　数　　s1　　指向被拆成若干个字句的字符串的指针
s2　　指向用于拆分字符串的字符串指针

例

```
#include <string.h>
char *s1, *ret;
const char *s2;
    ret=strtok(s1,s2);
```

备　注　　为了拆分字符串而连续调用 strtok 函数。
(a) 最初调用时
用 s2 指定的字符串中的字符，将 s1 指定的字符串从头开始拆成字句。如果拆成字句，就将指向该字句开头的指针作为返回值返回，否则返回 NULL。
(b) 第 2 次以后的调用时
用 s2 指定的字符串中的字符，从以前被拆分的字句的下一个字符开始，将字符串拆成字句。如果拆成字句，就将指向该字句开头的指针作为返回值返回，否则返回 NULL。
在第 2 次以后的调用时，给第 1 个参数指定 NULL，每次调用时的 s2 指定的字符串可以不同。在被拆成字句的末尾附加空字符。
strtok 函数的使用例子如下所示：

例：

```
1   #include <string.h>
2   static char s1[]="a@b,@c/@d";
3   char *ret;
4
5   ret=strtok(s1,"@");
6   ret=strtok(NULL,",@");
7   ret=strtok(NULL,"/@");
8   ret=strtok(NULL,"@");
```

【说明】
此例是通过 strtok 函数将字符串 “a@b,@c/@d” 拆成 a、 b、 c、 d 字句的程序。
在第 2 行，给字符串 s1 设定字符串 “a@b,@c/@d” 的初始值。
在第 5 行，为了将 “@” 作为分割符拆分字句而调用 strtok 函数。结果得到指向字符 'a' 的指针的返回值，并且对字符 'a' 的下一个最初的分割符 “@” 埋入空字符。结果字符串 "a" 被分开。
然后，为了将同一个字符串依次拆成字句，给第 1 个参数指定 NULL，调用 strtok 函数。结果字符串依次被拆成 "b"、 "c"、 "d"。

字符的重复

void *memset(void *s, long c, size_t n)

说　明　　从指定的存储区的开头设定指定字符数的指定字符。

头文件　　<string.h>

返回值　　s 的值

参　数

s	指向被设定字符的存储区的指针
c	要设定的字符
n	要设定的字符数

例

```
#include <string.h>
void *s, *ret;
int c;
size_t n;
   ret=memset(s,c,n);
```

备　注　　给 s 指定的存储区设定 n 个字符 c。

错误信息字符串

char *strerror(long s)

说　明　　指定错误号，返回对应的错误信息。

头文件　　<string.h>

返回值　　指向错误号对应的错误信息（字符串）的指针

参　数

s	错误号

例

```
#include <string.h>
char *ret;
int s;
   ret=strerror(s);
```

备　注　　将指向错误号 s 对应的错误信息的指针作为返回值返回。
有关错误信息的内容为实现定义。
如果修正作为返回值返回的错误信息，就无法保证运行。

字符串的字符数

size_t strlen(const char *s)

说　明　　计算字符串的字符数。

头文件　　<string.h>

返回值　　字符串的字符数

参　数　　s　　指向要计算长度的字符串的指针

例

```
#include <string.h>
const char *s;
size_t ret;
   ret=strlen(s);
```

备　注　　表示 s 指定的字符串结束的空字符不在字符串字符数的计算范围内。

存储区的移动

void *memmove(void *s1, const void *s2, size_t n)

说　明　　将指定容量的源存储区的内容复制到目标存储区。
即使源存储区和目标存储区有重叠部分，因为在重写源存储区的重叠部分前先进行复制，所以也能正常复制。

头文件　　<string.h>

返回值　　s1 的值

参　数　　s1　　指向复制目标存储区的指针
s2　　指向复制源存储区的指针
n　　要复制的字符数

例

```
#include <string.h>
void *ret, *s1;
const void *s2;
size_t n;
   ret=memmove(s1,s2,n);
```

(15) <complex.h>

计算各种复数。在 float 型复数的情况下，定义名的最后为 ‘f’；在 long double 型复数的情况下，定义名的最后为 ‘l’；在 double 型复数的情况下，定义名为函数名。

种类	定义名	说明
函数	cacos	计算复数的反余弦。
	casin	计算复数的反正弦。
	catan	计算复数的反正切。
	ccos	计算复数的余弦。
	csin	计算复数的正弦。
	ctan	计算复数的正切。
	cacosh	计算复数的反双曲余弦。
	casinh	计算复数的反双曲正弦。
	catanh	计算复数的反双曲正切。
	ccosh	计算复数的双曲余弦。
	csinh	计算复数的双曲正弦。
	ctanh	计算复数的双曲正切。
	cexp	计算复数以 e 为底的 z 乘方的自然对数。
	clog	计算复数的自然对数。
	cabs	计算复数的绝对值。
	cpow	计算复数的乘方。
	csqrt	计算复数的平方根。
	carg	计算辐角。
	cimag	计算虚部。
	conj	将虚数的符号取反，计算复共轭。
	cproj	计算黎曼球面上的投影。
	creal	计算实部。

复数的反余弦

float complex cacosf(float complex z)
double complex cacos(double complex z)
long double complex cacosl(long double complex z)

说　明　　计算复数的反余弦。

头文件　　<complex.h>

返回值　　正常：z 的反余弦值
异常：在发生定义域错误时，返回非数值。

参　数　　z　　要计算复数反余弦的复数

例
```
#include <complex.h>
double complex z, ret;
   ret=cacos(z);
```

错误条件　　当 z 的值超出 [-1.0,1.0] 的范围时，发生定义域错误。

备　注　　cacos 函数返回值的实数轴方向的范围为 [0,π]，虚轴方向的范围为无限区间。

复数的反正弦

float complex casinf(float complex z)
double complex casin(double complex z)
long double complex casinl(long double complex z)

说　明　　计算复数的反正弦。

头文件　　<complex.h>

返回值　　正常：z 的复数反正弦值
异常：在发生定义域错误时，返回非数值。

参　数　　z　　要计算复数反正弦的复数

例
```
#include <complex.h>
double complex z, ret;
   ret=casin(z);
```

错误条件　　当 z 的值超出 [-1.0,1.0] 的范围时，发生定义域错误。

备　注　　casin 函数返回值的实数轴方向的范围为 [-π/2,π/2]，虚轴方向的范围为无限区间。

复数的反正切

float complex catanf(float complex z)
double complex catan(double complex z)
long double complex catanl(long double complex z)

说　明　　计算复数的反正切。

头文件　　<complex.h>

返回值　　正常：z 的复数反正切值

参　数　　z　　要计算复数反正切的复数

例

```
#include <complex.h>
double complex z, ret;
   ret=catan(z);
```

备　注　　catan 函数返回值的实数轴方向的范围为 [-π/2,π/2]，虚轴方向的范围为无限区间。

复数的余弦

float complex ccosf(float complex z)
double complex ccos(double complex z)
long double complex ccosl(long double complex z)

说　明　　计算复数的余弦。

头文件　　<complex.h>

返回值　　z 的复数余弦值

参　数　　z　　要计算复数余弦的复数

例

```
#include <complex.h>
double complex z, ret;
   ret=ccos(z);
```

复数的正弦

float complex csinf(float complex z)
double complex csin(double complex z)
long double complex csinl(long double complex z)

说　明　　计算复数的正弦。

头文件　　<complex.h>

返回值　　z 的复数正弦值

参　数　　z　　要计算复数正弦的复数

例

```
#include <complex.h>
double complex z, ret;
   ret=csin(z);
```

复数的正切

float complex ctanf(float complex z)
double complex ctan(double complex z)
long double complex ctanl(long double complex z)

说　明　　计算复数的正切。

头文件　　<complex.h>

返回值　　z 的复数正切值

参　数　　z　　要计算复数正切的复数

例

```
#include <complex.h>
double complex z, ret;
   ret=ctan(z);
```

复数的反双曲余弦

float complex cacoshf(float complex z)
double complex cacosh(double complex z)
long double complex cacoshl(long double complex z)

说　明　　计算复数的反双曲余弦。

头文件　　<complex.h>

返回值　　正常：z 的复数反双曲余弦值
异常：当发生定义域错误时，返回非数值。

参　数　　z　　要计算复数反双曲余弦的复数

例
```
#include <complex.h>
double complex z, ret;
   ret=cacosh(z);
```

错误条件　　当 z 的值超出 [-1.0, 1.0] 的范围时，发生定义域错误。

备　注　　cacoshf 函数群返回值的范围为 [0,π]。

复数的反双曲正弦

float complex casinhf(float complex z)
double complex casinh(double complex z)
long double complex casinhl(long double complex z)

说　明　　计算复数的反双曲正弦。

头文件　　<complex.h>

返回值　　z 的复数反双曲正弦值

参　数　　z　　要计算复数反双曲正弦的复数

例
```
#include <complex.h>
double complex z, ret;
   ret=casinh(z);
```

复数的反双曲正切

float complex catanhf(float complex z)
double complex catanh(double complex z)
long double complex catanhl(long double complex z)

说　明　　计算复数的反双曲正切。

头文件　　<complex.h>

返回值　　z 的复数反双曲正切值

参　数　　z　　要计算复数反双曲正切的复数

例

```
#include <complex.h>
double complex z, ret;
   ret=catanh(z);
```

复数的双曲余弦

float complex ccoshf(float complex z)
double complex ccosh(double complex z)
long double complex ccoshl(long double complex z)

说　明　　计算复数的双曲余弦。

头文件　　<complex.h>

返回值　　z 的复数双曲余弦值

参　数　　z　　要计算双曲余弦的复数

例

```
#include <complex.h>
double complex z, ret;
   ret=ccosh(z);
```

复数的双曲正弦

float complex csinhf(float complex z)
double complex csinh(double complex z)
long double complex csinhl(long double complex z)

说　明　　计算复数的双曲正弦。

头文件　　<complex.h>

返回值　　z 的复数双曲正弦值

参　数　　z　　要计算双曲正弦的复数

例

```
#include <complex.h>
double complex z, ret;
   ret=csinh(z);
```

复数的双曲正切

float complex ctanhf(float complex z)
double complex ctanh(double complex z)
long double complex ctanhl(long double complex z)

说　明　　计算复数的双曲正切。

头文件　　<complex.h>

返回值　　z 的复数双曲正切值

参　数　　z　　要计算双曲正切的复数

例

```
#include <complex.h>
double complex z, ret;
   ret=ctanh(z);
```

复数的指数函数

float complex cexpf(float complex z)
double complex cexp(double complex z)
long double complex cexpl(long double complex z)

说　明　　计算复数的指数函数。

头文件　　<complex.h>

返回值　　z 的指数函数值

参　数　　z　　要计算指数函数的复数

例

```
#include <complex.h>
double complex z, ret;
   ret=cexp(z);
```

复数的自然对数

float complex clogf(float complex z)
double complex clog(double complex z)
long double complex clogl(long double complex z)

说　明　　计算复数的自然对数。

头文件　　<complex.h>

返回值　　正常：z 的复数自然对数值
异常：当发生定义域错误时，返回非数值。

参　数　　z　　要计算复数自然对数的复数

例

```
#include <complex.h>
double complex z, ret;
   ret=clog(z);
```

错误条件　　当 z 的值为负时，发生定义域错误。
当 z 的值为 0.0 时，发生范围错误。

备　注　　clog 函数群返回值的实数轴方向的范围为无限区间，虚轴方向的范围为 [-iπ,+iπ]。

复数的绝对值

float cabsf(float complex z)
double cabs(double complex z)
long double cabsl(long double complex z)

说　明　　计算复数的绝对值。

头文件　　<complex.h>

返回值　　z 的复数绝对值

参　数　　z　　要计算复数绝对值的复数

例
```
#include <complex.h>
double complex z, ret;
   ret=cabs(z);
```

复数的乘方

float complex cpowf(float complex x, float complex y)
double complex cpow(double complex x, double complex y)
long double complex cpowl(long double complex x, long double complex y)

说　明　　计算复数的乘方。

头文件　　<complex.h>

返回值　　正常：x 的 y 乘方的值
异常：当发生定义域错误时，返回非数值。

参　数　　x　　被乘方的值
y　　乘方的值

例
```
#include <complex.h>
double complex x, y;
   ret=cpow(x, y);
```

错误条件　　当 x 的值为 0.0 并且 y 的值小于等于 0.0 或者 x 的值为负值并且 y 的值不是整数值时，发生定义域错误。

备　注　　cpow 函数群的第 1 个伪参数的分支切断线沿着负的实数轴。

复数的平方根

float complex csqrtf(float complex z)
double complex csqrt(double complex z)
long double complex csqrtl(long double complex z)

说　明　　计算复数的平方根。

头文件　　<complex.h>

返回值　　正常：z 的复数平方根值
异常：当发生定义域错误时，返回非数值。

参　数　　z　　要计算平方根函数值的复数

例
```
#include <complex.h>
double complex z, ret;
   ret=csqrt(z);
```

错误条件　　当 z 的值为负值时，发生定义域错误。

备　注　　csqrt 函数群的分支切断线沿着负的实数轴。
csqrt 函数群返回值的区域是包含虚轴的右半平面。

辐角

float cargf(float complex z)
double carg(double complex z)
long double cargl(long double complex z)

说　明　　计算辐角。

头文件　　<complex.h>

返回值　　z 的辐角值

参　数　　z　　要计算辐角值的复数

例
```
#include <complex.h>
double complex z, ret;
   ret=carg(z);
```

备　注　　carg 函数群的分支切断线沿着负的实数轴。
carg 函数群返回值的范围为区间 [-π,+π]。

虚部

float cimagf(float complex z)
double cimag(double complex z)
long double cimagl(long double complex z)

说　明　　计算虚部。

头文件　　<complex.h>

返回值　　作为实数的 z 的虚部值

参　数　　z　　要计算虚部的复数

例

```
#include <complex.h>
double complex z, ret;
   ret=cimag(z);
```

复共轭

float complex conjf(float complex z)
double complex conj(double complex z)
long double complex conjl(long double complex z)

说　明　　将虚数的符号取反，计算复共轭。

头文件　　<complex.h>

返回值　　z 的复共轭值

参　数　　z　　要计算复共轭值的复数

例

```
#include <complex.h>
double complex z, ret;
   ret=conj(z);
```

黎曼球面上的投影

float complex cprojf(float complex z)
double complex cproj(double complex z)
long double complex cprojl(long double complex z)

说　明　　计算黎曼球面上的投影。

头文件　　<complex.h>

返回值　　黎曼球面上的 z 的投影值

参　数　　z　　要计算黎曼球面上的投影值的复数

例

```
#include <complex.h>
double complex z, ret;
   ret=cproj(z);
```

实部

float crealf(float complex z)
double creal(double complex z)
long double creall(long double complex z)

说　明　　计算实部。

头文件　　<complex.h>

返回值　　z 的实部值

参　数　　z　　要计算实部值的复数

例

```
#include <complex.h>
double complex z, ret;
   ret=creal(z);
```

(16) <fenv.h>

存取浮点环境。
以下全部为实现定义：

种类	定义名	说明
型 （宏）	fenv_t	这是整个浮点环境的型。
	fexcept_t	这是浮点状态标志的型。
常数 （宏）	FE_DIVBYZERO FE_INEXACT FE_INVALID FE_OVERFLOW FE_UNDERFLOW FE_ALL_EXCEPT	这是在支持浮点异常时被定义的宏。
常数 （宏）	FE_DOWNWARD FE_TONEAREST FE_TOWARDZERO FE_UPWARD	这是浮点数舍入方向的宏。
常数 （宏）	FE_DFL_ENV	这是程序既定的浮点环境。
函数	feclearexcept	尝试清除浮点异常。
	fegetexceptflag	尝试保存到浮点标志状态的目标。
	feraiseexcept	尝试生成浮点异常。
	fesetexceptflag	尝试将浮点标志置位。
	fetestexcept	确认是否已将浮点标志置位。
	fegetround	取得舍入方向。
	fesetround	设定舍入方向。
	fegetenv	尝试取得浮点环境。
	feholdexcept	保存浮点环境，清除浮点状态标志，并且针对浮点异常，设定为无停止模式。
	fesetenv	尝试设定浮点环境。
	feupdateenv	尝试保存到浮点异常的自动存储区，设定浮点环境以及生成已保存的浮点异常。

异常的清除

long feclearexcept(long e)

说　明　　尝试清除浮点异常。

头文件　　<fenv.h>

返回值　　正常：0
异常：非 0 值

参　数　　e　　浮点异常

例
```
#include <fenv.h>
#pragma STDC FENV_ACCESS ON
int ret, e;
   ret=creal(e);
```

异常标志状态的取得

long fegetexceptflag(fexcept_t *f, long e)

说　明　　取得异常标志的状态。

头文件　　<fenv.h>

返回值　　正常：0
异常：非 0 值

参　数　　f　　指向异常标志状态保存目标的指针
e　　表示要取得状态的异常标志的值

例
```
#include <fenv.h>
#pragma STDC FENV_ACCESS ON
int ret;
fexcept_t f;
   ret=creal(&f, e);
```

异常的生成

long feraiseexcept(long e)

异常的生成

说　明　　尝试生成浮点异常。

头文件　　<fenv.h>

返回值　　正常：0
异常：非 0 值

参　数　　e　　指向要尝试生成异常的值

例
```
#include <fenv.h>
#pragma STDC FENV_ACCESS ON
int ret, e;
   ret=feraiseexcept(e);
```

备　注　　在生成“上溢”浮点异常或者“下溢”浮点异常时，feraiseexcept 函数是否生成“不正确的结果”浮点异常，是实现定义。

异常标志状态的设定

long fesetexceptflag(const fexcept_t *f, long e)

说　明　　设定异常标志的状态。

头文件　　<fenv.h>

返回值　　正常：0
异常：非 0 值

参　数　　f　　指向异常标志状态的取得源的指针
e　　表示要设定状态的异常标志的值

例
```
#include <fenv.h>
#pragma STDC FENV_ACCESS ON
fexcept_t f;
   fegetexceptflag(&f, FE_OVERFLOW) /* 保存标志状态。 */
   fesetexceptflag(&f,FE_OVERFLOW); /* 设定标志状态。 */
```

备　注　　必须在调用 fesetexceptflag 函数前，通过 fegetexceptflag 函数设定标志状态的取得源。
fesetexceptflag 函数不生成浮点异常而只设定标志的状态。

异常标志状态的判断

long fetestexcept(long e)

说　明　　判断异常标志的状态。

头文件　　<fenv.h>

返回值　　e 和浮点异常宏以位为单位的逻辑 “或”

参　数　　e　　表示要判断状态的标志 （可以为多个标志）的值

例

```
#include <fenv.h>
#pragma STDC FENV_ACCESS ON
int e = fetestexcept(FE_INVALID | FE_OVERFLOW);
if (e & FE_INVALID) fnc1();
if (e & FE_OVERFLOW) fnc2();
```

备　注　　能通过 1 次调用 fetestexcept 函数，判断多个浮点异常。

舍入方向的取得

long fegetround(void)

说　明　　取得当时的舍入方向。

头文件　　<fenv.h>

返回值　　正常：0
异常：当没有舍入方向的宏值或者无法决定舍入方向时，为负值。

例

```
#include <fenv.h>
#pragma STDC FENV_ACCESS ON
int ret = fgetround();
```

舍入方向的设定

long fesetround(long rnd)

说　明　　设定当时的舍入方向。

头文件　　<fenv.h>

返回值　　只在成功时为 0。

例
```
#include <fenv.h>
#include <assert.h>
void f(int round_dir)
{
#pragma STDC FENV_ACCESS ON
    int save_round;
    int setround_ok;
    save_round = fegetround();
    setround_ok = fesetround(round_dir);
    assert(setround_ok == 0);
    fesetround(save_round);
}
```

备　注　　如果向 fesetround 函数请求更改为和舍入方向的宏值不相等的值，就不更改舍入方向。

浮点环境的取得

long fegetenv(fenv_t *f)

说　明　　取得浮点环境。

头文件　　<fenv.h>

返回值　　正常：0
异常：非 0 值

参　数　　f　　指向浮点环境保存目标的指针

例
```
#include <fenv.h>
int ret, fenv_t f;
    ret=fegetenv(f);
```

浮点环境的保存

long feholdexcept(fenv_t *f)

说　明　　保存浮点环境。

头文件　　<fenv.h>

返回值　　只在成功时为 0。

参　数　　f　　指向浮点环境的指针

例

```
#include <fenv.h>
int ret, fenv_t f;
   ret=fesetenv(f);
```

备　注　　在保存浮点函数环境时，feholdexcept 函数清除浮点状态的标志，并且对全部浮点异常设定无停止（non-stop）模式。在设定无停止模式时，即使在发生浮点异常时，也继续执行。

环境的设定

long fesetenv(const fenv_t *f)

说　明　　设定浮点环境。

头文件　　<fenv.h>

返回值　　正常：0
异常：非 0 值

参　数　　f　　指向浮点环境的指针

例

```
#include <fenv.h>
int ret, fenv_t f;
   ret=fesetenv(f);
```

备　注　　必须给要设定的环境指定由 fegetenv 函数或者 feholdexcept 函数设定的环境，或者指定和浮点环境宏相等的环境。

long feupdateenv(const fenv_t *f)

说　明　　在保持已发生异常的情况下，设定浮点环境。

头文件　　<fenv.h>

返回值　　正常：0
异常：非 0 值

参　数　　f　　指向要设定的浮点环境的指针

例

```
#include <fenv.h>
double f(double x)
{
#pragma STDC FENV_ACCESS ON
    double ret;
    fenv_t prev_env;
    if (feholdexcept(&prev_env))
        return /* 环境有问题。 */;
   // 计算 ret。
    if (/* 调查是否发生下溢。 */)
        if (feclearexcept(FE_UNDERFLOW))
            return /* 环境有问题。 */;
        if (feupdateenv(&prev_env))
            return /* 环境有问题。 */;
    return ret;
}
```

备　注　　必须给要设定的浮点环境指定通过调用 fegetenv 函数或者 feholdexcept 函数设定的目标，或者指定和浮点环境宏相等的环境。

(17) <inttypes.h>

对整数型进行扩展。
以下全部为实现定义：

种类	定义名	说明
型 （宏）	Imaxdiv_t	这是 imaxdiv 函数返回值的型。
变量 （宏）	PRIdN PRIdLEASTN PRIdFASTN PRIdMAX PRIdPTR PRIiN PRIiLEASTN PRIiFASTN PRIiMAX PRIiPTR PRIoN PRIoLEASTN PRIoFASTN PRIoMAX PRIoPTR PRIuN PRIuLEASTN PRIuFASTN PRIuMAX PRIuPTR PRIxN PRIxLEASTN PRIxFASTN PRIxMAX PRIxPTR PRIXN PRIXLEASTN PRIXFASTN PRIXMAX PRIXPTR SCNdN SCNdLEASTN SCNdFASTN SCNdMAX SCNdPTR SCNiN SCNiLEASTN SCNiFASTN SCNiMAX SCNiPTR	

种类	定义名	说明
变量 （宏）	SCNoN	
	SCNoLEASTN	
	SCNoFASTN	
	SCNoMAX	
	SCNoPTR	
	SCNuN	
	SCNuLEASTN	
	SCNuFASTN	
	SCNuMAX	
	SCNuPTR	
	SCNxN	
	SCNxLEASTN	
	SCNxFASTN	
	SCNxMAX	
	SCNxPTR	
函数	imaxabs	计算绝对值。
	imaxdiv	计算商和余数。
	strtoimax strtoumax	除了将字符串的最初部分转换为 intmax_t 型和 uintmax_t 型以外，和 strtol 函数、 strtoll 函数、 strtoul 函数、 strtoull 函数等效。
	wcstoimax wcstoumax	除了将宽字符串的最初部分转换为 intmax_t 型和 uintmax_t 型以外，和 wcstol 函数、 wcstoll 函数、 wcstoul 函数、 wcstoull 函数等效。

绝对值

intmax_t imaxabs(intmax_t a)

说　明　　计算绝对值。

头文件　　<inttypes.h>

返回值　　a 的绝对值

参　数　　a　　要计算绝对值的值

例

```
#include <inttypes.h>
intmax a, ret;
   ret=imaxabs(a);
```

除法运算

intmaxdiv_t imaxdiv(intmax_t n, intmax_t d)

说　明　　进行除法运算。

头文件　　<inttypes.h>

返回值　　由商和余数构成的除法运算结果

参　数　　n　　要进行除法运算的值
d

例

```
#include <inttypes.h>
intmax_t n, m;
intmaxdiv_t ret;
   ret=imaxdiv(n, m);
```

将字符串转换为 intmax_t 型

intmax_t strtoimax(const char *nptr, char **endptr, long base)
uintmax_t strtoumax(const char *nptr, char **endptr, long base)

说　明　　将表示数的字符串转换为 intmax_t 型整数。

头文件　　<inttypes.h>

返回值　　正常：　当 nptr 指向的字符串以不构成整数的字符开始时：0
　　当 nptr 指向的字符串以构成整数的字符开始时：转换后的 intmax_t 型整数值
　　异常：　当转换后的值发生上溢时：INTMAX_MAX、INTMAX_MIN 或者 UINTMAX_MAX

参　数

nptr	指向要转换的表示数的字符串指针
endptr	指向存储区的指针，此存储区保存最初不构成整数的字符。
base	转换基数（0 或者 2 ～ 36）

例

```
#include <inttypes.h>
intmax_t ret;
const char *nptr;
char **endptr;
int base;
   ret=strtoimax(nptr,endptr,base);
```

错误条件　　如果转换后的值发生上溢，就给 errno 设定 ERANGE。

备　注　　strtoimax 函数和 strtoumax 函数除了分别将字符串的最初部分转换为 intmax_t 型和 uintmax_t 型整数以外，和 strtol 函数、strtoll 函数、strtoul 函数、strtoull 函数等效。

将宽字符串转换为整数

intmax_t wcstoimax(const wchar_t * restrict nptr, wchar_t ** restrict endptr, long base)
uintmax_t wcstoumax(const wchar_t * restrict nptr, wchar_t ** restrict endptr, long base)

说　明　　将表示数的字符串转换为 intmax_t 型或者 uintmax_t 型的整数。

头文件　　<stddef.h>、<inttypes.h>

返回值　　正常：　当 nptr 指向的字符串以不构成整数的字符开始时：0
　　当 nptr 指向的字符串以构成整数的字符开始时：转换后的 intmax_t 型整数值
　　异常：　当转换后的值发生上溢时：INTMAX_MAX、INTMAX_MIN 或者 UINTMAX_MAX

参　数

nptr	指向要转换的表示数的字符串指针
endptr	指向存储区的指针，此存储区保存最初不构成整数的字符
base	转换基数（0 或者 2 ～ 36）

例

```
#include <stddef.h>
#include <inttypes.h>
intmax_t ret;
const char *nptr;
char **endptr;
int base;
   ret=wcstoimax(nptr,endptr,base);
```

错误条件　　如果转换后的值发生上溢，就给 errno 设定 ERANGE。

备　注　　wcstrtoimax 函数和 wcstrtoumax 函数除了分别将字符串的最初部分转换为 intmax_t 型和 uintmax_t 型整数以外，和 wcstol 函数、wcstoll 函数、wcstoul 函数、wcstoull 函数等效。

(18) <iso646h>

以下全部为宏定义：

种类	定义名	说明
宏	and	&&
	and_eq	&=
	bitand	&
	bitor	|
	compl	~
	not	!
	not_eq	!=
	or	||
	or_eq	|=
	xor	^
	xor_eq	^=

(19) <stdbool.h>

以下全部为宏定义：

种类	定义名	说明
宏 （变量）	bool	展开为 _Bool。
宏 （常数）	true	展开为 1。
	false	展开为 0。
	__bool_true_false_are_defined	展开为 1。

(20) <stdint.h>

以下全部为宏定义：

种类	定义名	说明
宏	int_least8_t uint_least8_t int_least16_t uint_least16_t int_least32_t uint_least32_t int_least64_t uint_least64_t	这是有至少能保存 8 位、 16 位、 32 位和 64 位的有符号 / 无符号整数型的容量的型。
	int_fast8_t uint_fast8_t int_fast16_t uint_fast16_t int_fast32_t uint_fast32_t int_fast64_t uint_fast64_t	这是能对 8 位、 16 位、 32 位和 64 位的有符号 / 无符号的整数型进行最快速度运算的型。
	intptr_t uintptr_t	这是能将指向 void 的指针进行相互转换的有符号 / 无符号的整数型。
	intmax_t uintmax_t	这是能表示全部有符号 / 无符号的整数型值的有符号 / 无符号的整数型。
	int*N*_t uint*N*_t	这是有 N 位宽度的有符号 / 无符号的整数型。
	INT*N*_MIN INT*N*_MAX UINT*N*_MAX	这是指定宽度的有符号整数型的最小值。 这是指定宽度的有符号整数型的最大值。 这是指定宽度的无符号整数型的最大值。
	INT_LEAST*N*_MIN INT_LEAST*N*_MAX UINT_LEAST*N*_MAX	这是指定最小宽度的有符号整数型的最小值。 这是指定最小宽度的有符号整数型的最大值。 这是指定最小宽度的无符号的整数型最大值。
	INT_FAST*N*_MIN INT_FAST*N*_MAX UINT_FAST*N*_MAX	这是指定最快、最小宽度的有符号整数型的最小值。 这是指定最宽、最小宽度的有符号整数型的最大值。 这是指定最快、最小宽度的无符号整数型的最大值。
	INTPTR_MIN INTPTR_MAX UINTPTR_MAX	这是能保持指针的有符号整数型的最小值。 这是能保持指针的有符号整数型的最大值。 这是能保持指针的无符号整数型的最大值。
	INTMAX_MIN INTMAX_MAX UINTMAX_MAX	这是指定最大宽度的有符号整数型的最小值。 这是指定最大宽度的有符号整数型的最大值。 这是指定最大宽度的无符号整数型的最大值。
	PTRDIFF_MIN PTRDIFF_MAX	–65535 +65535
	SIG_ATOMIC_MIN SIG_ATOMIC_MAX	–127 +127
	SIZE_MAX	65535

种类	定义名	说明
宏	WCHAR_MIN	0
	WCHAR_MAX	65535U
	WINT_MIN	0
	WINT_MAX	4294967295U
函数 （宏）	INT*N*_C	展开为与 Int_least*N*_t 对应的整数常数表达式。
	UINT*N*_C	展开为与 uInt_least*N*_t 对应的整数常数表达式。
	INT_MAX_C	展开为 intmax_t 的整数常数表达式。
	UINT_MAX_C	展开为 uintmax_t 的整数常数表达式。

(21) <tgmath.h>

以下全部为宏定义：

总称型的宏	<math.h> 的函数	<complex.h> 的函数
acos	acos	cacos
asin	asin	casin
atan	atan	catan
acosh	acosh	cacosh
asinh	asinh	casinh
atanh	atanh	catanh
cos	cos	ccos
sin	sin	csin
tan	tan	ctan
cosh	cosh	ccosh
sinh	sinh	csinh
tanh	tanh	ctanh
exp	exp	cexp
log	log	clog
pow	pow	cpow
sqrt	sqrt	csqrt
fabs	fabs	cfabs
atan2	atan2	—
cbrt	cbrt	—
ceil	ceil	—
copysign	copysign	—
erf	erf	—
erfc	erfc	—
exp2	exp2	—
expm1	expm1	—
fdim	fdim	—
floor	floor	—
fma	fma	—
fmax	fmax	—

总称型的宏	<math.h> 的函数	<complex.h> 的函数
fmin	fmin	—
fmod	fmod	—
frexp	frexp	—
hypot	hypot	—
ilogb	ilogb	—
ldexp	ldexp	—
lgamma	lgamma	—
llrint	llrint	—
llround	llround	—
log10	log10	—
log1p	log1p	—
log2	log2	—
logb	logb	—
lrint	lrint	—
lround	lround	—
nearbyint	nearbyint	—
nextafter	nextafter	—
nexttoward	nexttoward	—
remainder	remainder	—
remquo	remquo	—
rint	rint	—
round	round	—
scalbn	scalbn	—
scalbln	scalbln	—
tgamma	tgamma	—
trunc	trunc	—
carg	—	carg
cimag	—	cimag
conj	—	conj
cproj	—	cproj
creal		creal

(22) <wchar.h>

以下全部为宏定义：

种类	定义名	说明
宏	mbstate_t	这是保持多字节字符排列和宽字符排列之间所需的转换状态的型。
	wint_t	这是保持扩展字符的型。
常数 （宏）	WEOF	表示文件的结束。
函数	fwprintf	转换输出格式并且输出到流。
	vfwprintf	和用 va_list 替换可变个数实际参数排列后的 fwprintf 等效。
	swprintf	转换输出格式并且写到宽字符的数组。
	vswprintf	和用 va_list 替换可变个数实际参数排列后的 swprintf。
	wprintf	和将 stdout 作为实际参数附加在提供的实际参数之前的 fwprintf 等效。
	vwprintf	和用 va_list 替换可变个数实际参数排列后的 wprintf 等效。
	fwscanf	按照宽字符串的控制，从流输入后进行转换，并且赋值给目标。
	vfwscanf	和用 va_list 替换可变个数实际参数排列后的 fwscanf 等效。
	swscanf	按照宽字符串的控制进行转换，并且赋值给目标。
	vswscanf	和用 va_list 替换可变个数实际参数排列后的 swscanf 等效。
	wscanf	和将 stdin 作为实际参数附加在提供的实际参数之前的 fwscanf 等效。
	vwscanf	和用 va_list 替换可变个数实际参数排列后的 wscanf 等效。
	fgetwc	作为 wchar_t 型取进来并且转换为 wint_t 型。
	fgetws	将宽字符串保存到数组。
	fputwc	写宽字符。
	fputws	写宽字符串。
	fwide	设定输入 / 输出的单位。
	getwc	和 fgetwc 等效。
	getwchar	和给实际参数指定了 stdin 的 getwc 等效。
	putwc	和 fputwc 等效。
	putwchar	和给第 2 个参数指定了 stdout 的 putwc 等效。
	ungetwc	将宽字符返回到流。
	wcstod wcstof wcstold	将宽字符串的最初部分转换为 double 型、 float 型、 long double 型。
	wcstol wcstoll wcstoul wcstoull	将宽字符串的最初部分转换为 long int 型、long long int 型、unsigned long int 型和 unsigned long long int 型。
	wcscpy	复制宽字符串。
	wcsncpy	复制不超过 n 个的宽字符。
	wmemcpy	复制 n 个宽字符。
	wmemmove	复制 n 个宽字符。
	wcscat	复制宽字符串并且附加到宽字符串的末尾。
	wcsncat	复制不超过 n 个的宽字符串并且附加到宽字符串的末尾。
	wcscmp	比较宽字符串。
	wcsncmp	比较不超过 n 个宽字符的数组。
	wmemcmp	比较 n 个宽字符。
	wcschr	在宽字符串中检索宽字符。

种类	定义名	说明
函数	wcscspn	检索宽字符串中是否包含宽字符串。
	wcspbrk	检索宽字符串中包含宽字符串的最初位置。
	wcsrchr	检索宽字符串中宽字符最后出现的位置。
	wcsspn	从宽字符串中计算包含宽字符的起始部分的最大长度。
	wcsstr	从宽字符串中检索宽字符排列最初出现的位置。
	wcstok	将宽字符串拆分成用宽字符分开的字句串。
	wmemchr	从目标的开头检索 n 个宽字符中宽字符最初出现的位置。
	wcslen	计算宽字符串的长度。
	wmemset	复制 n 个宽字符。
	wctob	判断多字节字符是否能表示为 1 个字节。
	mbsinit	判断是否为初始转换状态。
	mbrlen	计算构成多字节字符的字节数。
	mbrtowc	将多字节字符转换为宽字符。
	wcrtomb	将宽字符转换为多字节字符。
	mbsrtowcs	将多字节字符排列转换为对应的宽字符排列。
	wcsrtombs	将宽字符排列转换为对应的多字节字符排列。

带格式的宽字符版的文件输出

long fwprintf(FILE *restrict fp, const wchar_t *restrict control [, arg] …)

说　明　　按格式将数据输出到流输入 / 输出文件。

头文件　　<stdio.h>、<wchar.h>

返回值　　正常：转换并输出的宽字符串个数
异常：负值

参　数

fp	文件指针
control	指向表示格式的宽字符串的指针
arg, …	按格式输出的数据排列

例

```
#include <stdio.h>
#include <wchar.h>
FILE *fp;
const wchar_t *control=L” %s” ;
int ret;
wchar_t buffer[]=L” Hello World\n” ;
   ret=fwprintf(fp, control, buffer);
```

错误条件

备　注　　fwprintf 函数是 fprintf 函数的宽字符对应版。

带格式的宽字符版可变个数参数的文件输出

long vfwprintf(FILE *restrict fp, const char *restrict control, va_list arg)

说　明　　按格式将可变个数的参数列表输出到指定的流输入 / 输出文件。

头文件　　<stdarg.h>、<stdio.h>、<wchar.h>

返回值　　正常：转换并输出的字符数
异常：负值

参　数

fp	文件指针
control	指向表示格式的宽字符串的指针
arg	参数列表

例

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>
FILE *fp;
const wchar_t *control=L"%d";
int  ret;

void prlist(int count ,...)
{
   va_list ap;
   int i;
   va_start(ap, count);
   for(i=0;i<count;i++)
      ret=vfprintf(fp, control, ap);
   va_end(ap);
}
```

备　注　　vfwprintf 函数是 vfprintf 函数的宽字符对应版。

带格式的宽字符串的输出

long swprintf(wchar_t *restrict s, size_t n, const wchar_t *restrict control [, arg] …)

说　明　　按格式转换数据并且输出到指定的区域。

头文件　　<stdio.h>、<wchar.h>

返回值　　正常：转换后的字符数
异常：当发生表示格式错误或者请求至少写 n 个宽字符时，为负值。

参　数

s	指向要输出数据的存储区的指针
n	要输出的宽字符数
control	指向表示格式的宽字符串的指针
arg,…	按格式输出的数据

例

```
#include <stdio.h>
#include <wchar.h>
wchar_t s*;
size_t n=12;
const wchar_t *control=” %s” ;
int ret;
wchar_t buffer[]=” Hello World\n” ;
   ret=swprintf(s, n, control, buffer);
```

错误条件　　如果将不正确的多字节字符串传递给 mbrtowc() 函数，就发生表示格式错误。

备　注　　swprintf 函数是 sprintf 函数的宽字符对应版。

宽字符版可变个数参数的字符串的输出

long vswprintf(wchar_t *restrict s, size_t n, const wchar_t *restrict control, va_list arg)

说　明　　按格式将可变个数的参数列表输出到指定的存储区。

头文件　　<stdarg.h>、<wchar.h>

返回值　　正常：转换后的字符数
异常：负值

参　数

s	指向要输出数据的存储区的指针
n	要输出的宽字符数
control	指向表示格式的宽字符串的指针
arg	参数列表

例

```
#include <stdarg.h>
#include <wchar.h>
wchar_t *s;
const wchar_t *control=L"%d";
int ret;

void prlist(int count ,...)
{
    va_list ap;
    int i;
    va_start(ap, count);
    for(i=0;i<count;i++) {
        ret=vswprintf(s, control, ap);
        va_arg(ap,int);
        s += ret;
    }
}
```

备　注　　vswprintf 函数是 vsprintf 函数的宽字符对应版。

带格式的宽字符的输出

long wprintf(const wchar_t *restrict control [, arg] …)

说　明　　按格式转换数据并且输出到标准输出文件（stdout）。

头文件　　<stdio.h>、<wchar.h>

返回值　　正常：转换并输出的宽字符数
异常：负值

参　数　　control　　指向表示格式的宽字符串的指针
arg,…　　按格式输出的数据

例

```
#include <stdio.h>
#include <wchar.h>
const wchar_t *control=L"%s";
int ret;
wchar_t buffer[]=L"Hello World\n";
   ret=wprintf(control,buffer);
```

备　注　　wprintf 函数是 printf 函数的宽字符对应版。

可变个数参数的宽字符的输出

long vwprintf(const wchar_t *restrict control, va_list arg)

说　明　　按格式将可变个数的参数列表输出到标准输出文件（stdout）。

头文件　　<stdarg.h>、<wchar.h>

返回值　　正常：转换并输出的字符数
异常：负值

参　数　　control　　指向表示格式的宽字符串的指针
arg　　参数列表

例

```
#include <stdarg.h>
#include <wchar.h>
FILE *fp;
const wchar_t *control=L"%d";
int ret;

void wprlist(int count ,...)
{
    va_list ap;
    int i;
    va_start(ap, count);
    for(i=0;i<count;i++)
        ret=vwprintf(control, ap);
    va_end(ap);
}
```

备　注　　vwprintf 函数是 vprintf 函数的宽字符对应版。

带格式的宽字符的文件输入

long fwscanf(FILE *restrict fp, const wchar_t *restrict control [, ptr] …)

说　明　　从流输入 / 输出文件输入数据并且按格式进行转换。

头文件　　<stdio.h>、<wchar.h>

返回值　　正常：输入转换成功的数据个数
异常：在转换输入数据前输入数据结束时：EOF

参　数

fp	文件指针
control	指向表示格式的宽字符串的指针
ptr	指向保存输入数据的存储区的指针

例

```
#include <stdio.h>
#include <wchar.h>
FILE *fp;
const wchar_t *control=L” %d” ;
int ret, buffer[10];
   ret=fwscanf(fp, control, buffer);
```

备　注　　fwscanf 函数是 fscanf 函数的宽字符对应版。

带格式的可变个数参数宽字符的文件输入

long vfwscanf(FILE *restrict fp, const wchar_t *restrict control , va_list arg)

说　明　　从流输入 / 输出文件输入数据并且按格式进行转换。

头文件　　<stdarg.h>、<stdio.h>、<wchar.h>

返回值　　正常：输入转换成功的数据个数
异常：在转换输入数据前输入数据结束时：EOF

参　数

fp	文件指针
control	指向表示格式的宽字符串的指针
arg	参数列表

例

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>
FILE *fp;
const wchar_t *control=L"%d";
int ret;

void prlist(int count ,...)
{
   va_list ap;
   int i;
   va_start(ap, count);
   for(i=0;i<count;i++)
      ret=vfwscanf(fp, control, ap);
   va_end(ap);
}
```

备　注　　vfwscanf 函数是 vfscanf 函数的宽字符对应版。

带格式的宽字符串的输入

long swscanf(const wchar_t *restrict s, const wchar_t *restrict control [, ptr] …)

说　明　　从指定的存储区输入数据并且按格式进行转换。

头文件　　<stdio.h>、<wchar.h>

返回值　　正常：输入转换成功的数据个数
异常：EOF

参　数

s	有输入数据的存储区
control	指向表示格式的宽字符串的指针
ptr,…	指向保存输入转换数据的存储区的指针

例

```
#include <stdio.h>
#include <wchar.h>
const wchar_t *s, *control=L"%d";
int ret,buffer[10];
ret=swscanf(s, control, buffer);
```

备　注　　swscanf 函数是 sscanf 函数的宽字符对应版。

带格式的可变个数参数宽字符串的输入

long vswscanf(const wchar_t *restrict s, const wchar_t *restrict control, va_list arg)

说　明　　从指定的存储区输入数据并且按格式进行转换。

头文件　　<stdarg.h>、<wchar.h>

返回值　　正常：输入转换成功的数据个数
异常：EOF

参　数

s	有输入数据的存储区
control	指向表示格式的宽字符串的指针
arg	参数列表

例

```
#include <stdarg.h>
#include <wchar.h>
const wchar_t *s, *control=L"%d";
int ret,buffer[10];
ret=vswscanf(s, control, buffer);
```

带格式的宽字符的输入

long wscanf(const wchar_t *control [, ptr] …)

说　明　　从标准输入文件（stdin）输入数据并且按格式进行转换。

头文件　　<wchar.h>

返回值　　正常：输入转换成功的数据个数
异常：EOF

参　数　　control　　指向表示格式的宽字符串的指针
ptr,…　　指向保存输入转换数据的存储区的指针

例

```
#include <wchar.h>
const wchar_t *control=L"%d";
int ret,buffer[10];
   ret=wscanf(control, buffer);
```

备　注　　wscanf 函数是 scanf 函数的宽字符对应版。

带格式的可变个数参数宽字符的文件输入

long vwscanf(const wchar_t *restrict control , va_list arg)

说　明　　从指定的存储区输入数据并且按格式进行转换。

头文件　　<stdarg.h>、<wchar.h>

返回值　　正常：输入转换成功的数据个数
异常：在转换输入数据前输入数据结束时：EOF

参　数　　control　　指向表示格式的宽字符串的指针
arg　　参数列表

例

```
#include <stdarg.h>
#include <wchar.h>

FILE *fp;
const wchar_t *control=L"%d";
int ret;

void prlist(int count ,...)
{
    va_list ap;
    int i;
    va_start(ap, count);
    for(i=0;i<count;i++)
        ret=vwscanf(control, ap);
    va_end(ap);
}
```

备　注　　vwscanf 函数是将 vscanf 函数变为能使用宽字符串格式的函数。

从文件输入 1 个宽字符

wint_t fgetwc(FILE *fp)

说　明　　从流输入 / 输出文件输入 1 个宽字符。

头文件　　<stdio.h>、<wchar.h>

返回值　　正常：　当文件结束时　：EOF
　　当文件未结束时　：输入的字符
　　异常：　EOF

参　数　　fp　　文件指针

例

```
#include <stdio.h>
#include <wchar.h>
FILE *fp;
wint_t ret;
   ret=fgetwc(fp);
```

错误条件　　如果发生读错误，就给该文件设定错误指示符。

备　注　　fgetwc 函数是将 fgetc 函数变为能输入宽字符的函数。

从文件输入宽字符串

wchar_t *fgetws(wchar_t *restrict s, long n, FILE *fp)

说　明　　从流输入 / 输出文件输入宽字符串。

头文件　　<stdio.h>、<wchar.h>

返回值　　正常：　当文件结束时　：NULL
　　当文件未结束时　：s
　　异常：　NULL

参　数

s	指向要输入宽字符串的存储区的指针
n	要输入宽字符串的存储区的字节数
fp	文件指针

例

```
#include <stdio.h>
#include <wchar.h>
wchar_t *s, *ret;
int n;
FILE *fp;
   ret=fgetws(s,n,fp);
```

备　注　　fgetws 函数是使 fgets 函数支持能输入宽字符串的函数。

将 1 个宽字符输出到文件

wint_t fputwc(wchar_t c, FILE *fp)

说　明　　将 1 个宽字符输出到流输入 / 输出文件。

头文件　　<stdio.h>、<wchar.h>

返回值　　正常：输出的宽字符
异常：EOF

参　数

c	要输出的字符
fp	文件指针

例

```
#include <stdio.h>
#include <wchar.h>
FILE *fp;
wchar_t c;
wint_t ret;
   ret=fputwc(c,fp);
```

错误条件　　如果发生写错误，就对该文件设定错误指示符。

备　注　　fputwc 函数是 fputc 函数的宽字符对应版。

将宽字符串输出到文件

long fputws(const wchar_t *restrict s, FILE *restrict fp)

说　明　　将宽字符串输出到流输入 / 输出文件。

头文件　　<stdio.h>、<wchar.h>

返回值　　正常：0
异常：EOF

参　数

s	指向要输出的宽字符串的指针
fp	文件指针

例

```
#include <stdio.h>
#include <wchar.h>
const wchar_t *s;
int ret;
FILE *fp;
   ret=fputws(s,fp);
```

备　注　　fputws 函数是 fputs 函数的宽字符对应版。

文件的输入单位的设定

long fwide(FILE *fp, long mode)

说　明　　设定文件的输入单位。

头文件　　<stdio.h>、<wchar.h>

返回值　　当设定宽字符单位时：大于 0 的值
当设定字节单位时：小于 0 的值
当没有输入 / 输出单位时：0

参　数

fp	文件指针
mode	表示输入单位的值

例

```
#include <stdio.h>
#include <wchar.h>
FILE *fp;
int mode, ret;
   ret=fwide(fp,mode);
```

备　注　　在流输入 / 输出单位已被决定的情况下， fwide 函数不更改输入单位。

从文件输入 1 个宽字符

long getwc(FILE *fp)

说　明　　从流输入 / 输出文件输入 1 个宽字符。

头文件　　<stdio.h>、<wchar.h>

返回值　　正常：　当文件结束时　：WEOF
　　当文件未结束时 ：输入的字符
　　异常：　EOF

参　数　　fp　　文件指针

例

```
#include <stdio.h>
#include <wchar.h>
FILE *fp;
int ret;
   ret=getwc(fp);
```

错误条件　　如果发生读错误，就对该文件设定错误指示符。

备　注　　getwc 函数和 fgetwc 函数等效，因为作为宏进行安装，所以有可能对 fp 至少进行 2 次评价。因此， fp 必须是没有副作用的表达式。

输入 1 个宽字符

long getwchar(void)

说　明　　从标准输入文件 （stdin）输入 1 个宽字符。

头文件　　<wchar.h>

返回值　　正常：　当文件结束时　：WEOF
　　当文件未结束时 ：输入的字符
　　异常：　EOF

例

```
#include <wchar.h>
int ret;
   ret=getwchar();
```

错误条件　　如果发生读错误，就对该文件设定错误指示符。

备　注　　getwchar 函数是 getchar 函数的宽字符对应版。

将 1 个宽字符输出到文件

wint_t putwc(wchar_t c, FILE *fp)

说　明　　将 1 个宽字符输出到流输入 / 输出文件。

头文件　　<stdio.h>、<wchar.h>

返回值　　正常：　输出的宽字符
异常：　WEOF

参　数

c	要输出的宽字符
fp	文件指针

例

```
#include <stdio.h>
#include <wchar.h>
FILE *fp;
wchar_t c;
wint_t ret;
   ret=putwc(c,fp);
```

错误条件　　如果发生写错误，就对该文件设定错误指示符。

备　注　　putwc 函数和 fputwc 等效，因为作为宏进行安装，所以有可能对 fp 至少进行 2 次评价。因此， fp 必须是没有副作用的表达式。

输出 1 个宽字符

wint_t putwchar(wchar_t c)

说　明　　将 1 个宽字符输出到标准输出文件（stdout）。

头文件　　<wchar.h>

返回值　　正常：输出的宽字符
异常：WEOF

参　数

c	要输出的宽字符

例

```
#include <wchar.h>
wint_t ret;
wchar_t c;
   ret=putwchar(c);
```

错误条件　　如果发生写错误，就对该文件设定错误指示符。

备　注　　putwchar 函数是 putchar 函数的宽字符对应版。

将 1 个宽字符返回到文件

wint_t ungetwc(wint_t c, FILE *fp)

说　明　　将 1 个宽字符返回到流输入 / 输出文件。

头文件　　<stdio.h>、<wchar.h>

返回值　　正常：返回的宽字符
异常：WEOF

参　数

c	要返回的宽字符
fp	文件指针

例

```
#include <stdio.h>
#include <wchar.h>
wint_t ret;
wchar_t c;
FILE *fp;
   ret=ungetwc(c,fp);
```

备　注　　ungetwc 函数是 ungetc 函数的宽字符对应版。

将宽字符串转换为浮点值

double wcstod(const wchar_t *restrict nptr, wchar_t **restrict endptr)
float wcstof(const wchar_t *restrict nptr, wchar_t **restrict endptr)
long double wcstold(const wchar_t *restrict nptr, wchar_t **restrict endptr)

说　明　　将宽字符串的最初部分转换为规定的型的浮点值。

头文件　　<wchar.h>

返回值　　正常：　当 nptr 指向的字符串以不构成浮点型的字符开始时：0
当 nptr 指向的字符串以构成浮点型的字符开始时：转换后的型的浮点值
异常：　当转换后的值发生上溢时：和转换字符串相同符号的 HUGE_VAL、 HUGE_VALF、HUGE_VALL
当转换后的值发生下溢时：0

参　数

nptr	指向要转换的表示数的字符串指针
endptr	指向存储区的指针，该存储区保存最初不构成浮点值的字符指针。

例

```
#include <wchar.h>
const wchar_t *nptr;
wchar_t **endptr;
double ret;
   ret=wcstod(nptr,endptr);
```

错误条件　　如果转换后的值发生上溢或者下溢，就设定 errno。

备　注　　wcstod 函数群是 strtod 函数群的宽字符对应版。

将宽字符串转换为整数值

long int wcstol(const wchar_t * restrict nptr, wchar_t ** restrict endptr, long base)
long long int wcstoll(const wchar_t * restrict nptr,wchar_t ** restrict endptr,long base)
unsigned long int wcstoul(const wchar_t * restrict nptr,wchar_t ** restrict endptr,long base)
unsigned long long int wcstoull(const wchar_t * restrict nptr,wchar_t ** restrict endptr,long base)

说　明　　将宽字符串的最初部分转换规定的型的整数值。

头文件　　<wchar.h>

返回值　　正常：　当 nptr 指向的字符串以不构成整数的字符开始时：0
　　当 nptr 指向的字符串以构成整数的字符开始时：转换后的型的整数值
　　异常：　当转换后的值发生上溢时：为 LONG_MIN、 LONG_MAX、 LLONG_MIN、 LLONG_MAX、 ULONG_MAX 或者 ULLONG_MAX （取决于要转换的字符串符号）。

参　数

nptr	指向要转换的表示数的字符串指针
endptr	指向存储区的指针，此存储区保存最初不构成整数的字符。
base	转换基数 （0 或者 2 ～ 36）

例

```
#include <wchar.h>
long ret;
const wchar_t *nptr;
wchar_t **endptr;
int base;
   ret=wcstoull(nptr,endptr,base);
```

错误条件　　如果转换后的值发生上溢，就设定 errno。

备　注　　wcstol 函数群是 strtol 函数群的字符对应版。

宽字符串的复制

wchar_t *wcscpy(wchar_t * restrict s1, const wchar_t * restrict s2)

说　明　　将包含空字符的源宽字符串内容复制到目标存储区。

头文件　　<wchar.h>

返回值　　s1 的值

参　数

s1	指向复制目标存储区的指针
s2	指向复制源字符串的指针

例

```
#include <wchar.h>
wchar_t *s1, *ret;
const wchar_t *s2;
   ret=wcscpy(s1,s2);
```

备　注　　wcscpy 函数群是 strcpy 函数群的宽字符对应版。

宽字符串的复制

wchar_t *wcsncpy(wchar_t * restrict s1, const wchar_t * restrict s2, size_t n)

说　明　　将指定字符数的源宽字符串复制到目标存储区。

头文件　　<wchar.h>

返回值　　s1 的值

参　数

s1	指向复制目标存储区的指针
s2	指向复制源字符串的指针
n	要复制的字符数

例

```
#include <wchar.h>
wchar_t *s1, *ret;
const wchar_t *s2;
size_t n;
   ret=wcsncpy(s1,s2,n);
```

备　注　　wcsnpy 函数是 strncpy 函数的宽字符对应版。

存储区的复制

wchar_t *wmemcpy(wchar_t *restrict s1, const wchar_t *restrict s2, size_t n)

说　明　　将指定容量的源存储区的内容复制到目标存储区。

头文件　　<wchar.h>

返回值　　s1 的值

参　数

s1	指向复制目标存储区的指针
s2	指向复制源存储区的指针
n	要复制的字符数

例

```
#include <wchar.h>
wchar_t *ret, *s1;
const wchar_t *s2;
size_t n;
   ret=wmemcpy(s1,s2,n);
```

备　注　　wmemcpy 函数是 memcpy 函数的宽字符对应版。

存储区的移动

wchar_t *wmemmove(wchar_t *s1, const wchar_t *s2, size_t n)

说　明　　将指定容量的源存储区的内容复制到目标存储区。
即使源存储区和目标存储区有重叠部分，因为在重写复制源的重叠部分前先进行复制，所以也能正常复制。

头文件　　<wchar.h>

返回值　　s1 的值

参　数

s1	指向复制目标存储区的指针
s2	指向复制源存储区的指针
n	要复制的字符数

例

```
#include <wchar.h>
wchar_t *ret, *s1;
const wchar_t *s2;
size_t n;
   ret=wmemmove(s1,s2,n);
```

备　注　　wmemmove 函数是 memmove 函数的宽字符对应版。

宽字符串和宽字符串的连接

wchar_t *wcscat(wchar_t *s1, const wchar_t *s2)

说　明　　将字符串连接到字符串的后面。

头文件　　<wchar.h>

返回值　　s1 的值

参　数　　s1　　指向被连接的字符串的指针
　　s2　　指向要连接的字符串的指针

例
```
#include <wchar.h>
wchar_t *s1, *ret;
const wchar_t *s2;
   ret=wcscat(s1,s2);
```

备　注　　wcscat 函数是 strcat 函数的宽字符对应版。

字符串的连接

wchar_t *wcsncat(wchar_t * restrict s1, const wchar_t * restrict s2, size_t n)

说　明　　将指定字符数的字符串连接到字符串。

头文件　　<wchar.h>

返回值　　s1 的值

参　数　　s1　　指向被连接的字符串的指针
　　s2　　指向要连接的字符串的指针
　　n　　要连接的字符数

例
```
#include <wchar.h>
wchar_t *s1, *ret;
const wchar_t *s2;
size_t n;
   ret=wcsncat(s1,s2,n);
```

备　注　　wcsncat 函数是 strncat 函数的宽字符对应版。

字符串的比较

long wcscmp(const wchar_t *s1, const wchar_t *s2)

说　明　　比较指定的 2 个字符串的内容。

头文件　　<wchar.h>

返回值　　当 s1 指定的字符串　>　s2 指定的字符串时：正值
当 s1 指定的字符串　==　s2 指定的字符串时：0
当 s1 指定的字符串　<　s2 指定的字符串时：负值

参　数　　s1　　指向被比较的字符串的指针
s2　　指向要比较的字符串的指针

例

```
#include <wchar.h>
const wchar_t *s1, *s2;
int ret;
   ret=wcscmp(s1,s2);
```

备　注　　wcscmp 函数是 strcmp 函数的宽字符对应版。

字符串的比较

long wcsncmp(const wchar_t *s1, const wchar_t *s2, size_t n)

说　明　　对指定的 2 个字符串进行指定字符数的比较。

头文件　　<wchar.h>

返回值　　当 s1 指定的字符串　>　s2 指定的字符串时：正值
当 s1 指定的字符串　==　s2 指定的字符串时：0
当 s1 指定的字符串　<　s2 指定的字符串时：负值

参　数　　s1　　指向被比较的字符串的指针
s2　　指向要比较的字符串的指针
n　　要比较的字符数的最大值

例

```
#include <wchar.h>
const wchar_t *s1, *s2;
size_t n;
int ret;
   ret=wcsncmp(s1,s2,n);
```

备　注　　wcsncmp 函数是 strncmp 函数的宽字符对应版。

存储区的比较

long wmemcmp(const wchar_t * s1, const wchar_t * s2, size_t n)

说　明　　比较指定的 2 个存储区的内容。

头文件　　<wchar.h>

返回值　　当 s1 指定的存储区　>　s2 指定的存储区时：正值
当 s1 指定的存储区　==　s2 指定的存储区时：0
当 s1 指定的存储区　<　s2 指定的存储区时：负值

参　数

s1	指向被比较的存储区的指针
s2	指向要比较的存储区的指针
n	要比较的存储区的字符数

例

```
#include <wchar.h>
const wchar_t *s1, *s2;
size_t n;
int ret;
   ret=wmemcmp(s1,s2,n);
```

备　注　　wmemcmp 函数是 memcmp 函数的宽字符对应版。

最初的字符位置

wchar_t *wcschr(const wchar_t *s, wchar_t c)

说　明　　在指定的字符串中检索指定的字符最初出现的位置。

头文件　　<wchar.h>

返回值　　当找到要检索的字符时　：指向找到的字符指针
当未找到要检索的字符时　：NULL

参　数

s	指向要检索的字符串的指针
c	要检索的字符

例

```
#include <wchar.h>
const wchar_t *s;
int c;
char *ret;
   ret=wcschr(s,c);
```

备　注　　wcschr 函数是 strchr 函数的宽字符对应版。

指定字符集最初出现前的字符数

size_t wcscspn(const wchar_t *s1, const wchar_t *s2)

说　明　　从头开始调查指定的字符串，取得另外指定的字符串中的字符最初出现前的字符数。

头文件　　<wchar.h>

返回值　　从 s1 的开头开始， s2 指定的字符串中的字符最初出现前的字符数。

参　数　　s1　　指向被调查的字符串的指针
　　s2　　指向用于调查 s1 的字符串的指针

例
```
#include <wchar.h>
const wchar_t *s1, *s2;
size_t ret;
   ret=wcscspn(s1,s2);
```

备　注　　wcscspn 函数是 strcspn 函数的宽字符对应版。

指定字符集最初出现的位置

wchar_t *wcspbrk(const wchar_t *s1, const wchar_t *s2)

说　明　　在指定的字符串中检索另外指定的字符串中的字符最初出现的位置。

头文件　　<wchar.h>

返回值　　当找到要检索的字符时　：指向找到的字符指针
　　当未找到要检索的字符时 ：NULL

参　数　　s1　　指向被检索的字符串的指针
　　s2　　指向表示在 s1 内要检索字符的字符串的指针

例
```
#include <wchar.h>
const wchar_t *s1, *s2;
char *ret;
   ret=wcspbrk(s1,s2);
```

备　注　　wcspbrk 函数是 strpbrk 函数的宽字符对应版。

最后的字符位置

wchar_t *wcsrchr(const wchar_t *s, wchar_t c)

说　明　在指定的字符串中检索指定的字符最后出现的位置。

头文件　<wchar.h>

返回值　当找到要检索的字符时　：指向找到的字符指针
当未找到要检索的字符时 ：NULL

参　数

s	指向被检索的字符串的指针
c	要检索的字符

例

```
#include <wchar.h>
const wchar_t *s;
int c;
wchar_t *ret;
   ret=wcsrchr(s,c);
```

指定字符集连续部分的长度

size_t wcsspn(const wchar_t *s1, const wchar_t *s2)

说　明　从头开始调查指定的字符串，取得另外指定的字符串中的字符从头连续出现的字符。

头文件　<wchar.h>

返回值　s2 指定的字符串中的字符从 s1 的开头连续出现的字符数

参　数

s1	指向被调查的字符串的指针
s2	指向用于调查 s1 的字符串的指针

例

```
#include <wchar.h>
const wchar_t *s1, *s2;
size_t ret;
   ret=wcsspn(s1,s2);
```

备　注　wcsspn 函数是 strspn 函数的宽字符对应版。

最初的字符串位置

wchar_t *wcsstr(const wchar_t *s1, const wchar_t *s2)

说　明　　在指定的字符串中检索另外指定的字符串最初出现的位置。

头文件　　<wchar.h>

返回值　　当找到要检索的字符时　：指向找到的字符指针
当未找到要检索的字符时　：NULL

参　数　　s1　　指向被检索的字符串的指针
s2　　指向要检索的字符串的指针

例

```
#include <wchar.h>
const wchar_t *s1, *s2;
wchar_t *ret;
   ret=wcsstr(s1,s2);
```

字句的拆分

wchar_t* wcstok(wchar_t * restrict s1, const wchar_t * restrict s2, wchar_t ** restrict ptr)

说　明　　将指定的字符串拆成若干个字句。

头文件　　<wchar.h>

返回值　　当拆成字句时　：指向已拆分字句的起始位置的指针
当没有拆成字句时　：NULL

参　数　　s1　　指向被拆成若干个字句的字符串的指针
s2　　指向用于拆分字符串的字符串指针
ptr　　指向在调用下一个函数时开始检索的字符串的指针

例

```
#include <wchar.h>
static wchar_t s1[] = L"?a???b,,,#c";
static wchar_t s2[] = L" \ t  \ t";
wchar_t *t, *p1, *p2;
t = wcstok(s1, L"?", &p1);     // t 指向字句 L"a"。
t = wcstok(NULL, L",", &p1);   // t 指向字句 L"??b"。
t = wcstok(s2, L" \ t", &p2);   // t 为 NULL 指针。
t = wcstok(NULL, L"#,", &p1); // t 指向字句 L"c"。
t = wcstok(NULL, L"?", &p1);   // t 为 NULL 指针。
```

备　注　　wcstok 函数是 strtok 函数的宽字符对应版。
在对相同的字符串进行第 2 次以后的检索时，必须给 s1 设定 NULL，给 ptr 设定在前一次对相同字符串进行函数调用时取得的值。

存储区内的字符检索

wchar_t *wmemchr(const wchar_t *s, wchar_t c, size_t n)

说　明　在指定的存储区中检索指定的字符最初出现的位置。

头文件　<wchar.h>

返回值　当找到要检索的字符时　：指向找到的字符指针
当未找到要检索的字符时 ：NULL

参　数

s	指向被检索的存储区的指针
c	要检索的字符
n	要检索的字符数

例

```
#include <wchar.h>
const wchar_t *s;
int c;
size_t n;
wchar_t *ret;
   ret=wmemchr(s,c,n);
```

备　注　wmemchr 函数是 memchr 函数的宽字符对应版。

宽字符串的字符数

size_t wcslen(const wchar_t *s)

说　明　计算除末尾 NULL 宽字符以外的宽字符串的长度。

头文件　<wchar.h>

返回值　宽字符串的字符数

参　数

s	指向要计算长度的宽字符串的指针

例

```
#include <wchar.h>
const wchar_t *s;
size_t ret;
   ret=wcslen(s);
```

备　注　wcslen 函数是 strlen 函数的宽字符对应版。

字符的重复

wchar_t *wmemset(wchar_t *s, wchar_t c, size_t n)

说　明　从指定的存储区的开头设定指定字符数的指定字符。

头文件　<wchar.h>

返回值　s 的值

参　数

s	指向要设定字符的存储区的指针
c	要设定的字符
n	要设定的字符数

例

```
#include <stdio.h>
#include <wchar.h>
wchar_t c, *s, *ret;
size_t n;
    ret=wmemset(s,c,n);
```

备　注　wmemset 函数是 memset 函数的宽字符对应版。

将宽字符转换为 1 字节

long wctob(wint_t c)

说　明　将宽字符转换为 1 字节表示。

头文件　<stdio.h>、<wchar.h>

返回值　正常：宽字符的 1 个字节的值
异常：EOF

参　数

c	宽字符

例

```
#include <stdio.h>
#include <wchar.h>
wint_t c;
int ret;
    ret=wctob(c);
```

错误条件　如果用 1 字节无法表示宽字符，就返回 EOF。

备　注　在 c 为扩展字符集的元素并且在初始移位的状态下，wctob 函数判断多字节字符是否对应 1 字节表示。

转换状态函数

long mbsinit(const mbstate_t *ps)

说　明　　判断指定的 mbstate_t 目标是否为初始转换状态。

头文件　　<wchar.h>

返回值　　在初始化状态下，为非 0 值。
在其他状态下，为 0。

参　数　　ps　　指向 mbstate_t 目标的指针

例

```
#include <wchar.h>
const mbstate_t *mt;
int ret;
   ret=mbsinit(mt);
```

多字节字符的字节数的取得

size_t mbrlen(const char * restrict s, size_t n, mbstate_t *restrict ps)

说　明　　取得指定的多字节字符的字节数。

头文件　　<wchar.h>

返回值

0	通过不超过 n 个的字节识别到 NULL 宽字符时
大于等于 1 且小于等于 n	通过不超过 n 个的字节识别到多字节字符时
(size_t)(-2)	只通过 n 个字节无法识别到全部的多字节字符时
(size_t)(-1)	遇到非法的多字节字符排列时

参　数

s	指向多字节字符串的指针
n	要识别的多字节字符的最大字节数
ps	指向 mbstate_t 目标的指针

例

```
#include <wchar.h>
const char *s;
size_t n;
const mbstate_t *mt;
int ret;
   ret=mbrlen(s, n, mt);
```

将多字节字符转换为宽字符

size_t mbrtowc(wchar_t * restrict pwc, const char * restrict s, size_t n, mbstate_t * restrict ps)

说　明　　将多字节字符转换为宽字符。

头文件　　<wchar.h>

返回值

0	通过不超过 n 个的字节识别到 NULL 宽字符时
大于等于 1 且小于等于 n	通过不超过 n 个的字节识别到多字节字符时
(size_t)(-2)	只通过 n 个字节无法识别到全部的多字节字符时
(size_t)(-1)	遇到非法的多字节字符排列时

参　数

pwc	指向保存已取得的宽字符的宽字符串的指针
s	指向多字节字符串的指针
n	要识别的多字节字符的最大字节数
ps	指向 mbstate_t 目标的指针

例

```
#include <wchar.h>
wchar_t *pwc;
const char *s;
size_t n, ret;
mbstate_t *ps;
   ret=mbrtowc(pwc, s, n, ps);
```

备　注　　如果遇到非法的多字节字符排列，就将宏 EILSEQ 的值保存到 errno，没有规定转换状态。

将宽字符转换为多字节字符

size_t wcrtomb(char * restrict s, wchar_t wc, mbstate_t * restrict ps)

说　明　　将宽字符转换为多字节字符。

头文件　　<wchar.h>

返回值　　正常：多字节字符的字节数
异常：(size_t)(-1)　遇到非法的多字节字符排列时

参　数

s	指向多字节字符串的指针
wc	要转换的宽字符
ps	指向 mbstate_t 目标的指针

例

```
#include <wchar.h>
wchar_t wc;
char *s;
size_t ret;
mbstate_t *ps;
ret=wctomb(s, wc, ps);
```

错误条件　　如果遇到非法的多字节字符排列，就将宏 EILSEQ 的值保存到 errno，没有规定转换状态。

备　注　　wcrtomb 函数决定的多字节字符的字节数包括移位序列，字节数不超过 MB_CUR_MAX。如果转换结果为 NULL 宽字符，就为初始转换状态。如果需要的话，就将用于返回初始移位状态的移位序列保存到宽字符的前面。

将多字节字符串转换为宽字符串

size_t mbstowcs(wchar_t * restrict pwcs, const char * restrict s, size_t n)

说　明　　将多字节字符串转换为宽字符串。

头文件　　<stdlib.h>

返回值　　正常：写到宽字符串的字符数
异常：(size_t)(-1)　遇到非法的多字节字符排列时

参　数

wcs	指向宽字符串的指针
s	指向多字节字符串的指针
n	要保存到宽字符串的宽字符数

例

```
#include <stdlib.h>
wchar_t *pwcs;
const char *s;
size_t n, ret;
   ret=mbstowcs(pwcs, s, n);
```

备　注　　mbstowcs 函数将在 s 指向的数组中的初始移位状态下开始的多字节字符排列转换为对应的宽字符排列，并且将不超过 n 个的宽字符保存到 pwcs 指向的数组。
如果发现 NULL 字符，就转换为 NULL 宽字符并且结束转换处理。除了 mbtowc 函数的转换状态不受影响以外，按调用 mbtowc 函数时的相同规则进行各多字节字符的转换。没有定义在区域相互重叠的目标之间进行复制时的运行。
即使是正常的返回值，也不包括末尾字符的字节。
当返回值为 n 时，数组没有以 NULL 字符结束。

将宽字符串转换为多字节字符串

size_t wcstombs(char * restrict s, const wchar_t * restrict pwcs, size_t n)

说　明　　将宽字符串转换为多字节字符串。

头文件　　<stdlib.h>

返回值　　正常：写到多字节字符串的字节数
异常：(size_t)(-1)　遇到非法的多字节字符排列时

参　数

s	指向多字节字符串的指针
pwcs	指向宽字符串的指针
n	要写到多字节字符串的字节数

例

```
#include <stdlib.h>
const char *s;
wchar_t *pwcs;
size_t n, ret;
   ret=wcstombs(s,pwcs,n);
```

备　注　　wcstombs 函数将 pwcs 指向的数组中的宽字符串转换为从初始移位状态开始对应的多字节字符排列，并且保存到 s 指向的数组。如果多字节字符的总数超过 n 字节的上限或者保存了 NULL 字符，就结束数组的保存。除了 wctomb 函数的转换状态不受影响以外，按调用 wctomb 函数时的相同规则进行各字符的转换。
没有定义在区域相互重叠的目标之间进行复制时的运行。
即使是正常的返回值，也不包括末尾字符的字节。
当返回值为 n 时，数组没有以 NULL 字符结束。

9.3.2 EC++ 类库

(1) 库的概要

以下说明 C++ 程序中能标准利用的 EC++ 类库的规格。在此说明有关类库的种类和对应的标准 include 文件。以下按库的结构，说明有关各类库的规格。

- 库的种类
  类库的种类和对应的标准include文件如表9.38所示。

表 9.38 类库的种类和标准 include 文件的对应

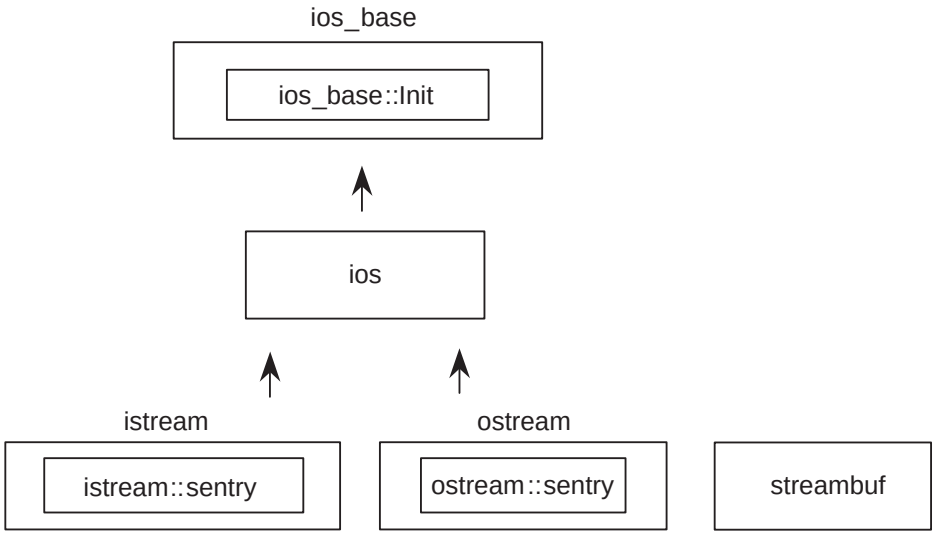
	库的种类	内容	标准 include 文件
1	流输入 / 输出类库	这是进行输入 / 输出操作的库。	<ios>、 <streambuf>、 <istream>、 <ostream>、 <iostream>、 <iomanip>
2	存储器操作库	这是确保和释放存储器的库。	<new>
3	复数计算类库	这是进行复数数据运算的库。	<complex>
4	字符串操作类库	这是进行字符串操作的库。	<string>

(2) 流输入 / 输出类库

流输入 / 输出类库对应的头文件如下：

- <ios>
  对设定输入/输出格式以及管理输入/输出状态的数据成员和函数成员进行定义。
  除ios类以外，还定义Init类和ios_base类。
- <streambuf>
  定义有关流缓冲区的函数。
- <istream>
  定义从输入流输入的函数。
- <ostream>
  定义从输出流输出的函数。
- <iostream>
  定义输入/输出函数。
- <iomanip>
  定义有参数的操纵符。

这些类的派生关系如下所示，箭头表示从派生类参照基类，而且 streambuf 类没有派生关系。



流输入 / 输出类库中通用的型名如下所示：

种类	定义名	说明
型	streamoff	这是用 long 型定义的型。
	streamsize	这是用 size_t 型定义的型。
	int_type	这是用 int 型定义的型。
	pos_type	这是用 long 型定义的型。
	off_type	这是用 long 型定义的型。

(a) ios_base::Init 类

种类	定义名	说明
变量	init_cnt	这是对流输入 / 输出目标数进行计数的静态数据成员。 需要通过低级接口例程初始化为 0。
函数	Init()	这是构造函数。
	~Init()	这是析构函数。

ios_base::Init::Init()

这是类 Init 的构造函数。

使 init_cnt 递增。

ios_base::Init::~Init()

这是类 Init 的析构函数。

使 init_cnt 递减。

(b) ios_base 类

种类	定义名	说明
型	fmtflags	这是表示格式控制信息的型。
	iostate	这是表示流缓冲区输入 / 输出状态的型。
	openmode	这是表示文件打开模式的型。
	seekdir	这是表示流缓冲区查找状态的型。
变量	fmtfl	这是格式标志。
	wide	这是字段宽度。
	prec	这是输出时的精度 （小数的位数）。
	fillch	这是填充字符。
函数	void _ec2p_init_base()	进行初始化。
	void _ec2p_copy_base(ios_base&ios_base_dt)	复制 ios_base_dt。
	ios_base()	这是构造函数。
	~ios_base()	这是析构函数。
	fmtflags flags() const	参照格式标志 （fmtfl）。
	fmtflags flags(fmtflags fmtflg)	将 fmtflg& 格式标志 （fmtfl）设定为格式标志 （fmtfl）。
	fmtflags setf(fmtflags fmtflg)	将 fmtflg 设定为格式标志 （fmtfl）。
	fmtflags setf(fmtflags fmtflg, fmtflags mask)	将 mask&fmtflg 设定为格式标志 （fmtfl）。
	void unsetf(fmtflags mask)	将 ~mask& 格式标志 （fmtfl）设定为格式标志 （fmtfl）。
	char fill() const	参照填充字符 （fillch）。
	char fill(char ch)	将 ch 设定为填充字符 （fillch）。
	int precision() const	参照精度 （prec）。
	streamsize precision(streamsize preci)	将 preci 设定为精度 （prec）。
	streamsize width() const	参照字段宽度 （wide）。
	streamsize width(streamsize wd)	将 wd 设定为字段宽度 （wide）。

ios_base::fmtflags

定义有关输入 / 输出的格式控制信息。

fmtflags 各位屏蔽的定义如下：

```
const ios_base::fmtflags ios_base::boolalpha    =  0x0000;
const ios_base::fmtflags ios_base::skipws       =  0x0001;
const ios_base::fmtflags ios_base::unitbuf      =  0x0002;
const ios_base::fmtflags ios_base::uppercase    =  0x0004;
const ios_base::fmtflags ios_base::showbase     =  0x0008;
const ios_base::fmtflags ios_base::showpoint    =  0x0010;
const ios_base::fmtflags ios_base::showpos      =  0x0020;
const ios_base::fmtflags ios_base::left         =  0x0040;
const ios_base::fmtflags ios_base::right        =  0x0080;
const ios_base::fmtflags ios_base::internal     =  0x0100;
const ios_base::fmtflags ios_base::adjustfield  =  0x01c0;
const ios_base::fmtflags ios_base::dec          =  0x0200;
const ios_base::fmtflags ios_base::oct          =  0x0400;
const ios_base::fmtflags ios_base::hex          =  0x0800;
const ios_base::fmtflags ios_base::basefield    =  0x0e00;
const ios_base::fmtflags ios_base::scientific   =  0x1000;
const ios_base::fmtflags ios_base::fixed        =  0x2000;
const ios_base::fmtflags ios_base::floatfield   =  0x3000;
const ios_base::fmtflags ios_base::_fmtmask     =  0x3fff;
```

ios_base::iostate

定义流缓冲区的输入 / 输出状态。

iostate 各位屏蔽的定义如下：

```
const ios_base::iostate ios_base::goodbit     =  0x0;
const ios_base::iostate ios_base::eofbit      =  0x1;
const ios_base::iostate ios_base::failbit     =  0x2;
const ios_base::iostate ios_base::badbit      =  0x4;
const ios_base::iostate ios_base::_statemask  =  0x7;
```

ios_base::openmode

定义文件的打开模式。

openmode 各位屏蔽的定义如下：

```
const ios_base::openmode ios_base::in      =  0x01;   打开输入文件。
const ios_base::openmode ios_base::out     =  0x02;   打开输出文件。
const ios_base::openmode ios_base::ate     =  0x04;   打开后只查找 1 次 eof。
const ios_base::openmode ios_base::app     =  0x08;   每次写时查找 eof。
const ios_base::openmode ios_base::trunc   =  0x10;   用重写模式打开文件。
const ios_base::openmode ios_base::binary  =  0x20;   用二进制模式打开文件。
```

ios_base::seekdir

定义流缓冲区的查找状态。

决定继续输入或者输出的流内的位置。

seekdir 各位屏蔽的定义如下：

```
const ios_base::seekdir ios_base::beg    =  0x0;
const ios_base::seekdir ios_base::cur    =  0x1;
const ios_base::seekdir ios_base::end    =  0x2;
```

void ios_base::_ec2p_init_base ()

用以下值进行初始设定：

```
fmtfl   =  skipws | dec;
wide    =  0;
prec    =  6;
fillch  =  ' ';
```

void ios_base::_ec2p_copy_base(ios_base& ios_base_dt)

复制 ios_base_dt。

ios_base::ios_base()

这是类 ios_base 的构造函数。

调用 Init::Init()。

ios_base::~ios_base()

这是类 ios_base 的析构函数。

ios_base::fmtflags ios_base::flags() const

参照格式标志 （fmtfl）。

返回值为格式标志 （fmtfl）。

ios_base::fmtflags ios_base::flags(fmtflags fmtflg)

将 fmtflg& 格式标志 （fmtfl）设定为格式标志 （fmtfl）。

返回值为设定前的格式标志 （fmtfl）。

ios_base::fmtflags ios_base::setf(fmtflags fmtflg)

将 fmtflg 设定为格式标志 （fmtfl）。

返回值为设定前的格式标志 （fmtfl）。

ios_base::fmtflags ios_base::setf(fmtflags fmtflg, fmtflags mask)

将 mask&fmtflg 的值设定为格式标志 （fmtfl）。

返回值为设定前的格式标志 （fmtfl）。

void ios_base::unsetf(fmtflags mask)

将 ~mask& 格式标志 （fmtfl）设定为格式标志 （fmtfl）。

char ios_base::fill() const

参照填充字符 （fillch）。

返回值为填充字符 （fillch）。

char ios_base::fill(char ch)

将 ch 设定为填充字符。

返回值为设定前的填充字符 （fillch）。

int ios_base::precision() const

参照精度 （prec）。

返回值为精度 （prec）。

streamsize ios_base::precision(streamsize preci)

将 preci 设定为精度 （prec）。

返回值为设定前的精度 （prec）。

streamsize ios_base::width() const

参照字段宽度 （wide）。

返回值为字段宽度 （wide）。

streamsize ios_base::width(streamsize wd)

将 wd 设定为字段宽度 （wide）。

返回值为设定前的字段宽度 （wide）。

(c) ios 类

种类	定义名	说明
变量	sb	这是指向 streambuf 目标的指针。
	tiestr	这是指向 ostream 目标的指针。
	state	这是 streambuf 的状态标志。
函数	ios()	这是构造函数。
	ios(streambuf* sbptr)	
	void init(streambuf* sbptr)	进行初始设定。
	virtual ~ios()	这是析构函数。
	operator void*() const	判断有无错误 （!state&(badbit | failbit)）。
	bool operator!() const	判断有无错误 （state&(badbit | failbit)）。
	iostate rdstate() const	参照状态标志 （state）。
	void clear(iostate st = goodbit)	清除指定状态 （st）以外的状态标志 （state）。
	void setstate(iostate st)	将 st 设定为状态标志 （state）。
	bool good() const	判断有无错误 （state==goodbit）。
	bool eof() const	判断是否为输入流的结尾 （state&eofbit）。
	bool bad() const	判断有无错误 （state&badbit）。
	bool fail() const	判断输入文本的模式是否和要求的模式不同 （state&(badbit | failbit)）。
	ostream* tie() const	参照指向 ostream 目标的指针 （tiestr）。
	ostream* tie(ostream* tstrptr)	将 tstrptr 设定为指向 ostream 目标的指针 （tiestr）。
	streambuf* rdbuf() const	参照指向 streambuf 目标的指针 （sb）。
	streambuf* rdbuf(streambuf* sbptr)	将 sbptr 设定为指向 streambuf 目标的指针 （sb）。
	ios& copyfmt(const ios& rhs)	复制 rhs 的状态标志 （state）。

ios::ios()

这是类 ios 的构造函数。

调用 init(0)，给其成员目标设定初始值。

ios::ios(streambuf* sbptr)

这是类 ios 的构造函数。

调用 init(sbptr)，给其成员目标设定初始值。

void ios::init(streambuf* sbptr)

将 sbptr 设定为 sb。

将 state、 tiestr 置 0。

virtual ios::~ios()

这是类 ios 的析构函数。

ios::operator void*() const

判断有无错误 （!state&(badbit | failbit)）。

返回值如下：

有错误时：false

无错误时：true

bool ios::operator!() const

判断有无错误 （state&(badbit | failbit)）。

返回值如下：

有错误时：true

无错误时：false

iostate ios::rdstate() const

参照状态标志 （state）。

返回值为状态标志 （state）。

void ios::clear(iostate st = goodbit)

清除指定状态 （st）以外的状态标志 （state）。

当指向 streambuf 目标的指针 （sb）为 0 时，将 badbit 设定为状态标志 （state）。

void ios::setstate(iostate st)

将 st 设定为状态标志 （state）。

bool ios::good() const

判断有无错误 （state==goodbit）。

返回值如下：

有错误时：false

无错误时：true

bool ios::eof() const

判断是否为输入流的结尾 （state&eofbit）。

返回值如下：

为输入流的结尾时：true

不为输入流的结尾时：false

bool ios::bad() const

判断有无错误 （state&badbit）。

返回值如下：

有错误时：true

无错误时：false

bool ios::fail() const

判断输入文本的模式是否和要求的模式不同 （state&(badbit | failbit)）。

返回值如下：

不同时：true

相同时：false

ostream* ios::tie() const

参照指向 ostream 目标的指针 （tiestr）。
返回值为指向 ostream 目标的指针 （tiestr）。

ostream* ios::tie(ostream* tstrptr)

将 tstrptr 设定为指向 ostream 目标的指针 （tiestr）。
返回值为设定前的指向 ostream 目标的指针 （tiestr）。

streambuf* ios::rdbuf() const

参照指向 streambuf 目标的指针 （sb）。
返回值为指向 streambuf 目标的指针 （sb）。

streambuf* ios::rdbuf(streambuf* sbptr)

将 sbptr 设定为指向 streambuf 目标的指针 （sb）。
返回值为设定前的指向 streambuf 目标的指针 （sb）。

ios& ios::copyfmt(const ios& rhs)

复制 rhs 的状态标志 （state）。
返回值为 *this。

(d) ios 类操纵符

种类	定义名	说明
函数	ios_base& showbase(ios_base& str)	设定为基数显示前缀模式。
	ios_base& noshowbase(ios_base& str)	清除基数显示前缀模式。
	ios_base& showpoint (ios_base& str)	设定为小数点生成模式。
	ios_base& noshowpoint (ios_base& str)	清除小数点生成模式。
	ios_base& showpos(ios_base& str)	设定为 + 符号生成模式。
	ios_base& noshowpos(ios_base& str)	清除 + 符号生成模式。
	ios_base& skipws(ios_base& str)	设定为空格跳读模式。
	ios_base& noskipws(ios_base& str)	清除空格跳读模式。
	ios_base& uppercase(ios_base& str)	设定为大写字母转换模式。
	ios_base& nouppercase(ios_base& str)	清除大写字母转换模式。
	ios_base& internal(ios_base& str)	设定为内部填充模式。
	ios_base& left(ios_base& str)	设定为左侧填充模式。
	ios_base& right(ios_base& str)	设定为右侧填充模式。
	ios_base& dec(ios_base& str)	设定为 10 进制模式。
	ios_base& hex(ios_base& str)	设定为 16 进制模式。
	ios_base& oct(ios_base& str)	设定为 8 进制模式。
	ios_base& fixed(ios_base& str)	设定为定点模式。
	ios_base& scientific(ios_base& str)	设定为科学记数法模式。

ios_base& showbase(ios_base& str)

设定为在数据开头的显示基数模式。

在 16 进制数的情况下，在行头附加 0x ；在 10 进制数的情况下，直接输出；在 8 进制数的情况下，在行头附加 0。

返回值为 str。

ios_base& noshowbase(ios_base& str)

清除在数据开头的显示基数模式。

返回值为 str。

ios_base& showpoint(ios_base& str)

设定为输出小数点的模式。

在没有指定精度时，用 6 位小数进行显示。

返回值为 str。

ios_base& noshowpoint(ios_base& str)

清除输出小数点的模式。

返回值为 str。

ios_base& showpos(ios_base& str)

设定为 + 符号生成输出模式 （给正数附加 + 符号）。
返回值为 str。

ios_base& noshowpos(ios_base& str)

清除 + 符号生成输出模式。
返回值为 str。

ios_base& skipws(ios_base& str)

设定为空格跳读输入模式 （跳读连续的空格）。
返回值为 str。

ios_base& noskipws(ios_base& str)

清除空格跳读输入模式。
返回值为 str。

ios_base& uppercase(ios_base& str)

设定为大写字母转换输出模式。
16 进制的基数表示为大写字母 0X，数值本身也为大写字母。
浮点的指数表示也为大写字母 E。
返回值为 str。

ios_base& nouppercase(ios_base& str)

清除大写字母转换输出模式。
返回值为 str。

ios_base& internal(ios_base& str)

在字段宽度 （wide）范围内进行输出时，按以下顺序输出：
符号、基数
填充字符 （fill）
数值
返回值为 str。

ios_base& left(ios_base& str)

在字段宽度 （wide）的范围内进行输出时向左对齐。
返回值为 str。

ios_base& right(ios_base& str)

在字段宽度 （wide）的范围内进行输出时向右对齐。
返回值为 str。

ios_base& dec(ios_base& str)

将转换基数设定为 10 进制模式。
返回值为 str。

ios_base& hex(ios_base& str)

将转换基数设定为 16 进制模式。
返回值为 str。

ios_base& oct(ios_base& str)

将转换基数设定为 8 进制模式。

返回值为 str。

ios_base& fixed(ios_base& str)

设定为定点输出模式。

返回值为 str。

ios_base& scientific(ios_base& str)

设定为科学记数法输出模式（指数记数法）。

返回值为 str。

(e) streambuf 类

种类	定义名	说明
常数	eof	表示文件的结束。
变量	_B_cnt_ptr	这是指向缓冲区有效数据长度的指针。
	B_beg_ptr	这是指向缓冲区基址指针的指针。
	_B_len_ptr	这是指向缓冲区长度的指针。
	B_next_ptr	这是指向缓冲区下一个读位置的指针。
	B_end_ptr	这是指向缓冲区结束位置的指针。
	B_beg_pptr	这是指向控制缓冲区起始位置的指针。
	B_next_pptr	这是指向缓冲区下一个读位置的指针。
	C_flg_ptr	这是指向文件的输入 / 输出控制标志的指针。
函数	char* _ec2p_getflag() const	参照文件输入 / 输出控制标志的指针。
	char*& _ec2p_gnptr()	参照指向缓冲区下一个读位置的指针。
	char*& _ec2p_pnptr()	参照指向缓冲区下一个写位置的指针。
	void _ec2p_bcntplus()	使缓冲区的有效数据长度递增。
	void _ec2p_bcntminus()	使缓冲区的有效数据长度递减。
	void _ec2p_setbPtr(char** begptr, char** curptr, long* cntptr, long* lenptr, char* flgptr)	设定 streambuf 的指针。
	streambuf()	这是构造函数。
	virtual ~streambuf()	这是析构函数。
	streambuf* pubsetbuf(char* s, streamsize n)	确保流输入 / 输出的缓冲区。此函数调用 setbuf(s,n)*1。
	pos_type pubseekoff(off_type off, ios _base::seekdir way, ios_base::openmode which = ios_base::in | ios_base::out)	用 way 指定的方法移动输入 / 输出流的读写位置。此函数调用 seekoff(off,way,which)*1。
	pos_type pubseekpos(pos_type sp, ios_base::openmode which = ios_base::in | ios_base::out)	求从流的起始位置到当前位置的偏移量。此函数调用 seekpos(sp,which)*1。
	int pubsync()	转储清除输出流。此函数调用 sync()*1。
	streamsize in_avail()	求从输入流的结束位置到当前位置的偏移量。
	int_type snextc()	读下一个字符。
	int_type sbumpc()	读一个字符，并且将指针设定到下一个位置。
	int_type sgetc()	读一个字符。
	int sgetn(char* s, streamsize n)	将 n 个字符设定到 s 指向的存储区。
	int_type sputbackc(char c)	返回读位置。

种类	定义名	说明
函数	int sungetc()	返回读位置。
	int sputc(char c)	插入字符 c。
	int_type sputn(const char* s, streamsize n)	插入 s 指向的 n 个字符。
	char* eback() const	求输入流的头指针。
	char* gptr() const	求输入流的 next 指针。
	char* egptr() const	求输入流的尾指针。
	void gbump(int n)	将输入流的 next 指针移动 n。
	void setg(char* gbeg, char* gnext, char* gend)	赋值输入流的各指针。
	char* pbase() const	求输出流的头指针。
	char* pptr() const	求输出流的 next 指针。
	char* epptr() const	求输出流的尾指针。
	void pbump(int n)	将输出流的 next 指针移动 n。
	void setp(char* pbeg, char* pend)	设定输出流的各指针。
	virtual streambuf* setbuf(char* s, streamsize n)*1	对派生的各类执行个别定义的运算。
	virtual pos_type seekoff(off_type off, ios_base::seekdir way, ios_base::openmode = (ios_base::openmode) (ios_base::in | ios_base::out))*1	更改流位置。
	virtual pos_type seekpos(pos_type sp, ios_base::openmode = (ios_base::openmode) (ios_base::in | ios_base::out)) *1	更改流位置。
	virtual int sync()*1	转储清除输出流。
	virtual int showmanyc()*1	求输入流的有效字符数。
	virtual streamsize xsgetn(char* s, streamsize n)	将 n 个字符设定到 s 指向的存储区。
	virtual int_type underflow()*1	不移动流位置，读一个字符。
	virtual int_type uflow()*1	读 next 指针的一个字符。
	virtual int_type pbackfail(int_type c = eof)*1	返回 c 表示的字符。
	virtual streamsize xsputn(const char* s, streamsize n)	插入 s 指向的 n 个字符。
	virtual int_type overflow(int_type c = eof)*1	将 c 插入到输出流。

【注】 *1 此类没有定义处理。

streambuf::streambuf()

这是构造函数。

用以下值进行初始化。

_B_cnt_ptr = B_beg_ptr = B_next_ptr = B_end_ptr = C_flg_ptr = _B_len_ptr = 0

B_beg_pptr = &B_beg_ptr

B_next_pptr = &B_next_ptr

virtual streambuf::~streambuf()

这是析构函数。

streambuf* streambuf::pubsetbuf(char* s, streamsize n)

确保流输入 / 输出的缓冲区。

此函数调用 setbuf(s,n)。

返回值为 *this。

pos_type streambuf::pubseekoff(off_type off, ios _base::seekdir way,
ios_base::openmode which = (ios_base::openmode)(ios_base::in | ios_base::out))

用 way 指定的方法移动输入 / 输出流的读写位置。

此函数调用 seekoff(off,way,which)。

返回值为新设定的流位置。

pos_type streambuf::pubseekpos(pos_type sp, ios_base::openmode which =
(ios_base::openmode)(ios_base::in | ios_base::out))

求从流的起始位置到当前位置的偏移量。

只将当前流指针移动 sp。

此函数调用 seekpos(sp,which)。

返回值为从起始位置开始的偏移量。

int streambuf::pubsync()

转储清除输出流。

此函数调用 sync()。

返回值为 0。

streamsize streambuf::in_avail()

求从输入流的结束位置到当前位置的偏移量。

返回值如下：

读位置有效时 ：从结束位置到当前位置的偏移量

读位置无效时 ：0 （调用 showmanyc()）

int_type streambuf::snextc()

读一个字符。如果读到的字符不为 eof，就读下一个字符。

返回值如下：

当不为 eof 时 ：读到的字符

当为 eof 时 ：eof

int_type streambuf::sbumpc()

读一个字符，并且将指针设定到下一个位置。

返回值如下：

当读位置有效时 ：读到的字符

当读位置无效时 ：eof

int_type streambuf::sgetc()

读一个字符。

返回值如下：

当读位置有效时 ：读到的字符

当读位置无效时 ：eof

int streambuf::sgetn(char* s, streamsize n)

将 n 个字符设定到 s 指向的存储区。

如果在字符串中检测到 eof，就结束设定。

返回值为设定的字符数。

int_type streambuf::sputbackc(char c)

在读位置正常并且读位置的返回数据和 c 相同时，返回读位置。

返回值如下：

当能返回时 ：c 的值

当不能返回时 ：eof

int streambuf::sungetc()

在读位置正常时，返回读位置。

返回值如下：

当能返回时 ：返回的值

当不能返回时 ：eof

int streambuf::sputc(char c)

插入字符 c。

返回值如下：

当写位置正确时 ：c 的值

当写位置不正确时 ：eof

int_type streambuf::sputn(const char* s, streamsize n)

插入 s 指向的 n 个字符。

当缓冲区小于 n 时，只插入缓冲区容量的字符。

返回值为插入的字符数。

char* streambuf::eback() const

求输入流的头指针。

返回值为头指针。

char* streambuf::gptr() const

求输入流的 next 指针。

返回值为 next 指针。

char* streambuf::egptr() const

求输入流的尾指针。

返回值为尾指针。

void streambuf::gbump(int n)

将输入流的 next 指针移动 n。

void streambuf::setg(char* gbeg, char* gnext, char* gend)

对输入流的各指针进行以下设定：

```
*B_beg_pptr     =  gbeg;
*B_next_pptr    =  gnext;
B_end_ptr       =  gend;
*_B_cnt_ptr     =  gend-gnext;
*_B_len_ptr     =  gend-gbeg;
```

char* streambuf::pbase() const

求输出流的头指针。

返回值为头指针。

char* streambuf::pptr() const

求输出流的 next 指针。

返回值为 next 指针。

char* streambuf::epptr() const

求输出流的尾指针。

返回值为尾指针。

void streambuf::pbump(int n)

将输出流的 next 指针移动 n。

void streambuf::setp(char* pbeg, char* pend)

对输出流的各指针进行以下设定：

```
*B_beg_pptr     =  pbeg;
*B_next_pptr    =  pbeg;
B_end_ptr       =  pend;
*_B_cnt_ptr     =  pend-pbeg;
*_B_len_ptr     =  pend-pbeg;
```

virtual streambuf* streambuf::setbuf(char* s, streamsize n)

对从 streambuf 派生的各类执行个别定义的运算。

返回值为 *this。此类没有定义处理。

virtual pos_type streambuf::seekoff(off_type off, ios_base::seekdir way, ios_base::openmode =
(ios_base::openmode)(ios_base::in | ios_base::out))

更改流位置。

返回值为 –1。此类没有定义处理。

virtual pos_type streambuf::seekpos(pos_type sp, ios_base::openmode =
(ios_base::openmode)(ios_base::in | ios_base::out))

更改流位置。

返回值为 –1。此类没有定义处理。

virtual int streambuf::sync()

转储清除输出流。

返回值为 0。此类没有定义处理。

virtual int streambuf::showmanyc()

求输入流的有效字符数。

返回值为 0。此类没有定义处理。

virtual streamsize streambuf::xsgetn(char* s, streamsize n)

将 n 个字符设定到 s 指向的存储区。

当缓冲区小于 n 时，只设定缓冲区容量的字符。

返回值为输入的字符数。

virtual int_type streambuf::underflow()

不移动流位置，读一个字符。

返回值为 eof。此类没有定义处理。

virtual int_type streambuf::uflow()

读 next 指针的一个字符。

返回值为 eof。此类没有定义处理。

virtual int_type streambuf::pbackfail(int_type c = eof)

返回 c 表示的字符。

返回值为 eof。此类没有定义处理。

virtual streamsize streambuf::xsputn(const char* s, streamsize n)

插入 s 指向的 n 个字符。

当缓冲区小于 n 时，只插入缓冲区容量的字符。

返回值为插入的字符数。

virtual int_type streambuf::overflow(int_type c = eof)

将 c 插入到输出流。

返回值为 eof。此类没有定义处理。

(f) istream::sentry 类

种类	定义名	说明
变量	ok_	表示是否为可输入的状态。
函数	sentry(istream& is, bool noskipws = false)	这是构造函数。
	~sentry()	这是析构函数。
	operator bool()	参照 ok_。

istream::sentry::sentry(istream& is, bool noskipws = _false)

这是内部类 sentry 的构造函数。

当 good() 不为 0 时，能进行带格式或者不带格式的输入。

当 tie() 不为 0 时，转储清除有关的输出流。

istream::sentry::~sentry()

这是内部类 sentry 的析构函数。

istream::sentry::operator bool()

参照 ok_。

返回值为 ok_。

(g) istream 类

种类	定义名	说明
变量	chcount	这是最后调用的输入函数抽出的字符数。
函数	int _ec2p_getistr(char* str, unsigned int dig, int mode)	用 dig 表示的基数转换 str。
	istream(streambuf* sb)	这是构造函数。
	virtual ~istream()	这是析构函数。
	istream& operator>>(bool& n)	将抽出的字符保存到 n。
	istream& operator>>(short& n)	
	istream& operator>>(unsigned short& n)	
	istream& operator>>(int& n)	
	istream& operator>>(unsigned int& n)	
	istream& operator>>(long& n)	
	istream& operator>>(unsigned long& n)	
	istream& operator>>(long long& n)	
	istream& operator>>(unsigned long long& n)	
	istream& operator>>(float& n)	
	istream& operator>>(double& n)	
	istream& operator>>(long double& n)	
	istream& operator>>(void*& p)	转换为指向 void 的指针并且保存到 p。
	istream& operator>>(streambuf* sb)	抽出字符，保存到 sb 指向的存储区。
	streamsize gcount() const	求 chcount （抽出的字符数）。
	int_type get()	抽出字符。
	istream& get(char& c)	抽出字符并且保存到 c。
	istream& get(signed char& c)	
	istream& get(unsigned char& c)	
	istream& get(char* s, streamsize n)	抽出长度为 n–1 的字符串，保存到 s 指向的存储区。
	istream& get(signed char* s, streamsize n)	
	istream& get(unsigned char* s, streamsize n)	
	istream& get(char* s, streamsize n, char delim)	抽出长度为 n–1 的字符串，保存到 s 指向的存储区。如果在字符串内检测到 'delim'，就结束输入。
	istream& get(signed char* s, streamsize n, char delim)	
	istream& get(unsigned char* s, streamsize n, char delim)	
	istream& get(streambuf& sb)	抽出字符串，保存到 sb 指向的存储区。
	istream& get(streambuf& sb, char delim)	抽出字符串，保存到 sb 指向的存储区。如果在中途检测到字符 'delim'，就结束输入。
	istream& getline(char* s, streamsize n)	抽出长度为 n–1 的字符串，保存到 s 指向的存储区。
	istream& getline(signed char* s, streamsize n)	
	istream& getline(unsigned char* s, streamsize n)	

种类	定义名	说明
函数	istream& getline(char* s, streamsize n, char delim)	抽出长度为 n–1 的字符串，保存到 s 指向的存储区。如果在中途检测到字符 'delim'，就结束输入。
	istream& getline(signed char* s, streamsize n, char delim)	
	istream& getline(unsigned char* s, streamsize n, char delim)	
	istream& ignore(streamsize n = 1, int_type delim = streambuf::eof)	跳读 n 个字符。如果在中途检测到字符 'delim'，就中止跳读处理。
	int_type peek()	查找下一个能输入的输入字符。
	istream& read(char* s, streamsize n)	抽出长度为 n 的字符串，保存到 s 指向的存储区。
	istream& read(signed char* s, streamsize n)	
	istream& read(unsigned char* s, streamsize n)	
	streamsize readsome(char* s, streamsize n)	抽出长度为 n 的字符串，保存到 s 指向的存储区。
	streamsize readsome(signed char* s, streamsize n)	
	streamsize readsome(unsigned char* s, streamsize n)	
	istream& putback(char c)	将字符返回到输入流。
	istream& unget()	返回输入流的位置。
	int sync()	调查是否有输入流。此函数调用 streambuf::pubsync()。
	pos_type tellg()	调查输入流的位置。此函数调用 streambuf::pubseekoff(0,cur,in)。
	istream& seekg(pos_type pos)	只将当前流指针移动 pos。此函数调用 streambuf::pubseekpos(pos)。
	istream& seekg(off_type off, ios_base::seekdir dir)	用 dir 指定的方法移动输入流的读位置。此函数调用 streambuf::pubseekoff(off,dir)。

int istream::_ec2p_getistr(char* str, unsigned int dig, int mode)

用 dig 表示的基数转换 str。

返回值为转换后的基数。

istream::istream(streambuf* sb)

这是类 istream 的构造函数。

调用 ios::init(sb)。

进行 chcount=0 的设定。

virtual istream::~istream()

这是类 istream 的析构函数。

istream& istream::operator>>(bool& n)
istream& istream::operator>>(short& n)
istream& istream::operator>>(unsigned short& n)
istream& istream::operator>>(int& n)
istream& istream::operator>>(unsigned int& n)
istream& istream::operator>>(long& n)
istream& istream::operator>>(unsigned long& n)
istream& istream::operator>>(long long& n)
istream& istream::operator>>(unsigned long long& n)
istream& istream::operator>>(float& n)
istream& istream::operator>>(double& n)
istream& istream::operator>>(long double& n)

将抽出的字符保存到 n。
返回值为 *this。

istream& istream::operator>>(void*& p)

将抽出的字符转换为 void* 型，保存到 p 指向的存储区。
返回值为 *this。

istream& istream::operator>>(streambuf* sb)

抽出字符，保存到 sb 指向的存储区。
如果没有抽出的字符，就调用 setstate(failbit)。
返回值为 *this。

streamsize istream::gcount() const

参照 chcount （抽出的字符数）。
返回值为 chcount。

int_type istream::get()

抽出字符。
返回值如下：

当能抽出时 ：抽出的字符
当不能抽出时 ：调用 setstate(failbat)，为 streambuf::eof。

istream& istream::get(char& c)
istream& istream::get(signed char& c)
istream& istream::get(unsigned char& c)

抽出字符并且保存到 c。当抽出的字符为 streambuf::eof 时，设定 failbit。
返回值为 *this。

istream& istream::get(char* s, streamsize n)
istream& istream::get(signed char* s, streamsize n)
istream& istream::get(unsigned char* s, streamsize n)

抽出长度为 n–1 的字符串，保存到 s 指向的存储区。
当 ok_==false 或者抽出的字符数为 0 时，设定 failbit。
返回值为 *this。

istream& istream::get(char* s, streamsize n, char delim)
istream& istream::get(signed char* s, streamsize n, char delim)
istream& istream::get(unsigned char* s, streamsize n, char delim)

抽出长度为 n–1 的字符串，保存到 s 指向的存储区。
如果在字符串内检测到 'delim'，就结束输入。
当 ok_==false 或者抽出的字符数为 0 时，设定 failbit。
返回值为 *this。

istream& istream::get(streambuf& sb)

抽出字符串，保存到 sb 指向的存储区。
当 ok_==false 或者抽出的字符数为 0 时，设定 failbit。
返回值为 *this。

istream& istream::get(streambuf& sb, char delim)

抽出字符串，保存到 sb 指向的存储区。
如果在中途检测到字符 'delim'，就结束输入。
当 ok_==false 或者抽出的字符数为 0 时，设定 failbit。
返回值为 *this。

istream& istream::getline(char* s, streamsize n)
istream& istream::getline(signed char* s, streamsize n)
istream& istream::getline(unsigned char* s, streamsize n)

抽出长度为 n–1 的字符串，保存到 s 指向的存储区。
当 ok_==false 或者抽出的字符数为 0 时，设定 failbit。
返回值为 *this。

istream& istream::getline(char* s, streamsize n, char delim)
istream& istream::getline(signed char* s, streamsize n, char delim)
istream& istream::getline(unsigned char* s, streamsize n, char delim)

抽出长度为 n–1 的字符串，保存到 s 指向的存储区。
如果在中途检测到字符 'delim'，就结束输入。
当 ok_==false 或者抽出的字符数为 0 时，设定 failbit。
返回值为 *this。

istream& istream::ignore(streamsize n = 1, int_type delim = streambuf::eof)

跳读 n 个字符。
如果在中途检测到字符 'delim'，就中止跳读处理。
返回值为 *this。

int_type istream::peek()

查找下一个能输入的输入字符。
返回值如下：

当 ok_==false 时 ： streambuf::eof
当 ok_!=false 时 ： rdbuf()->sgetc()

istream& istream::read(char* s, streamsize n)
istream& istream::read(signed char* s, streamsize n)
istream& istream::read(unsigned char* s, streamsize n)

当 ok_!=false 时，抽出长度为 n 的字符串，保存到 s 指向的存储区。
当抽出的字符数不是 n 时，设定 eofbit。
返回值为 *this。

streamsize istream::readsome(char* s, streamsize n)
streamsize istream::readsome(signed char* s, streamsize n)
streamsize istream::readsome(unsigned char* s, streamsize n)

抽出长度为 n 的字符串，保存到 s 指向的存储区。
如果字符数大于流容量，就保存流容量的字符数。
返回值为抽出的字符数。

istream& istream::putback(char c)

将字符 c 返回到输入流。当返回的字符为 streambuf::eof 时，设定 badbit。
返回值为 *this。

istream& istream::unget()

将输入流的指针返回一个位置。
当抽出的字符为 streambuf::eof 时，设定 badbit。
返回值为 *this。

int istream::sync()

调查是否有输入流。
此函数调用 streambuf::pubsync()。
返回值如下：
当没有输入流时 ：streambuf::eof
当有输入流时 ：0

pos_type istream::tellg()

调查输入流的位置。
此函数调用 streambuf::pubseekoff(0,cur,in)。
返回值如下：
从流起始位置开始的偏移量
但是在输入处理发生错误时，返回值为 –1。

istream& istream::seekg(pos_type pos)

只将当前流指针移动 pos。
此函数调用 streambuf::pubseekpos(pos)。
返回值为 *this。

istream& istream::seekg(off_type off, ios_base::seekdir dir)

用 dir 指定的方法移动输入流的读位置。
此函数调用 streambuf::pubseekoff(off,dir)。
如果输入处理发生错误，就不进行处理。
返回值为 *this。

(h) istream 类操纵符

种类	定义名	说明
函数	istream& ws(istream& is)	跳读空格类字符。

istream& ws(istream& is)

跳读空格类字符。

返回值为 is。

(i) istream 成员外函数

种类	定义名	说明
函数	istream& operator>>(istream& in, char* s)	抽出字符串，保存到 s 指向的存储区。
	istream& operator>>(istream& in, signed char* s)	
	istream& operator>>(istream& in, unsigned char* s)	
	istream& operator>>(istream& in, char& c)	抽出字符，保存到 c。
	istream& operator>>(istream& in, singed char& c)	
	istream& operator>>(istream& in, unsigned char& c)	

istream& operator>>(istream& in, char* s)
istream& operator>>(istream& in, signed char* s)
istream& operator>>(istream& in, unsigned char* s)

抽出字符串，保存到 s 指向的存储区。

在保存 (字段宽度 –1) 个字符或者输入流中出现 streambuf::eof 或者下一个能输入的字符 c 为 isspace(c)==1 时，结束处理。当保存的字符数为 0 时，设定 failbit。

返回值为 in。

istream& operator>>(istream& in, char& c)
istream& operator>>(istream& in, singed char& c)
istream& operator>>(istream& in, unsigned char& c)

抽出字符，保存到 c。

当没有抽出输入时，设定 failbit。

返回值为 in。

(j) ostream::sentry 类

种类	定义名	说明
变量	ok_	表示是否为能输出的状态。
	_ _ec2p_os	这是指向 ostream 目标的指针。
函数	sentry(ostream& os)	这是构造函数。
	~sentry()	这是析构函数。
	operator bool()	参照 ok_。

ostream::sentry::sentry(ostream& os)

这是内部类 sentry 的构造函数。

如果 good() 不为 0 并且 tie() 不为 0，就调用 flush()。给 _ _ec2p_os 设定 os。

ostream::sentry::~sentry()

这是内部类 sentry 的析构函数。

如果 _ _ec2p_os->flags() & ios_base::unitbuf 为真，就调用 flush()。

ostream::sentry::operator bool()

参照 ok_。

返回值为 ok_。

(k) ostream 类

种类	定义名	说明
函数	ostream(streambuf* sbptr)	这是构造函数。
	virtual ~ostream()	这是析构函数。
	ostream& operator<<(bool n)	将 n 插入到输出流。
	ostream& operator<<(short n)	
	ostream& operator<<(unsigned short n)	
	ostream& operator<<(int n)	
	ostream& operator<<(unsigned int n)	
	ostream& operator<<(long n)	
	ostream& operator<<(unsigned long n)	
	ostream& operator<<(long long n)	
	ostream& operator<<(unsigned long long n)	
	ostream& operator<<(float n)	
	ostream& operator<<(double n)	
	ostream& operator<<(long double n)	
	ostream& operator<<(void* n)	
	ostream& operator<<(streambuf* sbptr)	将 sbptr 的输出行插入到输出流。
	ostream& put(char c)	将字符 c 插入到输出流。
	ostream& write(const char* s, streamsize n)	将 s 的 n 个字符插入到输出流。
	ostream& write(const signed char* s, streamsize n)	
	ostream& write(const unsigned char* s, streamsize n)	
	ostream& flush()	转储清除输出流。此函数调用 streambuf::pubsync()。
	pos_type tellp()	求当前的写位置。此函数调用 streambuf::pubseekoff(0,cur,out)
	ostream& seekp(pos_type pos)	求从流的起始位置到当前位置的偏移量。只将当前的流指针移动 pos。此函数调用 streambuf::pubseekpos(pos)。
	ostream& seekp(off_type off, seekdir dir)	以 dir 为基准，只将流的写位置移动 off。此函数调用 streambuf::pubseekoff(off,dir)。

ostream::ostream(streambuf* sbptr)

这是构造函数。

调用 ios(sbptr)。

virtual ostream::~ostream()

这是析构函数。

ostream& ostream::operator<<(bool n)
ostream& ostream::operator<<(short n)
ostream& ostream::operator<<(unsigned short n)
ostream& ostream::operator<<(int n)
ostream& ostream::operator<<(unsigned int n)
ostream& ostream::operator<<(long n)
ostream& ostream::operator<<(unsigned long n)
ostream& ostream::operator<<(long long n)
ostream& ostream::operator<<(unsigned long long n)
ostream& ostream::operator<<(float n)
ostream& ostream::operator<<(double n)
ostream& ostream::operator<<(long double n)
ostream& ostream::operator<<(void* n)

当 sentry::ok_==true 时，将 n 插入到输出流。

当 sentry::ok_==false 时，设定 failbit。

返回值为 *this。

ostream& ostream::operator<<(streambuf* sbptr)

当 sentry::ok_==true 时，将 sbptr 的输出行插入到输出流。

当 sentry::ok_==false 时，设定 failbit。

返回值为 *this。

ostream& ostream::put(char c)

当 sentry::ok_==true 并且 rdbuf()->sputc(c)!=streambuf::eof 时，将 c 插入到输出流。

在上述以外的情况下，设定 badbit。

返回值为 *this。

ostream& ostream::write(const char* s, streamsize n)
ostream& ostream::write(const signed char* s, streamsize n)
ostream& ostream::write(const unsigned char* s, streamsize n)

当 sentry::ok_==true 并且 rdbuf()->sputn(s, n)==n 时，将 s 的 n 个字符插入到输出流。

在上述以外的情况下，设定 badbit。

返回值为 *this。

ostream& ostream::flush()

转储清除输出流。

此函数调用 streambuf::pubsync()。

返回值为 *this。

pos_type ostream::tellp()

求当前的写位置。

此函数调用 streambuf::pubseekoff(0,cur,out)。

返回值如下：

当前的流位置

但是在处理过程中发生错误时，返回值为 –1。

ostream& ostream::seekp(pos_type pos)

在没有错误时，求从流的起始位置到当前位置的偏移量。

只将当前的流指针移动 pos。

此函数调用 streambuf::pubseekpos(pos)。

返回值为 *this。

ostream& ostream::seekp(off_type off, seekdir dir)

在没有错误时，以 dir 为基准将流位置移动 off。

此函数调用 streambuf::pubseekoff(off,dir)。

返回值为 *this。

(l) ostream 类操纵符

种类	定义名	说明
函数	ostream& endl(ostream& os)	插入换行字符，转储清除输出流。
	ostream& ends(ostream& os)	插入空码。
	ostream& flush(ostream& os)	转储清除输出流。

ostream& endl(ostream& os)

将换行字符插入到流。

转储清除输出流。此函数调用 flush()。

返回值为 os。

ostream& ends(ostream& os)

将空码插入到输出流。

返回值为 os。

ostream& flush(ostream& os)

转储清除输出流。此函数调用 streambuf::sync()。

返回值为 os。

(m) ostream 成员外函数

种类	定义名	说明
函数	ostream& operator<<(ostream& os, char s)	将 s 插入到输出流。
	ostream& operator<<(ostream& os, signed char s)	
	ostream& operator<<(ostream& os, unsigned char s)	
	ostream& operator<<(ostream& os, const char* s)	
	ostream& operator<<(ostream& os, const singed char*s)	
	ostream& operator<<(ostream& os, const unsigned char*s)	

ostream& operator<<(ostream& os, char s)
ostream& operator<<(ostream& os, signed char s)
ostream& operator<<(ostream& os, unsigned char s)
ostream& operator<<(ostream& os, const char* s)
ostream& operator<<(ostream& os, const singed char* s)
ostream& operator<<(ostream& os, const unsigned char* s)

当 sentry::ok_==true 并且没有错误时，将 s 插入到输出流。
在上述以外的情况下，设定 failbit。
返回值为 os。

(n) smanip 类操纵符

种类	定义名	说明
函数	smanip resetiosflags(ios_base::fmtflags mask)	清除 mask 值指定的标志。
	smanip setiosflags(ios_base::fmtflags mask)	设定格式标志 （fmtfl）。
	smanip setbase(int base)	设定输出时使用的基数。
	smanip setfill(char c)	设定填充字符 （fillch）。
	smanip setprecision(int n)	设定精度 （prec）。
	smanip setw(int n)	设定字段宽度 （wide）。

smanip resetiosflags(ios_base::fmtflags mask)

清除 mask 值指定的标志。

返回值为输入 / 输出对象的目标。

smanip setiosflags(ios_base::fmtflags mask)

设定格式标志 （fmtfl）。

返回值为输入 / 输出对象的目标。

smanip setbase(int base)

设定输出时使用的基数。

返回值为输入 / 输出对象的目标。

smanip setfill(char c)

设定填充字符 （fillch）。

返回值为输入 / 输出对象的目标。

smanip setprecision(int n)

设定精度 （prec）。

返回值为输入 / 输出对象的目标。

smanip setw(int n)

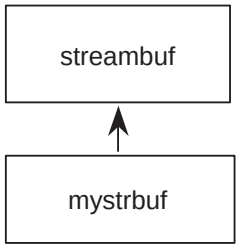
设定字段宽度 （wide）。

返回值为输入 / 输出对象的目标。

(o) EC++ 输入 / 输出库的使用例子

在对 istream、 ostream 的目标进行初始化时，通过使用指向 mystrbuf 类而不指向 streambuf 类的目标的指针，使输入 / 输出流变为能使用的状态。

类的派生关系如下，箭头表示从派生类参照基类。



种类	定义名	说明
变量	_file_Ptr	这是文件指针。
函数	mystrbuf ()	这是构造函数。对 streambuf 缓冲区进行初始化。
	mystrbuf(void* ptr)	
	virtual ~mystrbuf()	这是析构函数。
	void* myfptr() const	返回指向 FILE 型结构体的指针。
	mystrbuf* open(const char* filename, int mode)	指定文件名和模式，打开文件。
	mystrbuf* close()	关闭文件。
	virtual streambuf* setbuf(char* s, streamsize n)	确保流输入 / 输出的缓冲区。
	virtual pos_type seekoff(off_type off, ios_base::seekdir way, ios_base::openmode = (ios_base::openmode) (ios_base::in | ios_base::out))	更改流指针的位置。
	virtual pos_type seekpos(pos_type sp, ios_base::openmode = (ios_base::openmode) (ios_base::in | ios_base::out))	更改流指针的位置。
	virtual int sync()	转储清除流。
	virtual int showmanyc()	返回输入流的有效字符数。
	virtual int_type underflow()	不移动流位置，读一个字符。
	virtual int_type pbackfail(int_type c = streambuf::eof)	返回 c 表示的字符。
函数	virtual int_type overflow(int_type c = streambuf::eof)	插入 c 表示的字符。
	void _Init(_f_type* fp)	这是初始处理。

例：

```
#include <istream>
#include <ostream>
#include <mystrbuf>
#include <string>
#include <new>
#include <stdio.h>
void main(void)
{
    mystrbuf myfin(stdin);
    mystrbuf myfout(stdout);
    istream mycin(&myfin);
    ostream mycout(&myfout);

    int i;
    short s;
    long l;
    char c;
    string str;

    mycin >> i >> s >> l >> c >> str;
    mycout << "This is EC++ Library." << endl
           << i << s << l << c << str << endl;
    return;
}
```

(3) 存储器管理库

存储器管理库对应的头文件如下：

- <new>
  定义存储器的确保函数和释放函数。

通过给 _ec2p_new_handler 变量设定异常处理函数的地址，能在存储器确保失败时进行异常处理。
_ec2p_new_handler 为 static 变量，初始值为 NULL。如果使用此处理程序，就会失去可重入性。

异常处理函数所需的运行：

- 建立和返回能分配的区域。
- 对不能建立时的运行没有规定。

种类	定义名	说明
型	new_handler	这是指向返回 void 型函数的指针型。
变量	_ec2p_new_handler	这是指向异常处理函数的指针。
函数	void* operator new(size_t size)	确保 size 字节的区域。
	void* operator new[ ](size_t size)	确保 size 字节的数组区。
	void* operator new(size_t size, void* ptr)	将 ptr 指向的区域分配为存储区。
	void* operator new[ ](size_t size, void* ptr)	将 ptr 指向的区域分配为数组区。
	void operator delete(void* ptr)	释放区域。
	void operator delete[ ](void* ptr)	释放数组区。
	new_handler set_new_handler(new_handler new_P)	给 _ec2p_new_handler 设定异常处理函数地址（new_P）。

void* operator new(size_t size)

分配 size 字节的区域。
如果区域分配失败并且设定了 new_handler，就调用 new_handler。
返回值如下：

当区域确保成功时 ：指向 void 型的指针
当区域确保失败时 ：NULL

void* operator new[ ](size_t size)

确保 size 字节的数组区。
如果区域分配失败并且设定了 new_handler，就调用 new_handler。
返回值如下：

当区域确保成功时 ：指向 void 型的指针
当区域确保失败时 ：NULL

void* operator new(size_t size, void* ptr)

将 ptr 指向的区域分配为存储区。
返回值为 ptr。

void* operator new[ ](size_t size, void* ptr)

将 ptr 指向的区域分配为数组区。

返回值为 ptr。

void operator delete(void* ptr)

释放 ptr 指向的存储区。当 ptr 为 NULL 时，不进行任何操作。

void operator delete[ ](void* ptr)

释放 ptr 指向的数组区。当 ptr 为 NULL 时，不进行任何操作。

new_handler set_new_handler(new_handler new_P)

给 _ec2p_new_handler 设定 new_P。

返回值为 _ec2p_new_handler。

(4) 复数计算类库

复数计算类库对应的头文件如下：

- <complex>

  定义float_complex类、 double_complex类。

这些类没有派生关系。

(a) float_complex 类

种类	定义名	说明
型	value_type	这是 float 型。
变量	_re	定义 float 精度的实部。
	_im	定义 float 精度的虚部。
函数	float_complex(float re = 0.0f, float im = 0.0f)	这是构造函数。
	float_complex(const double_complex& rhs)	
	float real() const	求实部 （_re）。
	float imag() const	求虚部 （_im）。
	float_complex& operator=(float rhs)	将 rhs 复制到实部，虚部设定为 0.0f。
	float_complex& operator+=(float rhs)	实部加上 rhs，结果保存到 *this。
	float_complex& operator–=(float rhs)	实部减去 rhs，结果保存到 *this。
	float_complex& operator*=(float rhs)	乘 rhs，结果保存到 *this。
	float_complex& operator/=(float rhs)	除 rhs，结果保存到 *this。
	float_complex& operator=(const float_complex& rhs)	复制 rhs。
	float_complex& operator+=(const float_complex& rhs)	加上 rhs，结果保存到 *this。
	float_complex& operator–=(const float_complex& rhs)	减去 rhs，结果保存到 *this。
	float_complex& operator*=(const float_complex& rhs)	乘 rhs，结果保存到 *this。
	float_complex& operator/=(const float_complex& rhs)	除 rhs，结果保存到 *this。

float_complex::float_complex(float re = 0.0f, float im = 0.0f)

这是类 float_complex 的构造函数。

用以下值进行初始化：

_re = re;

_im = im;

float_complex::float_complex(const double_complex& rhs)

这是类 float_complex 的构造函数。

用以下值进行初始化：

_re = (float)rhs.real();

_im = (float)rhs.imag();

float float_complex::real() const

求实部。

返回值为 this->_re。

float float_complex::imag() const

求虚部。

返回值为 this->_im。

float_complex& float_complex::operator=(float rhs)

将 rhs 复制到实部 （_re），虚部 （_im）设定为 0.0f。

返回值为 *this。

float_complex& float_complex::operator+=(float rhs)

实部 （_re）加上 rhs，结果保存到实部 （_re），虚部 （_im）的值不变。

返回值为 *this。

float_complex& float_complex::operator–=(float rhs)

实部 （_re）减去 rhs，结果保存到实部 （_re），虚部 （_im）的值不变。

返回值为 *this。

float_complex& float_complex::operator*=(float rhs)

乘 rhs，结果保存到 *this。

(_re=_re*rhs, _im=_im*rhs)

返回值为 *this。

float_complex& float_complex::operator/=(float rhs)

除 rhs，结果保存到 *this。

(_re=_re/rhs, _im=_im/rhs)

返回值为 *this。

float_complex& float_complex::operator=(const float_complex& rhs)

复制 rhs。

返回值为 *this。

float_complex& float_complex::operator+=(const float_complex& rhs)

加上 rhs，结果保存到 *this。

返回值为 *this。

float_complex& float_complex::operator–=(const float_complex& rhs)

减去 rhs，结果保存到 *this。

返回值为 *this。

float_complex& float_complex::operator*=(const float_complex& rhs)

乘 rhs，结果保存到 *this。

返回值为 *this。

float_complex& float_complex::operator/=(const float_complex& rhs)

除 rhs，结果保存到 *this。

返回值为 *this。

(b) float_complex 成员外函数

种类	定义名	说明
函数	float_complex operator+(const float_complex& lhs)	进行 lhs 的一元 + 运算。
	float_complex operator+(const float_complex& lhs, const float_complex& rhs)	lhs 加上 rhs，结果保存到 lhs。
	float_complex operator+(const float_complex& lhs, const float& rhs)	
	float_complex operator+(const float& lhs, const float_complex& rhs)	
	float_complex operator–(const float_complex& lhs)	进行 lhs 的一元 – 运算。
	float_complex operator–(const float_complex& lhs, const float_complex& rhs)	lhs 减去 rhs，结果保存到 lhs。
	float_complex operator–(const float_complex& lhs, const float& rhs)	
	float_complex operator–(const float& lhs, const float_complex& rhs)	
	float_complex operator*(const float_complex& lhs, const float_complex& rhs)	lhs 乘 rhs，结果保存到 lhs。
	float_complex operator*(const float_complex& lhs, const float& rhs)	
	float_complex operator*(const float& lhs, const float_complex& rhs)	
	float_complex operator/(const float_complex& lhs, const float_complex& rhs)	lhs 除 rhs，结果保存到 lhs。
	float_complex operator/(const float_complex& lhs, const float& rhs)	
	float_complex operator/(const float& lhs, const float_complex& rhs)	

种类	定义名	说明
函数	bool operator==(const float_complex& lhs, const float_complex& rhs)	将 lhs 和 rhs 的实部和实部、虚部和虚部进行比较。
	bool operator==(const float_complex& lhs, const float& rhs)	
	bool operator==(const float& lhs, const float_complex& rhs)	
	bool operator!=(const float_complex& lhs, const float_complex& rhs)	将 lhs 和 rhs 的实部和实部、虚部和虚部进行比较。
	bool operator!=(const float_complex& lhs, const float& rhs)	
	bool operator!=(const float& lhs, const float_complex& rhs)	
	istream& operator>>(istream& is, float_complex& x)	以 u、(u) 或者 (u,v)（u：实部、v：虚部）格式输入 x。
	ostream& operator<<(ostream& os, float_complex& x)	以 u、(u) 或者 (u,v)（u：实部、v：虚部）格式输出 x。
	float real(const float_complex& x)	求实部。
	float imag(const float_complex& x)	求虚部。
	float abs(const float_comlex& x)	求绝对值。
	float arg(const float_complex& x)	求相位角。
	float norm(const float_complex& x)	求平方的绝对值。
	float_complex conj(const float_complex& x)	求共轭复数。
	float_complex polar(const float& rho, const float& theta)	求对应大小为 rho 并且相位角为 theta 的复数的 float_complex 值。
	float_complex cos(const float_complex& x)	求复数的余弦。
	float_complex cosh(const float_complex& x)	求复数的双曲余弦。
	float_complex exp(const float_complex& x)	求指数函数。
	float_complex log(const float_complex& x)	求自然对数。
	float_complex log10(const float_complex& x)	求常用对数。

种类	定义名	说明
函数	float_complex pow(const float_complex& x, int y)	求 x 的 y 乘方。
	float_complex pow(const float_complex& x, const float& y)	
	float_complex pow(const float_complex& x, const float_complex& y)	
	float_complex pow(const float& x, const float_complex& y)	
	float_complex sin(const float_complex& x)	求复数的正弦。
	float_complex sinh(const float_complex& x)	求复数的双曲正弦。
	float_complex sqrt(const float_complex& x)	求右半空间范围内的平方根。
	float_complex tan(const float_complex& x)	求复数的正切。
	float_complex tanh(const float_complex& x)	求复数的双曲正切。

float_complex operator+(const float_complex& lhs)

进行 lhs 的一元 + 运算。

返回值为 lhs。

float_complex operator+(const float_complex& lhs, const float_complex& rhs)
float_complex operator+(const float_complex& lhs, const float& rhs)
float_complex operator+(const float& lhs, const float_complex& rhs)

lhs 加上 rhs，结果保存到 lhs。

返回值为 float_complex(lhs)+=rhs。

float_complex operator–(const float_complex& lhs)

进行 lhs 的一元 – 运算。

返回值为 float_complex(-lhs.real(),-lhs.imag())。

float_complex operator–(const float_complex& lhs, const float_complex& rhs)
float_complex operator–(const float_complex& lhs, const float& rhs)
float_complex operator–(const float& lhs, const float_complex& rhs)

lhs 减去 rhs，结果保存到 lhs。

返回值为 float_complex(lhs)-=rhs。

float_complex operator*(const float_complex& lhs, const float_complex& rhs)
float_complex operator*(const float_complex& lhs, const float& rhs)
float_complex operator*(const float& lhs, const float_complex& rhs)

lhs 乘 rhs，结果保存到 lhs。

返回值为 float_complex(lhs)*=rhs。

float_complex operator/(const float_complex& lhs, const float_complex& rhs)
float_complex operator/(const float_complex& lhs, const float& rhs)
float_complex operator/(const float& lhs, const float_complex& rhs)

lhs 除 rhs，结果保存到 lhs。
返回值为 float_complex(lhs)/=rhs。

bool operator==(const float_complex& lhs, const float_complex& rhs)
bool operator==(const float_complex& lhs, const float& rhs)
bool operator==(const float& lhs, const float_complex& rhs)

将 lhs 和 rhs 的实部和实部、虚部和虚部进行比较。在 float 型参数的情况下，虚部假设为 float 型的 0.0f。
返回值为 lhs.real()==rhs.real() && lhs.imag()==rhs.imag()。

bool operator!=(const float_complex& lhs, const float_complex& rhs)
bool operator!=(const float_complex& lhs, const float& rhs)
bool operator!=(const float& lhs, const float_complex& rhs)

将 lhs 和 rhs 的实部和实部、虚部和虚部进行比较。在 float 型参数的情况下，虚部假设为 float 型的 0.0f。
返回值为 lhs.real()!=rhs.real() || lhs.imag()!=rhs.imag()。

istream& operator>>(istream& is, float_complex& x)

以 u、 (u) 或者 (u,v) （u 为实部， v 为虚部）格式输入 x。输入值被转换为 float_complex。
如果不以 u、 (u) 或者 (u,v) 格式输入 x，就调用 is.setstate(ios_base::failbit)。
返回值为 is。

ostream& operator<<(ostream& os, const float_complex& x)

将 x 输出到 os。
输出格式为 u、 (u) 或者 (u,v) （u 为实部， v 为虚部）。
返回值为 os。

float real(const float_complex& x)

求实部。
返回值为 x.real()。

float imag(const float_complex& x)

求虚部。
返回值为 x.imag()。

float abs(const float_complex& x)

求绝对值。
返回值为 $(|x.real()|^2 + |x.imag()|^2)^{1/2}$。

float arg(const float_complex& x)

求相位角。
返回值为 atan2f(x.imag() , x.real())。

float norm(const float_complex& x)

求平方的绝对值。
返回值为 $|x.real()|^2 + |x.imag()|^2$。

float_complex conj(const float_complex& x)

求共轭复数。

返回值为 float_complex(x.real(), (–1)*x.imag())。

float_complex polar(const float& rho, const float& theta)

求对应大小为 rho 并且相位角 （辐角）为 theta 的复数的 float_complex 值。

返回值为 float_complex(rho*cosf(theta), rho*sinf(theta))。

float_complex cos(const float_complex& x)

求复数的余弦。

返回值为 float_complex(cosf(x.real())*coshf(x.imag()), (–1)*sinf(x.real())*sinhf(x.imag()))。

float_complex cosh(const float_complex& x)

求复数的双曲余弦。

返回值为 cos(float_complex((–1)*x.imag(), x.real()))。

float_complex exp(const float_complex& x)

求指数函数。

返回值为 expf(x.real())*cosf(x.imag()),expf(x.real())*sinf(x.imag())。

float_complex log(const float_complex& x)

求以 e 为底的自然对数。

返回值为 float_complex(logf(abs(x)), arg(x))。

float_complex log10(const float_complex& x)

求以 10 为底的常用对数。

返回值为 float_complex(log10f(abs(x)), arg(x)/logf(10))。

float_complex pow(const float_complex& x, int y)
float_complex pow(const float_complex& x, const float& y)
float_complex pow(const float_complex& x, const float_complex& y)
float_complex pow(const float& x, const float_complex& y)

求 x 的 y 乘方。

当 pow(0,0) 时，为定义域错误。

返回值如下：

当 float_complex pow(const float_complex& x,const float_complex& y) 时 ： exp(y*logf(x))

上述以外 ： exp(y*log(x))

float_complex sin(const float_complex& x)

求复数的正弦。

返回值为 float_complex(sinf(x.real())*coshf(x.imag()), cosf(x.real())*sinhf(x.imag()))。

float_complex sinh(const float_complex& x)

求复数的双曲正弦。

返回值为 float_complex(0,–1)*sin(float_complex((–1)*x.imag(),x.real())) 。

float_complex sqrt(const float_complex& x)

求右半空间范围内的平方根。

返回值为 float_complex(sqrtf(abs(x))*cosf(arg(x)/2), sqrtf(abs(x))*sinf(arg(x)/2))。

float_complex tan(const float_complex& x)

求复数的正切。

返回值为 sin(x)/cos(x)。

float_complex tanh(const float_complex& x)

求复数的双曲正切。

返回值为 sinh(x)/cosh(x)。

(c) double_complex 类

种类	定义名	说明
型	value_type	这是 double 型。
变量	_re	定义 double 精度的实部。
	_im	定义 double 精度的虚部。
函数	double_complex(double re = 0.0, double im = 0.0)	这是构造函数。
	double_complex(const float_complex&)	
	double real() const	求实部。
	double imag() const	求虚部。
	double_complex& operator=(double rhs)	将 rhs 复制到实部，虚部设定为 0.0。
	double_complex& operator+=(double rhs)	实部加上 rhs，结果保存到 *this。
	double_complex& operator–=(double rhs)	实部减去 rhs，结果保存到 *this。
	double_complex& operator*=(double rhs)	乘 rhs，结果保存到 *this。
	double_complex& operator/=(double rhs)	除 rhs，结果保存到 *this。
	double_complex& operator=(const double_complex& rhs)	复制 rhs。
	double_complex& operator+=(const double_complex& rhs)	加上 rhs，结果保存到 *this。
	double_complex& operator–=(const double_complex& rhs)	减去 rhs，结果保存到 *this。
	double_complex& operator*=(const double_complex& rhs)	乘 rhs，结果保存到 *this。
	double_complex& operator/=(const double_complex& rhs)	除 rhs，结果保存到 *this。

double_complex::double_complex(double re = 0.0, double im = 0.0)

这是类 double_complex 的构造函数。

用以下值进行初始化：

_re = re;

_im = im;

double_complex::double_complex(const float_complex&)

这是类 double_complex 的构造函数。

用以下值进行初始化：

_re = (double)rhs.real();

_im = (double)rhs.imag();

double double_complex::real() const

求实部。

返回值为 this->_re。

double double_complex::imag() const

求虚部。

返回值为 this->_im。

double_complex& double_complex::operator=(double rhs)

将 rhs 复制到实部 （_re），虚部 （_im）设定为 0.0。

返回值为 *this。

double_complex& double_complex::operator+=(double rhs)

实部 （_re）加上 rhs，结果保存到实部 （_re），虚部 （_im）的值不变。

返回值为 *this。

double_complex& double_complex::operator–=(double rhs)

实部 （_re）减去 rhs，结果保存到实部 （_re），虚部 （_im）的值不变。

返回值为 *this。

double_complex& double_complex::operator*=(double rhs)

乘 rhs，结果保存到 *this。

(_re=_re*rhs, _im=_im*rhs)

返回值为 *this。

double_complex& double_complex::operator/=(double rhs)

除 rhs，结果保存到 *this。

(_re=_re/rhs, _im=_im/rhs)

返回值为 *this。

double_complex& double_complex::operator=(const double_complex& rhs)

复制 rhs。

返回值为 *this。

double_complex& double_complex::operator+=(const double_complex& rhs)

加上 rhs，结果保存到 *this。

返回值为 *this。

double_complex& double_complex::operator–=(const double_complex& rhs)

减去 rhs，结果保存到 *this。

返回值为 *this。

double_complex& double_complex::operator*=(const double_complex& rhs)

乘 rhs，结果保存到 *this。

返回值为 *this。

double_complex& double_complex::operator/=(const double_complex& rhs)

除 rhs，结果保存到 *this。

返回值为 *this。

(d) double_complex 成员外函数

种类	定义名	说明
函数	double_complex operator+(const double_complex& lhs)	进行 lhs 的一元 + 运算。
	double_complex operator+(const double_complex& lhs, const double_complex& rhs)	lhs 加上 rhs 相，结果保存到 lhs。
	double_complex operator+(const double_complex& lhs, const double& rhs)	
	double_complex operator+(const double& lhs, const double_complex& rhs)	
	double_complex operator–(const double_complex& lhs)	进行 lhs 的一元 – 运算。
	double_complex operator–(const double_complex& lhs, const double_complex& rhs)	lhs 减去 rhs，结果保存到 lhs。
	double_complex operator–(const double_complex& lhs, const double& rhs)	
	double_complex operator–(const double& lhs, const double_complex& rhs)	
	double_complex operator*(const double_complex& lhs, const double_complex& rhs)	lhs 乘 rhs，结果保存到 lhs。
	double_complex operator*(const double_complex& lhs, const double& rhs)	
	double_complex operator*(const double& lhs, const double_complex& rhs)	
	double_complex operator/(const double_complex& lhs, const double_complex& rhs)	lhs 除 rhs，结果保存到 lhs。
	double_complex operator/(const double_complex& lhs, const double& rhs)	
	double_complex operator/(const double& lhs, const double_complex& rhs)	

种类	定义名	说明
函数	bool operator==(const double_complex& lhs, const double_complex& rhs)	将 lhs 和 rhs 的实部和实部、虚部和虚部进行比较。
	bool operator==(const double_complex& lhs, const double& rhs)	
	bool operator==(const double& lhs, const double_complex& rhs)	
	bool operator!=(const double_complex& lhs, const double_complex& rhs)	将 lhs 和 rhs 的实部和实部、虚部和虚部进行比较。
	bool operator!=(const double_complex& lhs, const double& rhs)	
	bool operator!=(const double& lhs, const double_complex& rhs)	
	istream& operator>>(istream& is, double_complex& x)	以 u、(u) 或者 (u,v)（u：实部、v：虚部）格式输入 x。
	ostream& operator<<(ostream& os, const double_complex& x)	以 u、(u) 或者 (u,v)（u：实部、v：虚部）格式输出 x。
	double real(const double_complex& x)	求实部。
	double imag(const double_complex& x)	求虚部。
	double abs(const double_comlex& x)	求绝对值。
	double arg(const double_complex& x)	求相位角。
	double norm(const double_complex& x)	求平方的绝对值。
	double_complex conj(const double_complex& x)	求共轭复数。
	double_complex polar(const double& rho, const double& theta)	求对应大小为 rho 并且相位角为 theta 的复数的 double_complex 值。
	double_complex cos(const double_complex& x)	求复数的余弦。
	double_complex cosh(const double_complex& x)	求复数的双曲余弦。
	double_complex exp(const double_complex&)	求指数函数。

种类	定义名	说明
函数	double_complex log(const double_complex& x)	求自然对数。
	double_complex log10(const double_complex& x)	求常用对数。
	double_complex pow(const double _complex& x, int y)	求 x 的 y 乘方。
	double_complex pow(const double _complex& x, const double & y)	
	double_complex pow(const double _complex& x, const double _complex& y)	
	double_complex pow(const double & x, const double _complex& y)	
	double_complex sin(const double_complex& x)	求复数的正弦。
	double_complex sinh(const double_complex& x)	求复数的双曲正弦。
	double_complex sqrt(const double_complex& x)	求右半空间范围内的平方根。
	double_complex tan(const double_complex& x)	求复数的正切。
	double_complex tanh(const double_complex& x)	求复数的双曲正切。

double_complex operator+(const double_complex& lhs)

进行 lhs 的一元 + 运算。

返回值为 lhs。

double_complex operator+(const double_complex& lhs, const double_complex& rhs)
double_complex operator+(const double_complex& lhs, const double& rhs)
double_complex operator+(const double& lhs, const double_complex& rhs)

lhs 加上 rhs，结果保存到 lhs。

返回值为 double_complex(lhs)+=rhs。

double_complex operator–(const double_complex& lhs)

进行 lhs 的一元 – 运算。

返回值为 double_complex(-lhs.real(), -lhs.imag())。

double_complex operator–(const double_complex& lhs, const double_complex& rhs)
double_complex operator–(const double_complex& lhs, const double& rhs)
double_complex operator–(const double& lhs, const double_complex& rhs)

lhs 减去 rhs，结果保存到 lhs。

返回值为 double_complex(lhs)-=rhs。

double_complex operator*(const double_complex& lhs, const double_complex& rhs)
double_complex operator*(const double_complex& lhs, const double& rhs)
double_complex operator*(const double& lhs, const double_complex& rhs)

lhs 乘 rhs，结果保存到 lhs。
返回值为 double_complex(lhs)*=rhs。

double_complex operator/(const double_complex& lhs, const double_complex& rhs)
double_complex operator/(const double_complex& lhs, const double& rhs)
double_complex operator/(const double& lhs, const double_complex& rhs)

lhs 除 rhs，结果保存到 lhs。
返回值为 double_complex(lhs)/=rhs。

bool operator==(const double_complex& lhs, const double_complex& rhs)
bool operator==(const double_complex& lhs, const double& rhs)
bool operator==(const double& lhs, const double_complex& rhs)

将 lhs 和 rhs 的实部和实部、虚部和虚部进行比较。在 double 型参数的情况下，虚部假设为 double 型的 0.0。
返回值为 lhs.real()==rhs.real() && lhs.imag()==rhs.imag()。

bool operator!=(const double_complex& lhs, const double_complex& rhs)
bool operator!=(const double_complex& lhs, const double& rhs)
bool operator!=(const double& lhs, const double_complex& rhs)

将 lhs 和 rhs 的实部和实部、虚部和虚部进行比较。在 double 型参数的情况下，虚部假设为 double 型的 0.0。
返回值为 lhs.real()!=rhs.real() || lhs.imag()!=rhs.imag()。

istream& operator>>(istream& is, double_complex& x)

以 u、 (u) 或者 (u,v) （u 为实部， v 为虚部）格式输入复数 x。输入值被转换为 double_complex。
如果不以 u、 (u) 或者 (u,v) 格式输入 x，就调用 is.setstate(ios_base::failbit)。
返回值为 is。

ostream& operator<<(ostream& os, const double_complex& x)

将 x 输出到 os。
输出格式为 u、 (u) 或者 (u,v) （u 为实部， v 为虚部）。
返回值为 os。

double real(const double_complex& x)

求实部。
返回值为 x.real()。

double imag(const double_complex& x)

求虚部。
返回值为 x.imag()。

double abs(const double_complex& x)

求绝对值。
返回值为 $(|x.real()|^2 + |x.imag()|^2)^{1/2}$。

double arg(const double_complex& x)

求相位角。

返回值为 atan2(x.imag() , x.real())。

double norm(const double_complex& x)

求平方的绝对值。

返回值为 $|x.real()|^2 + |x.imag()|^2$。

double_complex conj(const double_complex& x)

求共轭复数。

返回值为 double_complex(x.real(), (–1)*x.imag())。

double_complex polar(const double& rho, const double& theta)

求对应大小为 rho 并且相位角 （辐角）为 theta 的复数的 double_complex 值。

返回值为 double_complex(rho*cos(theta), rho*sin(theta))。

double_complex cos(const double_complex& x)

求复数的余弦。

返回值为 double_complex(cos(x.real())*cosh(x.imag()), (–1)*sin(x.real())*sinh(x.imag()))。

double_complex cosh(const double_complex& x)

求复数的双曲余弦。

返回值为 cos(double_complex((–1)*x.imag(), x.real()))。

double_complex exp(const double_complex& x)

求指数函数。

返回值为 exp(x.real())*cos(x.imag()),exp(x.real())*sin(x.imag())。

double_complex log(const double_complex& x)

求以 e 为底的自然对数。

返回值为 double_complex(log(abs(x)), arg(x))。

double_complex log10(const double_complex& x)

求以 10 为底的常用对数。

返回值为 double_complex(log10(abs(x)), arg(x)/log(10))。

double_complex pow(const double_complex& x, int y)
double_complex pow(const double_complex& x, const double& y)
double_complex pow(const double_complex& x, const double_complex& y)
double_complex pow(const double& x, const double_complex& y)

求 x 的 y 乘方。

当 pow(0,0) 时，为定义域错误。

返回值为 exp(y*log(x))。

double_complex sin(const double_complex& x)

求复数的正弦。

返回值为 double_complex(sin(x.real())*cosh(x.imag()), cos(x.real())*sinh(x.imag()))。

double_complex sinh(const double_complex& x)

求复数的双曲正弦。

返回值为 double_complex(0,–1)*sin(double_complex((–1)*x.imag(),x.real()))。

double_complex sqrt(const double_complex& x)

求右半空间范围内的平方根。

返回值为 double_complex(sqrt(abs(x))*cos(arg(x)/2), sqrt(abs(x))*sin(arg(x)/2))。

double_complex tan(const double_complex& x)

求复数的正切。

返回值为 sin(x)/cos(x)。

double_complex tanh(const double_complex& x)

求复数的双曲正切。

返回值为 sinh(x)/cosh(x)。

(5) 字符串操作类库

字符串操作类库对应的头文件如下：

- <string>

  定义string类。

本类没有派生关系。

(a) string 类

种类	定义名	说明
型	iterator	这是 char* 型。
	const_iterator	这是 const char* 型。
常数	npos	这是字符串的最大长度 （UINT_MAX 字符）。
变量	s_ptr	这是指向目标保存字符串的区域的指针。
	s_len	这是目标保存的字符串长度。
	s_res	这是确保目标保存字符串的区域容量。
函数	string(void)	这是构造函数。
	string::string(const string& str, size_t pos = 0, size_t n = npos)	
	string::string(const char* str, size_t n)	
	string::string(const char* str)	
	string::string(size_t n, char c)	
	~string()	这是析构函数。
	string& operator=(const string& str)	赋值 str。
	string& operator=(const char* str)	
	string& operator=(char c)	赋值 c。
	iterator begin()	求字符串的头指针。
	const_iterator begin() const	
	iterator end()	求字符串的尾指针。
	const_iterator end() const	
	size_t size() const	求保存的字符串的长度。
	size_t length() const	
	size_t max_size() const	求确保的区域容量。
	void resize(size_t n, char c)	将能保存的字符串的字符数更改为 n。
	void resize(size_t n)	将能保存的字符串的字符数更改为 n。
	size_t capacity() const	求确保的区域容量。
	void reserve(size_t res_arg = 0)	重新分配区域。
	void clear()	清除保存的字符串。
	bool empty() const	检查保存的字符串的字符数是否为 0。
	const char& operator[](size_t pos) const	参照 s_ptr[pos]。
	char& operator[](size_t pos)	
	const char& at(size_t pos) const	
	char& at(size_t pos)	

种类	定义名	说明
函数	string& operator+=(const string& str)	追加 str 字符串。
	string& operator+=(const char* str)	
	string& operator+=(char c)	追加 c 字符。
	string& append(const string& str)	追加 str 字符串。
	string& append(const char* str)	
	string& append(const string& str, size_t pos, size_t n)	给目标位置 pos 追加 str 字符串的 n 个字符。
	string& append(const char* str, size_t n)	追加字符串 str 的 n 个字符。
	string& append(size_t n, char c)	追加 n 个字符 c。
	string& assign(const string& str)	赋值 str 字符串。
	string& assign(const char* str)	
	string& assign(const string& str, size_t pos, size_t n)	将字符串 str 的 n 个字符赋值到位置 pos。
	string& assign(const char* str, size_t n)	赋值字符串 str 的 n 个字符。
	string& assign(size_t n, char c)	赋值 n 个字符 c。
	string& insert(size_t pos1, const string& str)	将 str 字符串插入到位置 pos1。
	string& insert(size_t pos1, const string& str, size_t pos2, size_t n)	将从 str 字符串的位置 pos2 开始的 n 个字符插入到位置 pos1。
	string& insert(size_t pos, const char* str, size_t n)	将字符串 str 的 n 个字符插入到 pos 位置。
	string& insert(size_t pos, const char* str)	将字符串 str 插入到 pos 位置。
	string& insert(size_t pos, size_t n, char c)	将 n 个字符 c 的字符串插入到位置 pos。
	iterator insert(iterator p, char c = char())	将字符 c 插入到 p 指向的字符串之前。
	void insert(iterator p, size_t n, char c)	将 n 个字符 c 插入到 p 指向的字符之前。
	string& erase(size_t pos = 0, size_t n = npos)	从位置 pos 删除 n 个字符。
	iterator erase(iterator position)	删除由 position 参照的字符。
	iterator erase(iterator first, iterator last)	删除范围 [first, last] 内的字符。
	string& replace(size_t pos1, size_t n1, const string& str)	用 str 字符串替换从位置 pos1 开始的 n1 个字符的字符串。
	string& replace(size_t pos1, size_t n1, const char* str)	

种类	定义名	说明
函数	string& replace(size_t pos1, size_t n1, const string& str, size_t pos2, size_t n2)	用从 str 的位置 pos2 开始的 n2 个字符的字符串替换从位置 pos1 开始的 n1 个字符的字符串。
	string& replace(size_t pos, size_t n1, const char* str, size_t n2)	用 str 字符串的 n2 个字符替换从位置 pos 开始的 n1 个字符的字符串。
	string& replace(size_t pos, size_t n1, size_t n2, char c)	用 n2 个字符 c 替换从位置 pos 开始的 n1 个字符的字符串。
	string& replace(iterator i1, iterator i2, const string& str)	用 str 字符串替换从位置 i1 到 i2 的字符串。
	string& replace(iterator i1, iterator i2, const char* str)	
	string& replace(iterator i1, iterator i2, const char* str, size_t n)	用 str 字符串的 n 个字符替换从位置 i1 到 i2 的字符串。
	string& replace(iterator i1, iterator i2, size_t n, char c)	用 n 个字符 c 替换从位置 i1 到 i2 的字符串。
	size_t copy(char* str, size_t n, size_t pos = 0) const	将字符串 str 的 n 个字符的字符串复制到位置 pos。
	void swap(string& str)	和 str 字符串进行交换。
	const char* c_str() const	参照指向保存字符串的区域的指针。
	const char* data() const	

种类	定义名	说明
函数	size_t find(const string& str, size_t pos = 0) const	检索在位置 pos 以后和 str 字符串相同的字符串最初出现的位置。
	size_t find(const char* str, size_t pos = 0) const	
	size_t find(const char* str, size_t pos, size_t n) const	检索在位置 pos 以后和 str 的 n 个字符相同的字符串最初出现的位置。
	size_t find(char c, size_t pos = 0) const	检索在位置 pos 以后字符 c 最初出现的位置。
	size_t rfind(const string& str, size_t pos = npos) const	检索在位置 pos 以前和 str 字符串相同的字符串最后出现的位置。
	size_t rfind(const char* str, size_t pos = npos) const	
	size_t rfind(const char* str, size_t pos, size_t n) const	检索在位置 pos 以前和 str 个 n 个字符相同的字符串最后出现的位置。
	size_t rfind(char c, size_t pos = npos) const	检索在位置 pos 以前字符 c 最后出现的位置。
	size_t find_first_of(const string& str, size_t pos = 0) const	检索在位置 pos 以后字符串 str 中的任意字符最初出现的位置。
	size_t find_first_of(const char* str, size_t pos = 0) const	
	size_t find_first_of(const char* str, size_t pos, size_t n) const	检索在位置 pos 以后字符串 str 的 n 个字符中的任意字符最初出现的位置。
	size_t find_first_of(char c, size_t pos = 0) const	检索在位置 pos 以后字符 c 最初出现的位置。
	size_t find_last_of(const string& str, size_t pos = npos) const	检索在位置 pos 以前字符串 str 中的任意字符最后出现的位置。
	size_t find_last_of(const char* str, size_t pos = npos) const	
	size_t find_last_of(const char* str, size_t pos, size_t n) const	检索在位置 pos 以前字符串 str 的 n 个字符中的任意字符最后出现的位置。
	size_t find_last_of(char c, size_t pos = npos) const	检索在位置 pos 以前字符 c 最后出现的位置。

种类	定义名	说明
函数	size_t find_first_not_of(const string& str, size_t pos = 0) const	检索在位置 pos 以后和 str 中的任意字符不同的字符最初出现的位置。
	size_t find_first_not_of(const char* str, size_t pos = 0) const	
	size_t find_first_not_of(const char* str, size_t pos, size_t n)	检索在位置 pos 以后和 str 中的前 n 个任意字符不同的字符最初出现的位置。
	size_t find_first_not_of(char c, size_t pos = 0) const	检索在位置 pos 以后和字符 c 不同的字符最初出现的位置。
	size_t find_last_not_of(const string& str, size_t pos = npos) const	检索在位置 pos 以前和 str 中的任意字符不同的的字符最后出现的位置。
	size_t find_last_not_of(const char* str, size_t pos = npos) const	
	size_t find_last_not_of(const char* str, size_t pos, size_t n) const	检索在位置 pos 以前和 str 中的前 n 个任意字符不同的字符最后出现的位置。
	size_t find_last_not_of(char c, size_t pos = npos) const	检索在位置 pos 以前和字符 c 不同的字符最后出现的位置。
	string substr(size_t pos = 0, size_t n = npos) const	对于保存的字符串，生成有范围 [pos,n] 的字符串的目标。
	int compare(const string& str) const	将字符串和 str 字符串进行比较。
	int compare(size_t pos1, size_t n1, const string& str) const	将从位置 pos1 开始的 n1 个字符的字符串和 str 进行比较。
	int compare(size_t pos1, size_t n1, const string& str, size_t pos2, size_t n2) const	将从位置 pos1 开始的 n1 个字符的字符串以及从 str 的位置 pos2 开始的 n2 个字符的字符串进行比较。
	int compare(const char* str) const	和 str 进行比较。
	int compare(size_t pos1, size_t n1, const char* str, size_t n2 = npos) const	将从位置 pos1 开始的 n1 个字符的字符串和 str 的 n2 个字符的字符串进行比较。

string::string(void)

进行以下设定：

s_ptr = 0;

s_len = 0;

s_res = 1;

string::string(const string& str, size_t pos = 0, size_t n = npos)

复制 str。但是， s_len 为 n 和 s_len 中较小的值。

string::string(const char* str, size_t n)

进行以下设定：

s_ptr = str;

s_len = n;

s_res = n+1;

string::string(const char* str)

进行以下设定：

s_ptr = str;

s_len = str 的字符串长度 ;

s_res = str 的字符串长度 +1;

string::string(size_t n, char c)

进行以下设定：

s_ptr = n 个字符 c 的字符串 ;

s_len = n;

s_res = n+1;

string::~string()

这是类 string 的析构函数。

释放保存字符串的区域。

string& string::operator=(const string& str)

赋值 str 的数据。

返回值为 *this。

string& string::operator=(const char* str)

从 str 生成 string 目标，赋值该数据。

返回值为 *this。

string& string::operator=(char c)

从 c 生成 string 目标，赋值该数据。

返回值为 *this。

string::iterator string::begin()

string::const_iterator string::begin() const

求字符串的头指针。

返回值为字符串的头指针。

string::iterator string::end()
string::const_iterator string::end() const

求字符串的尾指针。
返回值为字符串的尾指针。

size_t string::size() const
size_t string::length() const

求保存的字符串的长度。
返回值为保存的字符串的长度。

size_t string::max_size() const

求确保的区域容量。
返回值为确保的区域容量。

void string::resize(size_t n, char c)

将目标能保存的符串的字符数更改为 n。
当 n<=size() 时，和长度为 n 的原字符串进行替换。
当 n>size() 时，和在原字符串之后填入 c （使其长度为 n）的字符串进行替换。
需要 n<=max_size()。
当 n>max_size() 时，作为 n=max_size() 进行计算。

void string::resize(size_t n)

将目标能保存的字符串的字符数更改为 n。
当 n<=size() 时，和长度为 n 的原字符串进行替换。
需要 n<=max_size()。

size_t string::capacity() const

求确保的区域容量。
返回值为确保的区域容量。

void string::reserve(size_t res_arg = 0)

重新分配存储区。
在 reserve() 后， capacity() 大于等于 reserve() 的参数。
如果重新进行分配，就将全部的参照、指针以及参照此数列中的元素的 iterator 设定为无效。

void string::clear()

清除保存的字符串。

bool string::empty() const

检查保存的字符串的字符数是否为 0。
返回值如下：

当保存的字符串长度为 0 时 ：true
当保存的字符串长度不为 0 时 ：false

const char& string::operator[](size_t pos) const
char& string::operator[](size_t pos)
const char& string::at(size_t pos) const
char& string::at(size_t pos)

参照 s_ptr[pos]。
返回值如下：

当 n<s_len 时 ：s_ptr [pos]
当 n>=s_len 时 ：'\0'

string& string::operator+=(const string& str)

追加 str 保存的字符串。
返回值为 *this。

string& string::operator+=(const char* str)

从 str 生成 string 目标，追加该字符串。
返回值为 *this。

string& string::operator+=(char c)

从 c 生成 string 目标，追加该字符串。
返回值为 *this。

string& string::append(const string& str)
string& string::append(const char* str)

将 str 字符串追加到目标。
返回值为 *this。

string& string::append(const string& str, size_t pos, size_t n)

将 str 字符串的 n 个字符追加到目标的位置 pos。
返回值为 *this。

string& string::append(const char* str, size_t n)

追加字符串 str 的 n 个字符。
返回值为 *this。

string& string::append(size_t n, char c)

追加 n 个字符 c。
返回值为 *this。

string& string::assign(const string& str)
string& string::assign(const char* str)

赋值 str 字符串。
返回值为 *this。

string& string::assign(const string& str, size_t pos, size_t n)

将字符串 str 的 n 个字符赋值到位置 pos。
返回值为 *this。

string& string::assign(const char* str, size_t n)

赋值字符串 str 的 n 个字符。

返回值为 *this。

string& string::assign(size_t n, char c)

赋值 n 个字符 c。

返回值为 *this。

string& string::insert(size_t pos1, const string& str)

将 str 字符串插入到位置 pos1。

返回值为 *this。

string& string::insert(size_t pos1, const string& str, size_t pos2, size_t n)

将从 str 字符串的位置 pos2 开始的 n 个字符插入到位置 pos1。

返回值为 *this。

string& string::insert(size_t pos, const char* str, size_t n)

将字符串 str 的 n 个字符插入到位置 pos。

返回值为 *this。

string& string::insert(size_t pos, const char* str)

将字符串 str 插入到位置 pos。

返回值为 *this。

string& string::insert(size_t pos, size_t n, char c)

将 n 个字符 c 的字符串插入到位置 pos。

返回值为 *this。

string::iterator string::insert(iterator p, char c = char())

将字符 c 插入到 p 指向的字符串之前。

返回值为插入的字符。

void string::insert(iterator p, size_t n, char c)

将 n 个字符 c 插入到 p 指向的字符之前。

string& string::erase(size_t pos = 0, size_t n = npos)

从位置 pos 删除 n 个字符。

返回值为 *this。

iterator string::erase(iterator position)

删除由 position 参照的字符。

返回值如下：

当有删除元素的下一个 iterator 时 ： 删除元素的下一个 iterator

当没有删除元素的下一个 iterator 时 ： end()

iterator string::erase(iterator first, iterator last)

删除范围 [first, last] 内的字符。

返回值如下：

当有 last 的下一个 iterator 时 ：last 的下一个 iterator

当没有 last 的下一个 iterator 时 ：end()

string& string::replace(size_t pos1, size_t n1, const string& str)

string& string::replace(size_t pos1, size_t n1, const char* str)

用 str 字符串替换从位置 pos1 开始的 n1 个字符的字符串。

返回值为 *this。

string& string::replace(size_t pos1, size_t n1, const string& str, size_t pos2, size_t n2)

用从 str 的位置 pos2 开始的 n2 个字符的字符串替换从位置 pos1 开始的 n1 字符的字符串。

返回值为 *this。

string& string::replace(size_t pos, size_t n1, const char* str, size_t n2)

用 str 字符的 n2 个字符串替换从位置 pos 开始的 n1 个字符的字符串。

返回值为 *this。

string& string::replace(size_t pos, size_t n1, size_t n2, char c)

用 n2 个字符 c 替换从位置 pos 开始的 n1 个字符的字符串。

返回值为 *this。

string& string::replace(iterator i1, iterator i2, const string& str)

string& string::replace(iterator i1, iterator i2, const char* str)

用 str 字符串替换从位置 i1 到 i2 的字符串。

返回值为 *this。

string& string::replace(iterator i1, iterator i2, const char* str, size_t n)

用 str 的 n 个字符的字符串替换从位置 i1 到 i2 的字符串。

返回值为 *this。

string& string::replace(iterator i1, iterator i2, size_t n, char c)

用 n 个字符 c 替换从位置 i1 到 i2 的字符串。

返回值为 *this。

size_t string::copy(char* str, size_t n, size_t pos = 0) const

将字符串 str 的 n 个字符的字符串复制到位置 pos。

返回值为 rlen。

void string::swap(string& str)

和 str 字符串进行交换。

const char* string::c_str() const

const char* string::data() const

参照指向保存字符串的区域的指针。

返回值为 s_ptr。

size_t string::find(const string& str, size_t pos = 0) const
size_t string::find(const char* str, size_t pos = 0) const
检索在位置 pos 以后和 str 字符串相同的字符串最初出现的位置。
返回值为字符串的偏移量。

size_t string::find(const char* str, size_t pos, size_t n) const
检索在位置 pos 以后和 str 的 n 个字符相同的字符串最初出现的位置。
返回值为字符串的偏移量。

size_t string::find(char c, size_t pos = 0) const
检索在位置 pos 以后字符 c 最初出现的位置。
返回值为字符串的偏移量。

size_t string::rfind(const string& str, size_t pos = npos) const
size_t string::rfind(const char* str, size_t pos = npos) const
检索在位置 pos 以前和 str 字符串相同的字符串最后出现的位置。
返回值为字符串的偏移量。

size_t string::rfind(const char* str, size_t pos, size_t n) const
检索在位置 pos 以前和 str 的 n 个字符相同的字符串最后出现的位置。
返回值为字符串的偏移量。

size_t string::rfind(char c, size_t pos = npos) const
检索在位置 pos 以前字符 c 最后出现的位置。
返回值为字符串的偏移量。

size_t string::find_first_of(const string& str, size_t pos = 0) const
size_t string::find_first_of(const char* str, size_t pos = 0) const
检索在位置 pos 以后字符串 str 中的任意字符最初出现的位置。
返回值为字符串的偏移量。

size_t string::find_first_of(const char* str, size_t pos, size_t n) const
检索在位置 pos 以后字符串 str 的 n 个字符中的任意字符最初出现的位置。
返回值为字符串的偏移量。

size_t string::find_first_of(char c, size_t pos = 0) const
检索在位置 pos 以后字符 c 最初出现的位置。
返回值为字符串的偏移量。

size_t string::find_last_of(const string& str, size_t pos = npos) const
size_t string::find_last_of(const char* str, size_t pos = npos) const
检索在位置 pos 以前字符串 str 中的任意字符最后出现的位置。
返回值为字符串的偏移量。

size_t string::find_last_of(const char* str, size_t pos, size_t n) const
检索在位置 pos 以前字符串 str 的 n 个字符中的任意字符最后出现的位置。
返回值为字符串的偏移量。

size_t string::find_last_of(char c, size_t pos = npos) const

检索在位置 pos 以前字符 c 最后出现的位置。

返回值为字符串的偏移量。

size_t string::find_first_not_of(const string& str, size_t pos = 0) const
size_t string::find_first_not_of(const char* str, size_t pos = 0) const

检索在位置 pos 以后和 str 中的任意字符不同的字符最初出现的位置。

返回值为字符串的偏移量。

size_t string::find_first_not_of(const char* str, size_t pos, size_t n) const

检索在位置 pos 以后和 str 中的前 n 个任意字符不同的字符最初出现的位置。

返回值为字符串的偏移量。

size_t string::find_first_not_of(char c, size_t pos = 0) const

检索在位置 pos 以后和字符 c 不同的字符最初出现的位置。

返回值为字符串的偏移量。

size_t string::find_last_not_of(const string& str, size_t pos = npos) const
size_t string::find_last_not_of(const char* str, size_t pos = npos) const

检索在位置 pos 以前和 str 中的任意字符不同的字符最后出现的位置。

返回值为字符串的偏移量。

size_t string::find_last_not_of(const char* str, size_t pos, size_t n) const

检索在位置 pos 以前和 str 中的前 n 个任意字符不同的字符最后出现的位置。

返回值为字符串的偏移量。

size_t string::find_last_not_of(char c, size_t pos = npos) const

检索在位置 pos 以前和字符 c 不同的字符最后出现的位置。

返回值为字符串的偏移量。

string string::substr(size_t pos = 0, size_t n = npos) const

对于保存的字符串，生成有范围 [pos,n] 的字符串的目标。

返回值为有范围 [pos,n] 的字符串的目标。

int string::compare(const string& str) const

将字符串和 str 字符串进行比较。

返回值如下：

当字符串相同时 ：0

当字符串不同时 ：this->s_len>str.s_len 时为 1

this->s_len < str.s_len 时为 –1

int string::compare(size_t pos1, size_t n1, const string& str) const

将从位置 pos1 开始的 n1 个字符的字符串和 str 进行比较。

返回值如下：

当字符串相同时 ：0

当字符串不同时 ：this->s_len>str.s_len 时为 1

this->s_len < str.s_len 时为 –1

int string::compare(size_t pos1, size_t n1, const string& str, size_t pos2, size_t n2) const

将从位置 pos1 开始的 n1 个字符的字符串和从 str 的位置 pos2 开始的 n2 个字符的字符串进行比较。

返回值如下：

当字符串相同时 ：0

当字符串不同时 ：this->s_len>str.s_len 时为 1

this->s_len < str.s_len 时为 –1

int string::compare(const char* str) const

和 str 进行比较。

返回值如下：

当字符串相同时 ：0

当字符串不同时 ：this->s_len>str.s_len 时为 1

this->s_len < str.s_len 时为 –1

int string::compare(size_t pos1, size_t n1, const char* str, size_t n2 = npos) const

将从位置 pos1 开始的 n1 个字符的字符串和 str 的 n2 个字符的字符串进行比较。

返回值如下：

当字符串相同时 ：0

当字符串不同时 ：this->s_len>str.s_len 时为 1

this->s_len < str.s_len 时为 –1

(b) string 类操纵符

种类	定义名	说明
函数	string operator+(const string& lhs, const string& rhs)	将 rhs 的字符串 （或者字符）追加到 lhs 的字符串 （或者字符），在生成目标后保存到字符串。
	string operator+(const char* lhs, const string& rhs)	
	string operator+(char lhs, const string& rhs)	
	string operator+(const string& lhs, const char* rhs)	
	string operator+(const string& lhs, char rhs)	
	bool operator==(const string& lhs, const string& rhs)	将 lhs 的字符串和 rhs 的字符串进行比较。
	bool operator==(const char* lhs, const string& rhs)	
	bool operator==(const string& lhs, const char* rhs)	
	bool operator!=(const string& lhs, const string& rhs)	将 lhs 的字符串和 rhs 的字符串进行比较。
	bool operator!=(const char* lhs, const string& rhs)	
	bool operator!=(const string& lhs, const char* rhs)	
	bool operator<(const string& lhs, const string& rhs)	将 lhs 的字符串长度和 rhs 的字符串长度进行比较。
	bool operator<(const char* lhs, const string& rhs)	
	bool operator<(const string& lhs, const char* rhs)	
	bool operator>(const string& lhs, const string& rhs)	将 lhs 的字符串长度和 rhs 的字符串长度进行比较。
	bool operator>(const char* lhs, const string& rhs)	
	bool operator>(const string& lhs, const char* rhs)	
	bool operator<=(const string& lhs, const string& rhs)	将 lhs 的字符串长度和 rhs 的字符串长度进行比较。
	bool operator<=(const char* lhs, const string& rhs)	
	bool operator<=(const string& lhs, const char* rhs)	
	bool operator>=(const string& lhs, const string& rhs)	将 lhs 的字符串长度和 rhs 的字符串长度进行比较。
	bool operator>=(const char* lhs, const string& rhs)	
	bool operator>=(const string& lhs, const char* rhs)	
	void swap(string& lhs, string& rhs)	将 lhs 的字符串和 rhs 的字符串进行交换。
	istream& operator>>(istream& is, string& str)	将字符串抽出到 str。
	ostream& operator<<(ostream& os, const string& str)	插入字符串。
	istream& getline(istream& is, string& str, char delim)	从 is 抽出字符串，附加到 str。如果在中途检测到字符 'delim'，就结束输入。
	istream& getline(istream& is, string& str)	从 is 抽出字符串，附加到 str。如果在中途检测到换行字符，就结束输入。

string operator+(const string& lhs, const string& rhs)
string operator+(const char* lhs, const string& rhs)
string operator+(char lhs, const string& rhs)
string operator+(const string& lhs, const char* rhs)
string operator+(const string& lhs, char rhs)

将 rhs 的字符串 （或者字符）追加到 lhs 的字符串 （或者字符），在生成目标后保存该字符串。
返回值为保存连接后的字符串的目标。

bool operator==(const string& lhs, const string& rhs)
bool operator==(const char* lhs, const string& rhs)
bool operator==(const string& lhs, const char* rhs)

将 lhs 的字符串和 rhs 的字符串进行比较。
返回值如下：

当字符串相同时	：true
当字符串不同时	：false

bool operator!=(const string& lhs, const string& rhs)
bool operator!=(const char* lhs, const string& rhs)
bool operator!=(const string& lhs, const char* rhs)

将 lhs 的字符串和 rhs 的字符串进行比较。
返回值如下：

当字符串相同时	：false
当字符串不同时	：true

bool operator<(const string& lhs, const string& rhs)
bool operator<(const char* lhs, const string& rhs)
bool operator<(const string& lhs, const char* rhs)

将 lhs 的字符串长度和 rhs 的字符串长度进行比较。
返回值如下：

当 lhs.s_len < rhs.s_len 时	：true
当 lhs.s_len >= rhs.s_len 时	：false

bool operator>(const string& lhs, const string& rhs)
bool operator>(const char* lhs, const string& rhs)
bool operator>(const string& lhs, const char* rhs)

将 lhs 的字符串长度和 rhs 的字符串长度进行比较。
返回值如下：

当 lhs.s_len > rhs.s_len 时	：true
当 lhs.s_len <= rhs.s_len 时	：false

bool operator<=(const string& lhs, const string& rhs)
bool operator<=(const char* lhs, const string& rhs)
bool operator<=(const string& lhs, const char* rhs)

将 lhs 的字符串长度和 rhs 的字符串长度进行比较。
返回值如下：

当 lhs.s_len <= rhs.s_len 时	：true
当 lhs.s_len > rhs.s_len 时	：false

bool operator>=(const string& lhs, const string& rhs)
bool operator>=(const char* lhs, const string& rhs)
bool operator>=(const string& lhs, const char* rhs)

将 lhs 的字符串长度和 rhs 的字符串长度进行比较。
返回值如下：

当 lhs.s_len >= rhs.s_len 时	：true
当 lhs.s_len < rhs.s_len 时	：false

void swap(string& lhs, string& rhs)

将 lhs 的字符串和 rhs 的字符串进行交换。

istream& operator>>(istream& is, string& str)

将字符串抽出到 str。
返回值为 is。

ostream& operator<<(ostream& os, const string& str)

插入字符串。
返回值为 os。

istream& getline(istream& is, string& str, char delim)

从 is 抽出字符串，附加到 str。
如果在中途检测到字符 'delim'，就结束输入。
返回值为 is。

istream& getline(istream& is, string& str)

从 is 抽出字符串，附加到 str。
如果在中途检测到换行字符，就结束输入。
返回值为 is。

9.3.3 可重入库

除 rand、 srand 函数以外，用标准库生成工具指定 reent 选项而建立的库全部可重入。

没有指定 reent 选项时的可重入库一览表如**表 9.39** 所示。在表中，用△表示的函数设定 errno 变量，如果在程序中不参照 errno，就能执行可重入。

可重入栏 ○：可重入 ×：不可重入 △：设定 errno

表 9.39 可重入库一览表

标准 include 文件	函数名	可重入
stddef.h	offsetof	○
assert.h	assert	×
ctype.h	isalnum	○
	isalpha	○
	iscntrl	○
	isdigit	○
	isgraph	○
	islower	○
	isprint	○
	ispunct	○
	isspace	○
	isupper	○
	isxdigit	○
	tolower	○
	toupper	○
math.h	acos	△
	asin	△
	atan	△
	atan2	△
	cos	△
	sin	△
	tan	△
	cosh	△
	sinh	△
	tanh	△
	exp	△

标准 include 文件	函数名	可重入
math.h	frexp	△
	ldexp	△
	log	△
	log10	△
	modf	△
	pow	△
	sqrt	△
	ceil	△
	fabs	△
	floor	△
	fmod	△
mathf.h	acosf	△
	asinf	△
	atanf	△
	atan2f	△
	cosf	△
	sinf	△
	tanf	△
	coshf	△
	sinhf	△
	tanhf	△
	expf	△
	frexpf	△
	ldexpf	△
	logf	△
	log10f	△

标准 include 文件	函数名	可重入
mathf.h	modff	△
	powf	△
	sqrtf	△
	ceilf	△
	fabsf	△
	floorf	△
	fmodf	△
setjmp.h	setjmp	○
	longjmp	○
stdarg.h	va_start	○
	va_arg	○
	va_end	○
stdio.h	fclose	×
	fflush	×
	fopen	×
	freopen	×
	setbuf	×
	setvbuf	×
	fprintf	×
	fscanf	×
	printf	×
	scanf	×
	sprintf	△
	sscanf	△
	vfprintf	×
	vprintf	×
	vsprintf	△
	fgetc	×
	fgets	×
	fputc	×
	fputs	×
	getc	×
	getchar	×

标准 include 文件	函数名	可重入
stdio.h	gets	×
	putc	×
	putchar	×
	puts	×
	ungetc	×
	fread	×
	fwrite	×
	fseek	×
	ftell	×
	rewind	×
	clearerr	×
	feof	×
	ferror	×
	perror	×
stdlib.h	atof	△
	atoi	△
	atol	△
	atoll	△
	strtod	△
	strtol	△
	strtoul	△
	strtoll	△
	strtoull	△
	rand	×
	srand	×
	calloc	×
	free	×
	malloc	×
	realloc	×
	bsearch	○
	qsort	○
	abs	○
	div	△

标准 include 文件	函数名	可重入
string.h	labs	○
	llabs	○
	ldiv	△
	lldiv	△
	memcpy	○
	strcpy	○
	strncpy	○
	strcat	○
	strncat	○
	memcmp	○
	strcmp	○
	strncmp	○

标准 include 文件	函数名	可重入
string.h	memchr	○
	strchr	○
	strcspn	○
	strpbrk	○
	strrchr	○
	strspn	○
	strstr	○
	strtok	×
	memset	○
	strerror	○
	strlen	○
	memmove	○

9.3.4 不支持的库

在 C 语言规格定义的库中，本编译程序不支持的库如表 9.40 所示。

表 9.40 不支持的库

	头文件	库名
1	locale.h*1	setlocale、 localeconv
2	signal.h*1	signal、 raise
3	stdio.h	remove、 rename、 tmpfile、 tmpnam、 fgetpos、 fsetpos
4	stdlib.h	abort、 atexit、 exit、 _Exit、 getenv、 system、 mblen、 mbtowc、 wctomb、 mbstowcs、 wcstombs
5	string.h	strcoll、 strxfrm
6	time.h*1	clock、 difftime、 mktime、 time、 asctime、 ctime、 gmtime、 localtime、 strftime
7	wctype.h	iswalnum、 iswalpha、 iswblank、 iswcntrl、 iswdigit、 iswgraph、 iswlower、 iswprintf、 iswpunct、 iswspace、 iswupper、 iswxdigit、 iswctype、 wctype、 towlower、 towupper、 towctrans、 wctrans
8	wchar.h	wcsftime、 wcscoll、 wcsxfrm、 wctob、 mbrtowc、 wcrtomb、 mbsrtowcs、 wcsrtombs

【注】 *1 不支持头文件。

10. 汇编程序的语言规格

10.1 程序的记述方法

10.1.1 保留字

汇编程序将和汇编程序控制指令、助记符等相同的字符串作为保留字进行处理。因为保留字具有特殊功能，所以在汇编语言文件中不能用于标号名和符号名。保留字不分大小写字母，“ABS”和 “abs”是相同的保留字。

保留字有以下内容：

(1) 汇编程序控制指令

将汇编程序控制指令和以句点 （.）开始的字符串全部作为保留字。

(2) 助记符

将 RX 族的助记符全部作为保留字。

(3) 寄存器名和标志名

将 RX 族的寄存器名和标志名全部作为保留字。

(4) 运算符

将本章节说明的运算符全部作为保留字。

(5) 系统标号

汇编程序生成的以 2 个句点开始的名称称为系统标号，将系统标号全部作为保留字进行处理。

10.1.2 名称

能在汇编语言文件中任意定义和使用名称。

名称分为以下几种：

表 10.1 名称的种类

名称的种类	内容
标号名	这是以地址为值的名称。
符号名	这是以常数为值的名称。
段名	这是用 .SECTION 控制指令定义的段的名称。
定位符号名	表示记述了定位符号 ‘$’的行操作码的起始地址。
宏名	这是宏的定义名。

名称的记述规则

- 名称的字符个数没有限制。
- 名称区分大小写字母，将 “LAB”和 “Lab”作为不同的名称进行处理。
- 名称能使用英文字母、数字、下划线 （_）和美元符号 （$）。
- 名称的第一个字符不能使用数字。
- 名称不能使用保留字。

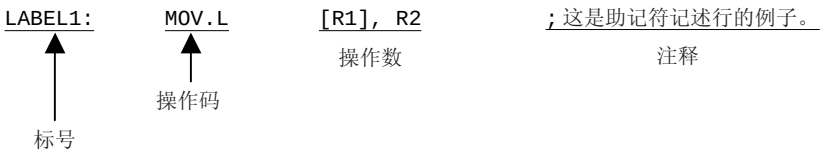
※但是，只有段名能使用保留字的标志名 （U、I、O、S、Z、C）。

10.1.3　助记符记述行的构成

助记符记述行的构成如下：

[标号][操作码 [△操作数]][注释]

（编程例子）



(1)　标号

给助记符记述行的地址定义名称。

(2)　操作码

操作码是助记符和控制指令。

(3)　操作数

操作数是操作的执行对象。操作数的个数和种类取决于操作码，有些操作码不需要操作数。

(4)　注释

这是为了方便理解程序的注释。

10.1.4　标号的记述方法

在记述标号时，必须在名称的最后附加冒号 （:）。

- 记述例子

  ```
  LABEL1:
  ```

10.1.5　操作码的记述方法

- 记述方法

  助记符[长度说明符(转移距离说明符)]

- 内容

  指令由以下2个元素构成：

  (1) 助记符　　・・・ 表示指令的操作。

  (2) 长度说明符　・・・ 指定处理对象的数据长度。

(1)　助记符

助记符表示指令的操作。

（例）

MOV　・・・ 传送指令

ADD　・・・ 算术运算指令 （加法运算指令）

(2) 长度说明符

长度说明符指定指令码的操作数长度。

- 记述方法
  `.size`
- 内容
  指定操作数的运算长度。为了正确处理该指令而指定读数据的长度。 size 如下表所示：

表 10.2 长度说明符

size	内容
B	字节 （8 位）
W	字 （16 位）
L	长字 （32 位）

size 可以是大写字母或者小写字母。

（例）MOV.B #0, R3・・・指定字节。

此说明符只能指定而且必须指定 《RX 族 用户手册 软件篇》的指令格式中明确记载了 （.size）的内容。

(3) 转移距离说明符

在转移指令或者相对子例程转移指令中指定转移距离说明符。

- 记述方法
  `.length`
- 内容
  length 如下表所示：

表 10.3 转移距离说明符

length	内容	
S	3 位 PC 前方相对	(+3 ～ +10)
B	8 位 PC 相对	(–128 ～ +127)
W	16 位 PC 相对	(–32768 ～ +32767)
A	24 位 PC 相对	(–8388608 ～ +8388607)
L	寄存器相对	(–2147483648 ～ +2147183647)

length 可以是大写字母或者小写字母。

（例）

BRA.W label ・・・ 指定 16 位相对。

BRA.L R1 ・・・ 指定寄存器相对。

能省略此说明符。如果省略此说明符，就只有在满足以下的全部条件下，汇编程序才能从 S/B/W/A 中选择代码，使操作码变为最小。

1. 在不是用寄存器记述了此操作数时
2. 在操作数是汇编时确定转移距离的转移目标时
   （例） 标号+汇编时的确定值
   标号–汇编时的确定值
   汇编时的确定值+标号
3. 在同一段内定义操作数的标号时
   在操作数为寄存器时，选择转移距离说明符L。

在条件转移指令的情况下，如果转移距离超出规定的范围，就将条件取反，生成代码。
能用各指令指定的转移距离说明符如下表所示：

表 10.4　各转移指令的转移距离说明符

指令		.S	.B	.W	.A	.L
BCnd	（Cnd=EQ/Z）	○	○	○	×	×
	（Cnd=NE/NZ）	○	○	○	×	×
	（Cnd= 上述以外）	×	○	×	×	×
BRA		○	○	○	○	○
BSR		×	×	○	○	○

10.1.6　操作数的记述方法

(1)　数值

能给程序记述的数值种类有以下 5 种。
用 32 位带符号的数处理所记述的值 （浮点数除外）。

(a)　2 进制数

用 0 ～ 1 的任意数字记述 2 进制数，添加后缀 B 或者 b。

- 记述例子

```
1011000B
1011000b
```

(b)　8 进制数

用 0 ～ 7 的数字记述 8 进制数，添加后缀 O 或者 o。

- 记述例子

```
60702O
60702o
```

(c)　10 进制数

用 0 ～ 9 的数字记述 10 进制数。

- 记述例子

```
9243
```

(d)　16 进制数

用 0 ～ 9、 A ～ F、 a ～ f 记述 16 进制数，添加后缀 H 或者 h。
对于以字母开头的数值，添加前缀 0。

- 记述例子

```
0A5FH
5FH
0a5fh
5fh
```

(e)　浮点数

只能给控制指令 “.FLOAT”和 “.DOUBLE”的操作数记述浮点数。
不能给表达式记述浮点数。
能记述用浮点数表示的以下范围内的值：
FLOAT (32bits) 1.17549435×10^{-38} ～ 3.40282347×10^{38}
DOUBLE (64bits) $2.2250738585072014\times10^{-308}$ ～ $1.7976931348623157\times10^{308}$

- 记述方法
  (尾数)E(指数)
  (尾数)e(指数)
- 记述例子

```
3.4E35      ;3.4×10**35
3.4e-35     ;3.4×10**-35
-.5E20      ;-0.5×10**20
5e-20       ;5.0×10**-20
```

(2)　表达式

能记述数值、符号和运算符的组合表达式。

- 能在运算符和数值之间记述空白字符或者制表符。
- 能记述多个运算符的组合。
- 在将表达式作为符号值进行记述时，为了在汇编时确定表达式的值，需要记述表达式。
- 表达式的项不能使用字符常数。
- 运算结果的范围为–2147483648～2147483647。即使运算结果超出此范围，也不判断为上溢。

(a)　运算符

能给程序记述的运算符一览表如下所示：

- 一元运算符

表 10.5　一元运算符

运算符	功能
+	将后续的值作为正值进行处理。
–	将后续的值作为负值进行处理。
~	将后续的值作为逻辑非值进行处理。
SIZEOF	将给操作数指定的段大小（字节数）作为值进行处理。
TOPOF	将给操作数指定的段的起始地址作为值进行处理。

在 SIZEOF、 TOPOF 和操作数之间记述空白字符或者制表符。
（例） SIZEOF program

- 二元运算符

表 10.6 二元运算符

运算符	功能
+	左边的值加上右边的值。
–	左边的值减去右边的值。
*	左边的值乘右边的值。
/	左边的值除右边的值。
%	处理左边的值除右边的值后的余数。
>>	将左边的值向右移位 （右边的值指示移位次数）。
<<	将左边的值向左移位 （右边的值指示移位次数）。
&	处理左右两边的值各位的逻辑与的值。
|	处理左右两边的值各位的逻辑或的值。
^	处理左右两边的值各位的逻辑异或的值。

- 条件运算符
  只能给控制指令 “.IF”和 “.ELIF”的操作数记述条件运算符。

表 10.7 条件运算符

运算符	功能
>	左边的值大于右边的值。
<	左边的值小于右边的值。
>=	左边的值大于等于右边的值。
<=	左边的值小于等于右边的值。
==	左边的值等于右边的值。
!=	左边的值不等于右边的值。

- 运算优先级的变更运算符

表 10.8 运算优先级的变更运算符

运算符	功能
()	最先进行 () 内的运算。当 1 个表达式中记述了多个 () 时，左侧的优先。 () 能嵌套记述。

(b) 表达式的运算优先级

对于给操作数记述的表达式，根据以下所示的优先顺序，将运算结果的数值作为值进行处理。

- 从优先级高的运算符开始进行运算，运算符的优先级如下表所示。
- 对于优先级相同的运算符，从左开始依次进行运算。
- () 内的运算优先级最高。

表 10.9 表达式的运算优先级

优先级	运算符的种类	运算符
1	运算优先级的变更运算符	()
2	一元运算符	+、 –、 ~、 SIZEOF、 TOPOF
3	二元运算符 1	*、 /、 %
4	二元运算符 2	+、 –
5	二元运算符 3	>>、 <<
6	二元运算符 4	&
7	二元运算符 5	|、 ^
8	条件运算符	>、 <、 >=、 <=、 ==、 !=

(3) 寻址方式

能给指令的操作数记述的寻址方式有以下 3 种：

(a) 一般指令寻址

- 寄存器直接
  指定的寄存器为运算对象。能记述R0～R15和SP， SP解释为R0。
  (R0=SP)
  `Rn(Rn=R0～R15,SP)`
  • 记述例子
  ```
  ADD  R1, R2
  ```
- 立即数
  用#imm表示的立即数为整数。
  用#uimm表示的立即数为无符号整数。
  用#simm表示的立即数为带符号整数。
  #imm:n、 #uimm:n、 #simm:n为n位长的立即数。
  `#imm:8、 #uimm:8、 #simm:8、 #imm:16、 #simm:16、 #simm:24、 #imm:32`

【注】 RTSD 指令的 #uimm:8 必须是确定的值。

• 记述例子
```
MOV.L  #-100, R2          ; #simm:8
```

- 寄存器间接
  寄存器的值为运算对象的有效地址，有效地址的范围为00000000h～FFFFFFFFh。
  `[Rn] (Rn=R0～R15, SP)`
  • 记述例子
  ```
  ADD  [R1], R2
  ```

- 寄存器相对
  将位移量 （dsp）扩展 （零扩展）为32位后的值加上寄存器值后的结果为运算对象的有效地址，有效地址的范围为00000000h～FFFFFFFFh。 dsp:n表示n位长的位移量。
  按以下规则，用换算后的值指定dsp的值。在汇编程序中，恢复换算前的值，填充到指令的码型。

表 10.10 dsp 值的换算规则

指令	规则
取长度说明符的传送指令	根据长度说明符 .B/.W/.L，分别为 1 倍、 2 倍、 4 倍。
取长度扩展说明符的运算指令	根据长度扩展说明符 .B/.UB/.W/.UW/.L，分别为 1 倍、 1 倍、 2 倍、 2 倍、 4 倍。
位操作指令	1 倍
上述以外	4 倍

`dsp:8[Rn], dsp:16[Rn] (Rn=R0～R15、 SP)`
• 记述例子
```
ADD  400[R1], R2          ; dsp:8[Rn] (400/4 = 100)
```
当长度说明符为W/L而地址不是2/4的倍数时，如果在汇编时是确定值，就发生汇编程序错误；如果在汇编时是未确定值，就发生连接错误。

(b) 扩展指令寻址

- 短型立即数
  用#imm表示的立即数为运算对象。如果立即数在汇编时不是确定值，就进行错误处理。

  #imm:1
  此寻址只用于DSP功能指令 （RACW）的src，能记述1或者2。
  • 记述例子
  ```
  RACW  #1                    ; RACW  #imm:1
  ```

  #imm:2
  用#imm表示的2位立即数为运算对象。此寻址只用于指定协处理器指令（MVFCP、 MVTCP、 OPECP）的协处理器号。
  • 记述例子
  ```
  MVTCP  #3, R1, #4:16        ; MVTCP  #imm:2, Rn, #imm:16
  ```

  #imm:3
  用#imm表示的3位立即数为运算对象。此寻址只用于指定位操作指令 （BCLR、 BMCnd、 BNOT、 BSET、 BTST）的位号。
  • 记述例子
  ```
  BSET   #7, R10              ; BSET   #imm:3, Rn
  ```

  #imm:4
  在用于ADD、AND、CMP、MOV、MUL、OR、SUB指令的源操作数时，运算对象是将用#imm表示的4位立即数扩展 （零扩展）为32位后的结果。
  在用于指定MVTIPL指令的中断优先级时，用#imm表示的4位立即数为运算对象。
  • 记述例子
  ```
  ADD   #15, R8               ; ADD   #imm:4, Rn
  ```

  #imm:5
  用#imm表示的5位立即数为运算对象。此寻址只用于指定位操作指令 （BCLR、 BMCnd、 BNOT、 BSET、 BTST）位号、移位指令（SHAR、 SHLL、 SHLR）的移位宽度和循环指令（ROTL、 ROTR）的循环宽度。
  • 记述例子
  ```
  BSET  #31, R10              ; BSET   #imm:5, Rn
  ```

- 短型寄存器相对
  将5位位移量 （dsp）扩展 （零扩展）为32位后的值加上寄存器的值，其结果为运算对象的有效地址，有效地址的范围为00000000h～FFFFFFFFh。
  根据长度说明符.B/.W/.L，分别用1倍、 2倍、 4倍的值指定dsp的值。如果dsp的值在汇编时不是确定值，就进行错误处理。此寻址只用于MOV指令和MOVU指令。
  ```
  dsp:5[Rn] (Rn=R0～R7、 SP)
  ```
  • 记述例子
  ```
  MOV.L  R3,124[R1]           ; dsp:5[Rn] (124/4 = 31)
  ```

  ※ src/dest 的寄存器必须是 R0 ～ R7。

- 后增寄存器间接
  根据长度说明符.B/.W/.L，寄存器的值分别加上1、2、4。更新前的寄存器值为运算对象的有效地址，有效地址的范围为00000000h～FFFFFFFFh。此寻址只用于MOV指令和MOVU指令。
  [Rn+] (Rn=R0～R15、SP)
  • 记述例子
  MOV.L [R3+],R1

- 先减寄存器间接
  根据长度说明符.B/.W/.L，寄存器的值分别减去1、2、4。更新后的寄存器值为运算对象的有效地址，有效地址的范围为00000000h～FFFFFFFFh。此寻址只用于MOV指令和MOVU指令。
  [-Rn] (Rn=R0～R15、SP)
  • 记述例子
  MOV.L [-R3],R1

- 带变址的寄存器间接
  根据长度说明符.B/.W/.L，将变址寄存器（Ri）分别扩大1倍、2倍、4倍后的值加上基址寄存器（Rb）的值，其结果的低32位为运算对象的有效地址，有效地址的范围为00000000h～FFFFFFFFh。此寻址只用于MOV指令和MOVU指令。
  [Ri,Rb] (Ri=R0～R15、SP)　(Rb=R0～R15、SP)
  • 记述例子
  MOV.L [R3,R1],R2
  MOV.L R3, [R1,R2]

(c)　特定指令寻址

- 控制寄存器直接
  指定的控制寄存器为运算对象。
  此寻址只用于MVTC、POPC、PUSHC和MVFC指令。
  PSW、FPSW、USP、ISP、INTB、BPSW、BPC、FINTV、PC、CPEN
  • 记述例子
  STC　PSW,R2

- PSW直接
  指定的标志或者位为运算对象。此寻址只用于CLRPSW指令和SETPSW指令。
  U、I、O、S、Z、C
  • 记述例子
  CLRPSW　U
- 程序计数器相对
  这是用于指定转移指令的转移目标的寻址。

  Rn(Rn=R0～R15、SP)
  将程序计数器的值和Rn的值进行带符号的加法运算后的结果为有效地址。Rn值的范围为–2147483648～2147483647，有效地址的范围为00000000h～FFFFFFFFh。此寻址用于BRA(.L)指令和BSR(.L)指令。

`label(PC + pcdsp:3)`

表示转移指令的转移目标地址。指定的符号和数值为有效地址。
将指定的转移目标地址减去程序计数器的值后的结果作为位移量 （pcdsp），填充到指令的码型。
当转移距离说明符为 “.S” 时，将程序计数器的值和位移量的值进行不带符号的加法运算，其结果的低32位为有效地址。
pcdsp的范围为3≤pcdsp:3≤10。
有效地址的范围为00000000h～FFFFFFFFh。此寻址只用于BRA和BCnd （Cnd==EQ,NE,Z,NZ）。

`label(PC + pcdsp:8/pcdsp:16/pcdsp:24)`

表示转移指令的转移目标地址。指定的符号和数值为有效地址。
将指定的转移目标地址减去程序计数器值后的结果作为位移量 （pcdsp），填充到指令的码型。
当转移距离说明符为 “.B”、“.W” 或者 “.A” 时，将程序计数器的值和位移量的值进行带符号的加法运算，其结果的低32位为有效地址。
pcdsp的范围如下：
当转移距离说明符为 “.B” 时，为–128≤pcdsp:8≤+127。
当转移距离说明符为 “.W” 时，为–32768≤pcdsp:16≤+32767。
当转移距离说明符为 “.A” 时，为–8388608≤pcdsp:24≤+8388607。
有效地址的范围为00000000h～FFFFFFFFh。

(4) 位长说明符

位长说明符指定操作数的立即数或者位移量的大小。

- 记述方法
  `:width`
- 内容
  在操作数记述的立即数或者位移量之后指定此说明符。
  汇编程序选择被指定的位长的寻址方式。
  如果省略此说明符，汇编程序就选择效率最高的位长。
  如果已经记述此说明符，就不选择最佳的位长而选择被指定的位长。
  不能给汇编程序控制指令的操作数记述此说明符。
  能在立即数、位移量的表达式和此说明符之间至少插入1个空白字符。
  如果指定指令格式中不存在的位长，就进行错误处理。
  width中能指定的位长如下：
  2: 表示有效位长为2位。
  #imm:2
  3: 表示有效位长为3位。
  #imm:3
  4: 表示有效位长为4位。
  #imm:4
  5: 表示有效位长为5位。
  #imm:5、 dsp:5
  8: 表示有效位长为8位。
  #uimm:8、 #simm:8、 dsp:8
  16: 表示有效位长为16位。
  #uimm:16、 #simm:16、 dsp:16
  24: 表示有效位长为24位。
  #simm:24
  32: 表示有效位长为32位。
  #imm:32

(5) 长度扩展说明符

在运算指令的源操作数为存储器操作数时，为了指定存储器操作数大小和扩展方法，附加长度扩展说明符。

- 记述方法
  .memex
- 内容
  在存储器操作数之后记述此说明符，不能在操作数和说明符之间插入空白字符。
  长度扩展说明符只对特定的指令和存储器操作数的组合有效，如果指定无效的指令和操作数的组合，就进行错误处理。
  能指定的指令和操作数的组合只是《RX族 用户手册 软件篇》的指令格式的操作数中附加了.memex的模式。
  在省略时，在位操作指令中作为 ‘B’处理，在其他指令中作为 ‘L’处理。
  能指定的长度扩展说明符和效果如下表所示：

表 10.11 长度扩展说明符

长度扩展说明符	效果
B	将 8 位数据扩展 （符号扩展）为 32 位。
UB	将 8 位数据扩展 （零扩展）为 32 位。
W	将 16 位数据扩展 （符号扩展）为 32 位
UW	将 16 位数据扩展 （零扩展）为 32 位。
L	加载 32 位数据。

（记述例子）

```
ADD  [R1].B, R2
AND  125[R1].UB, R2
```

10.1.7 注释的记述方法

在分号 （;）之后继续记述，将分号到行末的内容视为注释。

- 记述例子

```
ADD     R1, R2              ; R2加上R1。
```

10.2　指令的最佳选择

10.2.1　　指令格式的最佳选择

RX 族的有些指令对于相同的处理有多种指令格式。

在汇编程序中，根据指令和寻址方式的指定，选择最短代码的指令格式 （最佳选择）。

(1)　关于立即数

在操作数中有立即数的指令时，汇编程序根据给操作数指定的立即数范围，在能选择的寻址中进行最佳选择。按照从高到低的优先级，立即数的范围如下所示：

表 10.12　立即数的范围

#imm	10 进制记数法	16 进制记数法
#imm:1	1 ～ 2	1H ～ 2H
#imm:2	0 ～ 3	0H ～ 3H
#imm:3	0 ～ 7	0H ～ 7H
#imm:4	0 ～ 15	0H ～ 0FH
#imm:5	0 ～ 31	0H ～ 1FH
#imm:8	–128 ～ 255	–80H ～ 0FFH
#uimm:8	0 ～ 255	0H ～ 0FFH
#simm:8	–128 ～ 127	–80H ～ 7FH
#imm:16	–32768 ～ 65535	–8000H ～ 0FFFFH
#simm:16	–32768 ～ 32767	–8000H ～ 7FFFH
#simm:24	–8388608 ～ 8388607	–800000H ～ 7FFFFFH
#imm:32	–2147483648 ～ 4294967295	–80000000H ～ 0FFFFFFFFH

【注】　*1　也能用 32 位表示 16 进制数。
例：10进制数 “–127” 或者16进制数 “–7FH” 也能表示为 “0FFFFFF81H”

*2　INT 指令的 src 的 #imm 范围为 0 ～ 255。

*3　RTSD 指令的 src 的 #imm 范围为 #uimm:8 的 4 倍值。

(2)　ADC 指令和 SBB 指令

按照优先级从高到低的顺序， ADC 指令和 SBB 指令中作为最佳选择对象的指令格式和操作数如下所示：

【注】　没有记述非最佳选择对象的指令格式和操作数。对于表内的处理长度，如果没有特别说明，就为 “L”。

表 10.13　ADC 指令和 SBB 指令的格式

指令格式	对象			代码长度 [字节]
	src	src2	dest	
ADC src,dest	#simm:8	—	Rd	4
	#simm:16	—	Rd	5
	#simm:24	—	Rd	6
	#imm:32	—	Rd	7
ADC/SBB src,dest	dsp:8[Rs].L	—	Rd	4
	dsp:16[Rs].L	—	Rd	5

在 SBB 指令中不能给 src 指定立即数。

(3) ADD 指令

按照优先级从高到低的顺序， ADD 指令中作为最佳选择对象的指令格式和操作数如下所示：

表 10.14 ADD 指令的格式

指令格式	对象			代码长度 [字节]
	src	src2	dest	
(1) ADD src,dest	#uimm:4	—	Rd	2
	#simm:8	—	Rd	3
	#simm:16	—	Rd	4
	#simm:24	—	Rd	5
	#imm:32	—	Rd	6
	dsp:8[Rs].memex	—	Rd	3(memex = UB)、 4(memex≠ UB)
	dsp:16[Rs].memex	—	Rd	4(memex = UB)、 5(memex≠ UB)
(2) ADD src,src2,dest	#simm:8	Rs	Rd	3
	#simm:16	Rs	Rd	4
	#simm:24	Rs	Rd	5
	#imm:32	Rs	Rd	6

(4) AND、 OR、 SUB、 MUL 指令

按照优先级从高到低的顺序， AND、 OR、 SUB 和 MUL 指令中作为最佳选择对象的指令格式和操作数如下所示：

表 10.15 AND、 OR、 SUB 和 MUL 指令的格式

指令格式	对象			代码长度 [字节]
	src	src2	dest	
AND/OR/SUB/MUL src,dest	#uimm:4	—	Rd	2
	#simm:8	—	Rd	3
	#simm:16	—	Rd	4
	#simm:24	—	Rd	5
	#imm:32	—	Rd	6
	dsp:8[Rs].memex	—	Rd	3(memex = UB)、 4(memex≠ UB)
	dsp:16[Rs].memex	—	Rd	4(memex = UB)、 5(memex≠ UB)

在 SUB 指令中不能给 src 指定 #simm:8/16/24 和 #imm32。

(5) BMCnd 指令

按照优先级从高到低的顺序， BMCnd 指令中作为最佳选择对象的指令格式和操作数如下所示：

表 10.16 BMCnd 指令的格式

指令格式	处理长度	对象			代码长度 [字节]
		src	src2	dest	
BMCnd src,dest	B	#imm:3	—	dsp:8[Rs].B	4
	B	#imm:3	—	dsp:16[Rs].B	5

(6) CMP 指令

按照优先级从高到低的顺序， CMP 指令中作为最佳选择对象的指令格式和操作数如下所示：

表 10.17 CMP 指令的格式

指令格式	处理长度	对象			代码长度 [字节]
		src	src2	dest	
CMP src,src2	L	#uimm:4	Rd	—	2
	L	#uimm:8	Rd	—	3
	L	#simm:8	Rd	—	3
	L	#simm:16	Rd	—	4
	L	#simm:24	Rd	—	5
	L	#imm:32	Rd	—	6
	L	dsp:8[Rs].memex	Rd	—	3(memex = UB)、 4(memex≠ UB)
	L	dsp:16[Rs].memex	Rd	—	4(memex = UB)、5(memex ≠ UB)

(7) DIV、 DIVU、 EMUL、 EMULU、 ITOF、 MAX、 MIN、 TST、 XOR 指令

按照优先级从高到低的顺序，DIV、DIVU、EMUL、EMULU、ITOF、MAX、MIN、MUL、TST 和 XOR 指令中作为最佳选择对象的指令格式和操作数如下所示：

表 10.18 DIV、 DIVU、 EMUL、 EMULU、 ITOF、 MAX、 MIN、 TST 和 XOR 指令的格式

指令格式	对象			代码长度 [字节]
	src	src2	dest	
DIV/DIVU/	#simm:8	—	Rd	4
EMUL/EMULU/ITOF/	#simm:16	—	Rd	5
MAX/MIN/TST/XOR	#simm:24	—	Rd	6
	#imm:32	—	Rd	7
src,dest	dsp:8[Rs].memex	—	Rd	4(memex = UB)、 5(memex≠ UB)
	dsp:16[Rs].memex	—	Rd	5(memex = UB)、 6(memex≠ UB)

在 ITOF 指令中不能给 src 指定 #simm:8/16/24 和 #imm32。

(8) FADD、 FCMP、 FDIV、 FMUL、 FTOI 指令

按照优先级从高到低的顺序， FADD、 FCMP、 FDIV、 FMUL 和 FTOI 指令中作为最佳选择对象的指令格式和操作数如下所示：

表 10.19 FADD、 FCMP、 FDIV、 FMUL 和 FTOI 指令的格式

指令格式	对象			代码长度 [字节]
	src	src2	dest	
FADD/FCMP/FDIV/	#imm:32	—	Rd	7
FMUL/FTOI	dsp:8[Rs].L	—	Rd	4
src,dest	dsp:16[Rs].L	—	Rd	5

在 FTOI 指令中不能给 src 指定 #imm32。

(9) MVTC、 STNZ、 STZ 指令

按照优先级从高到低的顺序， MVTC、 STNZ 和 STZ 指令中作为最佳选择对象的指令格式和操作数如下所示：

表 10.20 MVTC、 STNZ 和 STZ 指令的格式

指令格式	对象			代码长度 [字节]
	src	src2	dest	
MVTC/STNZ/STZ src,dest	#simm:8	—	Rd	4
	#simm:16	—	Rd	5
	#simm:24	—	Rd	6
	#imm:32	—	Rd	7

(10) MOV 指令

按照优先级从高到低的顺序， MOV 指令中作为最佳选择对象的指令格式和操作数如下所示：

表 10.21 MOV 指令的格式

指令格式	size	处理长度	对象			代码长度 [字节]
			src	src2	dest	
MOV(.size) src,dest	B/W/L	size	Rs(Rs=R0-R7)	—	dsp:5[Rd](Rd=R0-R7)	2
	B/W/L	L	dsp:5[Rs](Rs=R0-R7)	—	Rd(Rd=R0-R7)	2
	B/W/L	L	#uimm:8	—	dsp:5[Rd](Rd=R0-R7)	3
	L	L	#uimm:4	—	Rd	2
	L	L	#uimm:8	—	Rd	3
	L	L	#simm:8	—	Rd	3
	L	L	#simm:16	—	Rd	4
	L	L	#simm:24	—	Rd	5
	L	L	#imm:32	—	Rd	6
	B	B	#imm:8	—	[Rd]	3
	W/L	W/L	#simm:8	—	[Rd]	3
	W	W	#imm:16	—	[Rd]	4
	L	L	#simm:16	—	[Rd]	4
	L	L	#simm:24	—	[Rd]	5
	L	L	#imm:32	—	[Rd]	6
	B	B	#imm:8	—	dsp:8[Rd]	4
	W/L	W/L	#simm:8	—	dsp:8[Rd]	4
	W	W	#imm:16	—	dsp:8[Rd]	5
	L	L	#simm:16	—	dsp:8[Rd]	5
	L	L	#simm:24	—	dsp:8[Rd]	6
	L	L	#imm:32	—	dsp:8[Rd]	7
	B	B	#imm:8	—	dsp:16[Rd]	5
	W/L	W/L	#simm:8	—	dsp:16[Rd]	5
	W	W	#imm:16	—	dsp:16[Rd]	6
	L	L	#simm:16	—	dsp:16[Rd]	6
	L	L	#simm:24	—	dsp:16[Rd]	7
	L	L	#imm:32	—	dsp:16[Rd]	8

指令格式	size	处理长度	对象			代码长度 [字节]
			src	src2	dest	
MOV(.size) src,dest	B/W/L	L	dsp:8[Rs]	—	Rd	3
	B/W/L	L	dsp:16[Rs]	—	Rd	4
	B/W/L	size	Rs	—	dsp:8[Rd]	3
	B/W/L	size	Rs	—	dsp:16[Rd]	4
	B/W/L	size	[Rs]	—	dsp:8[Rd]	3
	B/W/L	size	[Rs]	—	dsp:16[Rd]	4
	B/W/L	size	dsp:8[Rs]	—	[Rd]	3
	B/W/L	size	dsp:16[Rs]	—	[Rd]	4
	B/W/L	size	dsp:8[Rs]	—	dsp:8[Rd]	4
	B/W/L	size	dsp:8[Rs]	—	dsp:16[Rd]	5
	B/W/L	size	dsp:16[Rs]	—	dsp:8[Rd]	5
	B/W/L	size	dsp:16[Rs]	—	dsp:16[Rd]	6

(11)　MOVU 指令

按照优先级从高到低的顺序， MOVU 指令中作为最佳选择对象的指令格式和操作数如下所示：

表 10.22　MOVU 指令的格式

指令格式	size	处理长度	对象			代码长度 [字节]
			src	src2	dest	
MOVU(.size) src,dest	B/W	L	dsp:5[Rs](Rs=R0-R7)	—	Rd(Rd=R0-R7)	2
	B/W	L	dsp:8[Rs]	—	Rd	3
	B/W	L	dsp:16[Rs]	—	Rd	4

(12)　PUSH 指令

按照优先级从高到低的顺序， PUSH 指令中作为最佳选择对象的指令格式和操作数如下所示：

表 10.23　PUSH 指令的格式

指令格式	对象			代码长度 [字节]
	src	src2	dest	
PUSH src	dsp:8[Rs]	—	—	3
	dsp:16[Rs]	—	—	4

(13) ROUND 指令

按照优先级从高到低的顺序， ROUND 指令中作为最佳选择对象的指令格式和操作数如下所示：

表 10.24 ROUND 指令的格式

指令格式	对象			代码长度 [字节]
	src	src2	dest	
ROUND src,dest	dsp:8[Rs] dsp:16[Rs]	— —	Rd Rd	4 5

(14) SCCnd 指令

按照优先级从高到低的顺序， SCCnd 指令中作为最佳选择对象的指令格式和操作数如下所示：

表 10.25 SCCnd 指令的格式

指令格式	size	对象			代码长度 [字节]
		src	src2	dest	
SCCnd(.size) src,dest	B/W/L B/W/L	— —	— —	dsp:8[Rd] dsp:16[Rd]	4 5

(15) XCHG 指令

按照优先级从高到低的顺序， XCHG 指令中作为最佳选择对象的指令格式和操作数如下所示：

表 10.26 XCHG 指令的指令格式

指令格式	处理长度	对象			代码长度 [字节]
		src	src2	dest	
XCHG src,dest	L L	dsp:8[Rs].memex dsp:16[Rs].memex	— —	Rd Rd	4(memex = UB)、5(memex ≠ UB) 5(memex = UB)、 6(memex≠UB)

(16) BCLR、 BNOT、 BSET、 BTST 指令

按照优先级从高到低的顺序，BCLR、BNOT、BSET 和 BTST 指令中作为最佳选择对象的指令格式和操作数如下所示：

表 10.27 BCLR、 BNOT、 BSET 和 BTST 指令的格式

指令格式	处理长度	对象			代码长度 [字节]
		src	src2	dest	
BCLR/BNOT/BSET/BTST src,dest	B B	#imm:3 #imm:3	— —	dsp:8[Rd].B dsp:16[Rd].B	3 4
	B B	Rs Rs	— —	dsp:8[Rd].B dsp:16[Rd].B	4 5

10.2.2 转移指令的最佳选择

(1) 相对无条件转移指令 （BRA）的最佳选择

(a) 能指定的转移距离说明符

.S 3 位 PC 相对 （PC+pcdsp:3,3 ≤ pcdsp:3 ≤ 10）
.B 8 位 PC 相对 （PC+pcdsp:8， –128 ≤ pcdsp:8 ≤ 127）
.W 16 位 PC 相对 （PC+pcdsp:16， –32768 ≤ pcdsp:16 ≤ 32767）
.A 24 位 PC 相对 （PC+pcdsp:24， –8388608 ≤ pcdsp:24 ≤ 8388607）
.L 寄存器相对 （PC+Rs， –2147483648 ≤ Rs ≤ 2147483647）

※不通过最佳选择而只在操作数为寄存器时选择寄存器相对。

(b) 最佳选择

- 在汇编程序中，如果相对无条件转移指令的操作数满足转移优化对象条件，就选择最短的转移距离。有关条件，请参照 “10.1.5 (3) 转移距离说明符”。
- 对于不满足条件的操作数，选择24位PC相对 （.A）。

(2) 相对子例程转移指令 （BSR）的最佳选择

(a) 能指定的转移距离说明符

.W 16 位 PC 相对 （PC+pcdsp:16， –32768 ≤ pcdsp:16 ≤ 32767）
.A 24 位 PC 相对 （PC+pcdsp:24， –8388608 ≤ pcdsp:24 ≤ 8388607）
.L 寄存器相对 （PC+Rs， –2147483648 ≤ Rs ≤ 2147483647）

※不通过最佳选择而只在操作数为寄存器时选择寄存器相对。

(b) 最佳选择

- 在汇编程序中，如果相对子程序转移指令的操作数满足转移优化对象条件使，就选择最短的转移距离。有关条件，请参照 “10.1.5(3) 转移距离说明符 ”。
- 对于不满足条件的操作数，选择24位PC相对 （.A）。

(3) 条件转移指令 （BCnd）的最佳选择

(a) 能指定的转移距离说明符

BEQ.S 3 位 PC 相对 （PC+pcdsp:3， 3 ≤ pcdsp:3 ≤ 10）
BNE.S 3 位 PC 相对 （PC+pcdsp:3， 3 ≤ pcdsp:3 ≤ 10）
BCnd.B 8 位 PC 相对 （PC+pcdsp:8， –128 ≤ pcdsp:8 ≤ 127）
BEQ.W 16 位 PC 相对 （PC+pcdsp:16， –32768 ≤ pcdsp:16 ≤ 32767）
BNE.W 16 位 PC 相对 （PC+pcdsp:16， –32768 ≤ pcdsp:16 ≤ 32767）

(b) 最佳选择

- 在汇编程序中，如果条件转移指令的操作数满足转移优化对象条件，就选择并且生成逻辑取反后的条件转移指令和最佳转移距离的相对无条件转移指令组合的最佳条件转移代码。
- 对于不满足条件的操作数，选择8位PC相对 （.B）或者16位PC相对 （.W）。

(c) 被转换的条件转移指令的替代指令

表 10.28 条件转移指令的替代规则

条件转移指令	替代转移指令	条件转移指令	替代转移指令
BNC/BLTU dest	BC ..xx BRA.A dest ..xx:	BC/BGEU dest	BNC ..xx BRA.A dest ..xx:
BLEU dest	BGTU ..xx BRA.A dest ..xx:	BGTU dest	BLEU ..xx BRA.A dest ..xx:
BNZ/BNE dest	BZ ..xx BRA.A dest ..xx:	BZ/BEQ dest	BNZ ..xx BRA.A dest ..xx:
BPZ dest	BN ..xx BRA.A dest ..xx:	BO dest	BNO ..xx BRA.A dest ..xx:
BGT dest	BLE ..xx BRA.A dest ..xx:	BLE dest	BGT ..xx BRA.A dest ..xx:
BGE dest	BLT ..xx BRA.A dest ..xx:	BLT dest	BGE ..xx BRA.A dest ..xx:

上述内容是相对无条件转移指令的转移距离为 24 位 PC 相对的情况。

因为在内部处理 “..xx”标号和相对无条件转移指令，所以只在汇编列表文件中生成代码。

10.3 汇编程序控制指令的记述方法

汇编程序控制指令包括一般的汇编程序控制指令（以下称为汇编程序控制指令）和高级语言的汇编程序控制指令。

10.3.1 地址控制指令

地址控制指令是汇编程序在更新地址时进行指示的控制指令。

除绝对地址格式的段内地址以外，汇编程序控制的地址为可再定位的值。

表 10.29 地址控制指令

控制指令	功能内容
.ORG	声明起始地址。 记述本控制指令的段为绝对地址格式的段。
.OFFSET	指定从段起始位置开始的偏移量。 只能在相对地址格式的段内记述本控制指令。
.ENDIAN	指定段的字节序。
.BLKB	以 1 字节为单位确保 RAM 区。
.BLKW	以 2 字节为单位确保 RAM 区。
.BLKL	以 4 字节为单位确保 RAM 区。
.BLKD	以 8 字节为单位确保 RAM 区。
.BYTE	将 1 字节数据保存到 ROM 区。
.WORD	将 2 字节数据保存到 ROM 区。
.LWORD	将 4 字节数据保存到 ROM 区。
.FLOAT	将用 4 字节表示的浮点数据保存到 ROM 区。
.DOUBLE	将用 8 字节表示的浮点数据保存到 ROM 区。
.ALIGN	将定位计数器调整为边界调整数的倍数。

地址声明

.ORG

格　式　　.ORG △ < 数值 >

说　明　　将记述本控制指令的段作为绝对地址格式。
记述本控制指令的段地址为绝对值。
决定本控制指令下一行记述的助记符代码的保存地址。
决定本控制指令下一行记述的区域确保控制指令要确保的存储器地址。

例

```
.SECTION          value,ROMDATA
.ORG              0FF00H
.BYTE             "abcdefghijklmnopqrstuvwxyz"
.ORG              0FF80H
.BYTE             "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
.END
```

在以下情况下，因为没有在 .SECTION 之后记述 .ORG，所以发生错误。

```
.SECTION          value,ROMDATA
.BYTE             "abcdefghijklmnopqrstuvwxyz"
.ORG              0FF80H
.BYTE             "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
.END
```

备　注　　必须在段控制指令之后记述本控制指令。
在 “.SECTION” 的下一行没有记述 “.ORG” 时，该段为相对地址格式的段。
必须在控制指令和操作数之间记述空白字符或者制表符。
能给操作数记述的数值范围是 0 ～ 0FFFFFFFFH。
能给操作数记述表达式和符号。但是，必须在汇编时是确定值。
不能在指定了相对地址格式的段内记述本控制指令。
如果是在绝对地址格式的段内，就能进行多次记述。如果指定小于本控制指令记述行的地址值，就发生错误。

偏移量的声明

.OFFSET

格　式　　.OFFSET △ < 数值 >

说　明　　指定从段起始位置开始的偏移量。
决定从保存本控制指令下一行记述的助记符代码的段起始位置开始的偏移量。
决定本控制指令下一行记述的区域确保控制指令要确保存储器的段起始位置开始的偏移量。

例

```
.SECTION        value,ROMDATA
.BYTE           "abcdefghijklmnopqrstuvwxyz"
.OFFSET         80H
.BYTE           "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
.END
```

在以下情况下，因为指定了小于 .OFFSET 记述行的偏移量的值，所以发生错误。

```
.SECTION        value,ROMDATA
.OFFSET         80H
.BYTE           "abcdefghijklmnopqrstuvwxyz"
.OFFSET         70H
.BYTE           "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
.END
```

备　注　　必须在控制指令和操作数之间记述空白字符或者制表符。
能给操作数记述的数值范围是 0 ～ 0FFFFFFFFH。
能给操作数记述表达式和符号。但是，必须在汇编时是确定值。
不能在指定了绝对地址格式的段内记述本控制指令。
如果是在相对地址格式的段内，就能进行多次记述。如果指定小于本控制指令记述行的偏移量的值，就发生错误。

字节序的指定

.ENDIAN

格　式　　.ENDIAN △ BIG
.ENDIAN △ LITTLE

说　明　　指定记述本控制指令的段的字节序。
记述 .ENDIAN BIG 的段数据的字节排列顺序为 Big Endian。
记述 .ENDIAN LITTLE 的段数据的字节排列顺序为 Little Endian。
没有记述本控制指令的段数据的字节排列顺序取决于 -endian 选项。

例

```
.SECTION        value,ROMDATA
.ORG            0FF00H
.ENDIAN         BIG
.BYTE           "abcdefghijklmnopqrstuvwxyz"
```

在以下情况下，因为没有在 .SECTION 或者 .ORG 之后记述 .ENDIAN，所以发生错误。

```
.SECTION        value,ROMDATA
.ORG            0FF00H
.BYTE           "abcdefghijklmnopqrstuvwxyz"
.ENDIAN         BIG
.BYTE           "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

备　注　　必须在 .SECTION 控制指令或者紧接在后面的 .ORG 控制指令之后记述本控制指令。
必须在控制指令和操作数之间记述空白字符或者制表符。
不能给 CODE 属性的段追加本控制指令。

1 字节区域的确保

.BLKB

格　式　　△ .BLKB △ < 操作数 >
△ < 标号名 :> △ .BLKB △ < 操作数 >

说　明　　以 1 字节为单位确保指定字节数的 RAM 区。
也能给确保的 RAM 地址定义标号名。

例
```
symbol          .EQU 1
                .SECTION area,DATA
work1:          .BLKB 1
work2:          .BLKB symbol
                .BLKB symbol+1
```

备　注　　必须在 DATA 属性的段内记述本控制指令。在定义段时，通过在段名之后记述 “,DATA”，使段属性变为 DATA。
必须在控制指令和操作数之间记述空白字符或者制表符。
能给操作数记述数值、符号和表达式。
必须在汇编时确定操作数的值。
在给区域定义标号名时，必须在控制指令之前记述标号。
必须给标号名记述冒号 （:）。

2 字节区域的确保

.BLKW

格　式　　△ .BLKW △ < 操作数 >
△ < 标号名 :> △ .BLKW △ < 操作数 >

说　明　　以 2 字节为单位确保指定个数的 RAM 区。
也能给确保的 RAM 地址定义标号名。

例
```
symbol          .EQU 1
                .SECTION area,DATA
work1:          .BLKW 1
work2:          .BLKW symbol
                .BLKW symbol+1
```

备　注　　必须在 DATA 属性的段内记述本控制指令。在定义段时，通过在段名之后记述 “,DATA”，使段属性变为 DATA。
必须在控制指令和操作数之间记述空白字符或者制表符。
能给操作数记述数值、符号和表达式。
必须在汇编时确定操作数的值。
在给区域定义标号名时，必须在控制指令之前记述标号。
必须给标号名记述冒号 （:）。

4 字节区域的确保

.BLKL

格　式　　△ .BLKL △ < 操作数 >
△ < 标号名 :> △ .BLKL △ < 操作数 >

说　明　　以 4 字节为单位确保指定个数的 RAM 区。
也能给确保的 RAM 地址定义标号名。

例
```
symbol          .EQU 1
                .SECTION area,DATA
work1:          .BLKL 1
work2:          .BLKL symbol
                .BLKL symbol+1
```

备　注　　必须在 DATA 属性的段内记述本控制指令。在定义段时，通过在段名之后记述 “,DATA”，使段属性变为 DATA。
必须在控制指令和操作数之间记述空白字符或者制表符。
能给操作数记述数值、符号和表达式。
必须在汇编时确定操作数的值。
在给区域定义标号名时，必须在控制指令之前记述标号。
必须给标号名记述冒号 （:）。

8 字节区域的确保

.BLKD

格　式　　△ .BLKD △ < 操作数 >
△ < 标号名 :> △ .BLKD △ < 操作数 >

说　明　　以 8 字节为单位确保指定个数的 RAM 区。
也能给确保的 RAM 地址定义标号名。

例
```
symbol          .EQU 1
                .SECTION area,DATA
work1:          .BLKD 1
work2:          .BLKD symbol
                .BLKD symbol+1
```

备　注　　必须在 DATA 属性的段内记述本控制指令。在定义段时，通过在段名之后记述 “,DATA”，使段属性变为 DATA。
必须在控制指令和操作数之间记述空白字符或者制表符。
能给操作数记述数值、符号和表达式。
必须在汇编时确定操作数的值。
在给区域定义标号名时，必须在控制指令之前记述标号。
必须给标号名记述冒号 （:）。

1 字节数据的保存

.BYTE

格　式　　△ .BYTE △ < 操作数 >
△ < 标号名 :> △ .BYTE △ < 操作数 >

说　明　　将 1 字节的固定数据保存到 ROM。
也能给保存数据的地址定义标号名。

例　　< 当指定 endian=little 选项时 >

```
.SECTION value,ROMDATA
.BYTE 1
.BYTE "data"
.BYTE symbol
.BYTE symbol+1
.BYTE 1,2,3,4,5
.END
```

< 当指定 endian=big 选项时 >

```
.SECTION  program,CODE,ALIGN=4
MOV.L  R1,R2
.ALIGN 4
.BYTE 080000000H
.END
```

备　注　　必须在控制指令和操作数之间记述空白字符或者制表符。
能给操作数记述数值、符号和表达式。
在给操作数记述字符或者字符串时，必须用单引号 （‘）或者双引号 （“）括起来。此时保存的数据为 ASCII 码字符。
在定义标号时，必须在控制指令之前记述标号名。
必须给标号名记述冒号 （:）。

当指定 endian=big 选项时，只能在符合以下条件的段内记述本控制指令，否则就发生错误。

(1) ROMDATA 段

```
.SECTION data,ROMDATA
```

(2) 在定义段时给地址调整指示了 4 或者 8 的相对地址格式的 CODE 段

```
.SECTION program,CODE,ALIGN=4
```

(3) 绝对地址格式的 CODE 段

```
.SECTION program,CODE
.ORG 0fff00000H
```

当指定 endian=big 选项时，如果给 CODE 属性的段记述本控制指令，就必须在前一行记述地址调整控制指令 （.ALIGN　4）并且分配到 4 字节边界。如果没有记述，汇编程序就输出警告并且自动分配到 4 字节边界。

2 字节数据的保存

.WORD

格　式　　△ .WORD △ < 操作数 >
△ < 标号名 :> △ .WORD △ < 操作数 >

说　明　　将 2 字节的固定数据保存到 ROM。
也能给保存数据的地址定义标号名。

例
```
.SECTION value,ROMDATA
.WORD             1
.WORD             "da","ta"
.WORD             symbol
.WORD             symbol+1
.WORD             1,2,3,4,5
.END
```

备　注　　必须在控制指令和操作数之间记述空白字符或者制表符。
能给操作数记述数值、符号和表达式。
在给操作数记述字符或者字符串时，必须用单引号 （‘）或者双引号 （“）括起来。此时保存的数据为 ASCII 码字符。
在定义标号时，必须在控制指令之前记述标号名。
必须给标号名记述冒号 （:）。
能给操作数记述的字符串长度不超过 2 个字符。

4 字节数据的保存

.LWORD

格　式　　△ .LWORD △ < 操作数 >
△ < 标号名 :> △ .LWORD △ < 操作数 >

说　明　　将 4 字节的固定数据保存到 ROM。
也能给保存数据的地址定义标号名。

例
```
.SECTION value,ROMDATA
.LWORD          1
.LWORD          "data"
.LWORD          symbol
.LWORD          symbol+1
.LWORD          1,2,3,4,5
.END
```

备　注　　必须在控制指令和操作数之间记述空白字符或者制表符。
能给操作数记述数值、符号和表达式。
在给操作数记述字符或者字符串时，必须用单引号 （‘）或者双引号 （“）括起来。此时保存的数据为 ASCII 码字符。
在定义标号时，必须在控制指令之前记述标号名。
必须给标号名记述冒号 （:）。
能给操作数记述的字符串长度不超过 4 个字符。

4 字节数据的保存

.FLOAT

格　式　　△ .FLOAT △ < 数值 >
　　△ < 标号名 :> △ .FLOAT △ < 数值 >

说　明　　将 4 字节的固定数据保存到 ROM。
　　也能给保存数据的地址定义标号名。

例
```
                        .FLOAT   5E2
constant:               .FLOAT   5e2
```

备　注　　必须给操作数记述浮点数。
　　必须在控制指令和操作数之间记述空白字符或者制表符。
　　在定义标号时，必须在控制指令之前记述标号名。
　　必须给标号名记述冒号（:）。

8 字节数据的保存

.DOUBLE

格　式　　△ .DOUBLE △ < 数值 >
　　△ < 标号名 :> △ .DOUBLE △ < 数值 >

说　明　　将 8 字节的固定数据保存到 ROM。
　　也能给保存数据的地址定义标号名。

例
```
                        .DOUBLE  5E2
constant:               .DOUBLE  5e2
```

备　注　　必须给操作数记述浮点数。
　　必须在控制指令和操作数之间记述空白字符或者制表符。
　　在定义标号时，必须在控制指令之前记述标号名。
　　必须给标号名记述冒号（:）。

地址调整

.ALIGN

格　式　　△ .ALIGN △ < 调整值 >
< 调整值 >:[2|4|8]

说　明　　将本控制指令下一行代码的保存地址调整为 2、 4 或者 8 字节。
当段属性为 CODE 或者 ROMDATA 时，给调整地址后的结果为空的位置写 NOP 码 （03H）。
当段属性为 DATA 时，只进行地址调整。

例　　
```
.SECTION program,CODE,ALIGN=4
MOV.L  R1, R2
.ALIGN  4          ; 将地址调整为 4 的倍数。
RTS
.END
```

备　注　　能在符合以下条件的段内记述本控制指令：
(1) 在定义段时指示地址调整的相对地址格式的段
```
.SECTION program,CODE,ALIGN=4
```
(2) 绝对地址格式的段
```
.SECTION program,CODE
.ORG 0fff00000H
```
如果是相对地址格式的段并且在 .SECTION 控制指令行给未指定 ALIGN 的段记述本控制指令，就输出警告。
如果指定的值大于段的边界调整数，就输出警告。

10.3.2　汇编程序控制指令

控制指令对指令的机器码生成进行控制，但是本身不生成数据，也不更新地址。

表 10.30　汇编程序控制指令

控制指令	功能内容
.EQU	设定符号。
.END	指定汇编语言文件的结束。
.INCLUDE	将指定的文件内容读到记述本控制指令的位置。

数值符号的定义

.EQU

格　式　　< 名称 > △ .EQU △ < 数值 >

说　明　　给符号定义 32 位带符号的整数值（-2147483648 ～ 2147483647）范围内的值。
通过本控制指令进行符号的定义，能使用符号调试功能。

例

```
symbol  .EQU 1
symbol1 .EQU symbol+symbol
symbol2 .EQU 2
```

备　注　　必须在汇编时确定能给符号定义的值。
必须在控制指令和操作数之间记述空白字符或者制表符。
能给定义符号的操作数记述符号，但是不能记述前方参照的符号名。
能给操作数记述表达式。
能将符号指定为全局符号。
当通过本控制指令和 .DEFINE 控制指令声明了同名符号时，先记述的符号优先。

汇编语言文件结束的声明

.END

格　式　　.END

说　明　　声明汇编语言文件的结束。
只将记述本控制指令的行之后的记述内容输出到汇编列表文件，不进行代码生成等处理。

例　　.END

备　注　　1 个汇编语言文件必需记述 1 条本控制指令。

***include* 文件的指定**

.INCLUDE

格　式　　.INCLUDE △ <include 文件名 >

说　明　　将 include 文件的全部内容读到汇编语言文件的行。
如果在读取的汇编语言文件内记述了用本控制指令读取的 include 文件内容，就作为同 1 个汇编语言文件进行处理。
include 文件最多能嵌套 30 层。
如果给 include 文件名记述绝对路径，就检索所记述的目录内的文件。
如果找不到文件，就发生错误。
如果没有给 include 文件名记述绝对路径，就按以下顺序检索文件：
(1) 在启动汇编程序时，如果命令行中指定的汇编语言文件名没有指定目录，就检索 .INCLUDE 控制指令指定的 include 文件名；如果命令行中指定的汇编语言文件名指定了目录，就给 .INCLUDE 控制指令指定的 include 文件名附加命令行中指定的目录名，然后进行检索。
(2) 检索汇编程序选项 -include 指定的目录。
(3) 检索环境变量 INC_RXA 设定的目录。

例　　.INCLUDE initial.src
.INCLUDE ..FILE@.inc

备　注　　必须在控制指令和操作数之间记述空白字符或者制表符。
必须给操作数的 include 文件名记述文件扩展名。
能给操作数记述含有控制指令 “..FILE” 和 “@” 的字符串。
除开头以外，文件名能包含空白字符。
不能用双引号 “"” 将文件名括起来。
不能将文件本身指定为 include 文件。

10.3.3 连接控制指令

此控制指令执行将程序分割成多个文件进行记述的可再定位汇编程序。

表 10.31 连接控制指令

控制指令	功能内容
.SECTION	定义用于重新分配地址的最小单位的段。
.GLB	将符号声明为外部符号。
.RVECTOR	将符号注册到可变向量。

段的定义

.SECTION

格　式　　.SECTION △ < 段名 >
.SECTION △ < 段名 >,< 段属性 >
.SECTION △ < 段名 >,< 段属性 >,ALIGN=[2|4|8]
.SECTION △ < 段名 >,ALIGN=[2|4|8]
< 段属性 >:[CODE|ROMDATA|DATA]

说　明　　指定段的声明和重新开始。
(1) 段的声明
　　指定段名和段属性，定义段的开始。
(2) 段的重新开始
　　重新开始源程序中已经存在的段。在重新开始段时，指定已经存在的段名。段属性和调整值还是最初声明的内容。
当指定 ‘ALIGN=’ 时，能对指定的段更改调整值。
能给指定 ALIGN 的相对地址格式的段或者绝对地址格式的段记述控制指令 “.ALIGN”。
当没有指定 ALIGN 时，该段的边界调整数为 1。

例

```
.SECTION        program,CODE
NOP
.SECTION        ram,DATA
.BLKB           10
.SECTION        tbl1,ROMDATA
.BYTE           "abcd"
.SECTION        tbl2,ROMDATA,ALIGN=8
.LWORD          11111111H,22222222H
.END
```

备　注　　必须记述段名。
在记述汇编程序控制指令（用于确保存储区或者将数据保存到存储器）时，必须通过本控制指令进行段的定义。
在记述助记符时，必须通过本控制指令进行段的定义。
必须在段名之后记述段属性和 ALIGN。
在指定段属性和 ALIGN 时，必须用逗号分开记述。
段属性和 ALIGN 的记述顺序是任意的。
段属性能记述’ CODE’、’ ROMDATA’ 或者’ DATA’。
能省略段属性。此时，汇编程序将段属性作为 CODE 进行处理。

注意事项　　当指定 -endian=big 时，不能给绝对地址格式的 CODE 段的起始地址指定 4 倍以外的值。
当指定 -endian=big 时，绝对地址格式的 CODE 段输出警告，汇编程序将 NOP(0x03) 写到段的末尾，使段的大小成为 4 的倍数。

全局声明

.GLB

格　式　　.GLB △ < 名称 >
.GLB △ < 名称 >[,< 名称 > …]

说　明　　将本控制指令指定的标号和符号声明为全局标号或者全局符号。
对于是本控制指令指定的而在文件内尚未定义的标号和符号，将其作为外部文件中已定义的标号和符号进行处理。
对于是本控制指令指定的并且在文件内已定义的标号和符号，能从外部参照进行处理。

例

```
.GLB      name1,name2,name3
.GLB      name4
.SECTION  program
MOV.L     #name1,R1
```

备　注　　必须在控制指令和操作数之间记述空白字符或者制表符。
给操作数记述作为全局标号的标号名。
给操作数记述作为全局符号的符号名。
在给操作数记述多个符号名时，必须用逗号 （,）分开记述。

可变向量的注册

.RVECTOR

格　式　　.RVECTOR △ < 序号 >,< 名称 >

说　明　　将本控制指令指定的标号和符号注册为可变向量。
作为向量号，能给本控制指令的 < 序号 > 记述 0 ～ 255 的常数。
能给本控制指令的 < 名称 > 指定文件内定义的标号或者符号。
通过优化连接编辑程序，将注册的可变向量汇集到 1 个 C$VECT 段。

例

```
.RVECTOR   50,_rvfunc
_rvfunc:
MOV.L  #0,R1
RTE
```

备　注　　必须在控制指令和操作数之间记述空白字符或者制表符。

10.3.4　　汇编列表控制指令

此控制指令对输出到汇编列表文件的信息以及汇编列表文件的格式进行控制，不影响代码的生成。

表 10.32　汇编列表控制指令

控制指令	功能内容
.LIST	在生成汇编列表文件时，控制是否以汇编语言文件的行为单位输出到汇编列表文件。

汇编列表的输出控制指令

.LIST

格　式　　.LIST △ [ON|OFF]

说　明　　能停止（OFF）以行为单位输出到汇编列表文件。
即使在停止以行为单位输出到列表的范围内，也将发生错误的行输出到汇编列表文件。
能开始（ON）以行为单位输出到汇编列表文件。
在没有指定本控制指令时，将全部行输出到汇编列表文件。

例　　
```
.LIST  ON
.LIST  OFF
```

备　注　　必须在控制指令和操作数之间记述空白字符或者制表符。
在停止行的输出时，必须给操作数记述 ‘OFF’。
在开始行的输出时，必须给操作数记述 ‘ON’。

10.3.5 条件汇编控制指令

能使用条件汇编控制指令指定是否对指定范围内的行进行汇编。

表 10.33 条件汇编控制指令

控制指令	功能内容
.IF	表示条件汇编块的开始，判断条件。
.ELIF	在至少记述 2 个条件汇编块时，判断第二个以后的条件。
.ELSE	在全部条件都不满足时，表示汇编块的开始。
.ENDIF	表示条件汇编块的结束。

条件汇编指令

.IF, .ELIF, .ELSE, .ENDIF

格　式

```
.IF △条件表达式
主体
.ELIF △条件表达式
主体
.ELSE
主体
.ENDIF
```

说　明

根据 .IF 和 .ELIF 记述的条件，对汇编块进行控制。
判断 .IF 和 .ELIF 的操作数记述的条件，如果条件为真，就对后面的主体进行汇编。
如果条件为真，被汇编的行就在控制指令 “.ELIF”、“.ELSE”和 “.ENDIF”行之前。
能在条件汇编块内记述汇编语言文件能记述的全部指令。
根据条件表达式的结果，进行条件汇编。

例

＜条件表达式的记述例子＞

```
sym < 1
sym+2 < data1
sym+2 < data1+2
'smp1' == name
```

＜条件汇编的记述例子＞

```
.IF                TYPE==0
.byte              "Proto Type Mode"
.ELIF              TYPE>0
.byte              "Mass Produciton Mode"
.ELSE
.byte              "Debug Mode"
.ENDIF
```

备　注

必须给 .IF 控制指令和 .ELIF 控制指令的操作数记述条件表达式。
必须在控制指令 （.IF 和 .ELIF）和操作数之间记述空白字符或者制表符。
只能给控制指令的操作数记述 1 个条件表达式。
必须给条件表达式记述条件运算符。
能记述的运算符如下所示：

表 10.34 IF 控制指令和 .ELIF 控制指令的条件运算符

条件运算符	内容
>	当左边的值大于右边的值时，为真。
<	当左边的值小于右边的值时，为真。
>=	当左边的值大于等于右边的值时，为真。
<=	当左边的值小于等于右边的值时，为真。
==	当左边的值等于右边的值时，为真。
!=	当左边的值不等于右边的值时，为真。

用带符号的 32 位运算条件表达式。
能给条件运算符的左边和右边记述符号。
能给条件运算符的左边和右边记述表达式。必须按照“10.1.6 操作数的记述方法”的“(2) 表达式”记述表达式。
能给条件运算符的左边和右边记述字符串。必须用单引号（'）或者双引号（"）括起来记述字符串。此时，用字符码的值判断字符串的大小。
例）
当 "ABC"<"CBA" -> 414243 < 434241 时，为真。
当 "C" < "A" -> 43 < 41 时，为伪。
能在条件运算符的前后记述空白字符或者制表符。
能给控制指令“.IF”和“.ELIF”的操作数记述条件表达式。
不判断运算结果的上溢。
不能对符号进行前方参照（参照在本控制指令行之后定义的符号）。
如果记述了前方参照的符号或者未定义的符号，就将值作为 0 来判断表达式。

10.3.6 扩展功能控制指令

此控制指令不影响代码的生成。

表 10.35 扩展功能控制指令

控制指令	功能内容
.ASSERT	将操作数记述的字符串输出到标准错误输出文件或者文件。
?	指定临时标号的定义和参照。
@	连接 @ 前后的字符串，作为 1 个字符串进行处理。
..FILE	表示汇编程序正在处理的汇编语言文件名。
.STACK	给指定的符号定义堆栈值。
.LINE	更改行号。
.DEFINE	定义要替换的符号。

【注】 给编译程序要输出的汇编语言文件记述 .FILE。 .FILE 只在编译程序输出的汇编语言文件中有效，不能在用户建立的汇编语言文件中使用。

指定字符串的输出

.ASSERT

格　式　　.ASSERT △ “< 字符串 >”
.ASSERT △ “< 字符串 >” > △ < 文件名 >
.ASSERT △ “< 字符串 >” >> △ < 文件名 >

说　明　　在汇编时，将给操作数记述的字符串输出到标准错误输出文件。
当指定了文件名时，将给操作数记述的字符串输出到文件。
当给文件名记述了绝对路径时，在记述的目录中生成文件。
没有给文件名记述绝对路径的情况：
(1) 当没有给 output 选项指定的文件名指定目录时，在当前目录中生成本控制指令指定的文件。
(2) 当给 output 选项指定的文件名指定目录时，在 output 选项指定的文件目录中生成本控制指令指定的文件。
(3) 当没有 output 选项时，在和汇编程序启动时命令行指定的文件相同的目录中生成文件。
如果给文件名记述了控制指令 “..FILE”，就在和汇编程序启动时命令行指定的文件相同目录中生成文件。

例　　将信息输出到 sample.dat 文件。

```
.ASSERT "string" > sample.dat
```

将信息输出到 sample.dat 文件。

```
.ASSERT "string" >> sample.dat
```

将信息输出到和当前正在处理的文件同名并且不带扩展名的文件。

```
.ASSERT "string" > ..FILE
```

备　注　　必须在操作数和控制指令之间记述空白字符或者制表符。
必须用双引号将操作数的字符串括起来记述。
在将字符串输出到文件时，必须在 “>” 或者 “>>” 之后指定文件名。
“>” 表示生成新文件，并且将信息输出到该文件。如果和以前的文件同名，就覆盖该文件。
“>>” 表示给文件内容追加输出信息。如果指定的文件不存在，就生成新文件。
能在 “>” 或者 “>>” 前后记述空白字符或者制表符。
能给文件名记述控制指令 “..FILE”。

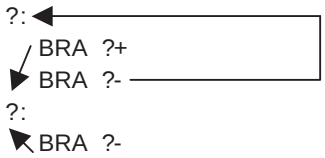
临时标号

?

格　式　　?:
△ < 助记符 > △ ?+
△ < 助记符 > △ ?-

说　明　　定义临时标号。
参照之前或者之后刚定义的临时标号。
能在同一文件内进行定义和参照。
在文件内最多能定义 65535 个临时标号。此时，如果在文件内记述了“.INCLUDE”，就能记述包括 include 文件中的临时标号在内的最多 65535 个标号。
将临时标号的转换结果输出到汇编列表文件。

例



指向箭头的临时标号。

备　注　　必须给想定义为临时标号的行记述“?:”。
当要参照之前刚定义的临时标号时，必须给指令的操作数记述“?-”。
当要参照之后刚定义的临时标号时，必须给指令的操作数记述“?+”。
能参照的标号只是之前和之后刚定义的标号。

字符串的连接

@

格　式　　< 字符串 >@< 字符串 >[@< 字符串 > ...]

说　明　　连接宏参数、宏变量、保留符号、控制指令 “..FILE” 展开的文件名和指定的字符串。

例　　文件名的连接例子：
当正在处理的文件名为 sample1.src 时，将信息输出到 sample.dat 文件。

```
.ASSERT   "sample" > ..FILE@.dat
```

字符串的连接例子：

```
mov_nibble .MACRO p1,src,dest
 MOV.@p1 src,dest
 .ENDM

mov_nibble  W,R1,R2      ; 调用宏。

 MOV.W R1,R2             ;宏展开后的代码
```

备　注　　将在本控制指令前后记述的空白字符和制表符作为字符串进行连接。
能在本控制指令前后记述字符串。
在将 @ 记述为字符数据 （40H）时，必须用双引号 （"）括起来记述。如果用单引号 （'）将含有 @ 的字符串括起来记述，就连接 @ 前后的字符串。
一行能记述多次。
如果将连接的字符串用作名称，就不能在本控制指令前后记述空白字符和制表符。

替换为源文件名信息

..FILE

格　式　　..FILE

说　明　　展开为汇编程序正在处理的文件名 （汇编语言文件名或者 include 文件）。

例　　当汇编语言文件名为 “sample.src” 时，将信息输出到 “sample” 文件。

```
.ASSERT "sample" > ..FILE
```

当汇编语言文件名为 “sample.src” 时，包含 “sample.inc” 文件。

```
.INCLUDE ..FILE@.inc
```

在 “sample.src” 文件中包含的 "incl.inc" 内记述了上述行时，通常将字符串输出到 “incl.mes”。

```
.ASSERT "sample" > ..FILE@.mes
```

备　注　　能给控制指令 “.ASSERT” 和控制指令 “.INCLUDE” 的操作数记述本控制指令。
通过本控制指令读到的文件名是不包括文件的扩展名和路径的部分。

给指定的符号设定堆栈值

.STACK

格　式　　.STACK △ < 名称 >=< 数值 >

说　明　　给符号定义 Call Walker 表示的堆栈使用量。

例　　.STACK SYMBOL=100H

备　注　　只能对 1 个符号定义 1 次堆栈值。
如果指定 2 次或者 2 次以上，该定义就无效。能指定的堆栈值为 0H ～ 0FFFFFFFCH 范围内的 4 的倍数，如果指定此范围外的值，该定义就无效。
< 数值 > 为常数值并且不使用前方参照符号、外部参照符号或者相对地址符号进行指定。

行号的变更

.LINE

格　式　　.LINE △ < 文件名 >,< 行号 >
.LINE △ < 行号 >

说　明　　更改汇编程序的错误信息或者调试时参照的行号和文件名。
不更新程序内的第 1 个 .LINE 到下一个 .LINE 之间的行号或者文件名。
在指定调试选项并且输出汇编语言文件时，编译程序生成与 C 语言源文件行对应的 .LINE。
如果省略文件名，就不更改文件名而只更改行号。

例　　.LINE "C:\asm\test.c",5

替换符号的定义

.DEFINE

格　式　　< 符号名 > △ .DEFINE △ < 字符串 >
< 符号名 > △ .DEFINE △ '< 字符串 >'
< 符号名 > △ .DEFINE △ "< 字符串 >"

说　明　　给符号定义字符串。
能重新定义符号。

例　　
```
X_HI    .DEFINE   R1
        MOV.L  #0, X_HI
```

备　注　　在定义含有空白字符或者制表符的字符串时，必须用单引号 （’）或者双引号 （”）括起来记述。
通过本控制指令定义的符号不能指定外部参照。
当通过本控制指令和 .EQU 控制指令声明了同名符号时，先记述的符号优先。

10.3.7 宏控制指令

此控制指令用于定义宏功能和重复宏功能。

表 10.36 宏控制指令

控制指令	功能内容
.MACRO	定义宏名并且定义宏主体的开始。
.EXITM	中断宏主体的展开。
.LOCAL	声明宏内的局部标号。
.ENDM	表示宏主体的结束。
.MREPEAT	表示重复宏主体的开始。
.ENDR	表示重复宏主体的结束。
..MACPARA	将调用宏的实际参数的个数作为值。
..MACREP	将重复宏主体的展开次数作为值。
.LEN	将指定字符串的字符数作为值。
.INSTR	在指定的字符串中，将指定字符串的起始位置作为值。
.SUBSTR	在指定的字符串中，从指定的位置开始抽出指定字符数的字符。

宏定义

.MACRO

格　式　　[宏定义]
△ < 宏名 > △ .MACRO[< 形式参数 >[,...]]
△主体
△ .ENDM
[调用宏]
△ < 宏名 > △ [< 实际参数 >[,...]]

说　明　　定义宏名。
表示宏定义的开始。

例　　• 例 1
[宏定义的例子]

```
name     .MACRO  string
         .BYTE  'string'
         .ENDM
```

[宏调用的例子 1]

```
name     "name,address"

.BYTE    'name,address'
```

[宏调用的例子 2]

```
name     (name,address)

.BYTE    '(name,address)'
```

• 例 2

```
mac                .MACRO    p1,p2,p3
        .IF        ..MACPARA == 3
                   .IF       'p1' == 'byte'
                             MOV.B  #p2,[p3]
                   .ELSE
                             MOV.W  #p2,[p3]
                   .ENDIF
        .ELIF      ..MACPARA == 2
                   .IF       'p1' == 'byte'
                             MOV.B  #p2,[R3]
                   .ELSE
                             MOV.W  #p2,[R3]
                   .ENDIF
        .ELSE
                  MOV.W  R3,R1
        .ENDIF
                   .ENDM
```

```
                mac     word,10,R3    ; 调用宏。

        .IF     3 == 3                ; 宏展开后的代码
        .ELSE
                        MOV.W  #10,[R3]
        .ENDIF
```

备　注　　必须记述宏名。
必须根据“10.1.2 名称”的“名称的记述规则”记述宏名。
必须根据“10.1.2 名称”的“名称的记述规则”记述宏的形式参数名称。
必须用不同的名称定义包括嵌套宏在内的宏的形式参数名称。
在多次定义形式参数时，必须用逗号（,）分开记述形式参数。
控制指令“.MACRO”的操作数记述的形式参数必须在宏主体内记述。
必须在宏名和实际参数之间记述空白字符或者制表符。
在调用宏时，必须记述与形式参数对应的实际参数。
在给实际参数记述特殊字符时，必须用双引号括起来记述。
能给实际参数记述标号、全局标号和符号。
能给实际参数记述表达式。
按照从左到右的记述顺序替换形式参数和实际参数。
如果定义了形式参数而没有通过调用宏记述实际参数，就不输出形式参数部分的代码。
如果形式参数的个数多于实际参数的个数，就不输出没有对应实际参数的形式参数部分的代码。
如果用单引号（'）将主体记述的形式参数括起来，就将对应的实际参数用单引号括起来输出。
如果 1 个实际参数含有逗号（,）并且用括号（()）括起来，就连括号一起转换。
如果实际参数的个数多于形式参数的个数，就不处理没有对应形式参数的实际参数。
用双引号括起来的字符串表示该字符串本身。不能将形式参数用双引号括起来记述。
最多能记述 80 个形式参数。
1 行最多能记述 80 个字符数。
如果实际参数和形式参数的个数不符，汇编程序就输出警告信息。

.EXITM

格　式　　< 宏名 > △ .MACRO
△主体
△ .EXITM
△主体
△ .ENDM

说　明　　中断宏主体的展开，将控制传递给最近的 “.ENDM”。

例

```
data1               .MACRO   value
        .IF         value == 0
                    .EXITM
        .ELSE
                    .BLKB    value
        .ENDIF
        .ENDM

data1               0        ; 调用宏。

.IF                 0 == 0   ; 宏展开后的代码
                    .EXITM
.ENDIF
```

备　注　　必须在宏定义的主体内记述本控制指令。

宏内局部标号的声明

.LOCAL

格　式　　.LOCAL △ < 标号名 >[,...]

说　明　　将操作数记述的标号声明为宏的局部标号。
如果宏局部标号是不同的宏定义或者不是宏定义，就能记述多个相同的名称。

例

```
name    .MACRO
  .LOCAL        m1          ; 'm1' is macro local label
m1:
  nop
  bra   m1
  .ENDM
```

备　注　　必须在宏主体内记述本控制指令。
必须在本控制指令和操作数之间记述空白字符或者制表符。
必须在定义标号名前记述本控制指令声明的宏局部标号。
必须根据 “10.1.2 名称”的 “名称的记述规则”记述宏局部标号名。
能给本控制指令的操作数记述用逗号分开的多个标号。此时，最大标号数为 100 个。
在嵌套宏定义时，宏定义内已定义的宏内的宏局部标号不能使用相同的名称。
能给包括 include 文件内容在内的 1 个汇编语言文件记述最多 65535 个宏局部标号。

宏定义的结束

.ENDM

格　式　　< 宏名 > △ .MACRO
△主体
△ .ENDM

说　明　　表示 1 个宏定义主体的结束。

例

```
lda     .MACRO
        MOV.L     #value,R3
        .ENDM

lda     0         ; 展开为 MOV.L   #0,R3。
```

重复宏的开始

.MREPEAT

格　式　　[< 标号 >:] △ .MREPEAT △ < 数值 >
△主体
△ .ENDR

说　明　　表示重复宏的开始。
重复展开指定次数的主体。
最多能指定 65535 次的重复次数。
最多能嵌套 65535 层。
在记述本控制指令的位置展开宏主体。

例

```
rep     .MACRO   num
        .MREPEAT   num
        .IF          num > 49
        .EXITM
        .ENDIF
        nop
        .ENDR
    .ENDM

    rep   3    ; 调用宏。

    nop        ; 宏展开后的代码
    nop
    nop
```

备　注　　必须记述操作数。
必须在本控制指令和操作数之间记述空白字符或者制表符。
能在本控制指令行的开头记述标号。
能给操作数记述符号。
不能记述前方参照符号。
能给操作数记述表达式。
能给主体记述宏定义和宏调用。
能在主体内记述控制指令 “.EXITM”。

重复宏的结束

.ENDR

格　式　　[< 标号 >:] △ .MREPEAT △ < 数值 >
△主体
△ .ENDR

说　明　　表示重复宏的结束。

备　注　　必须对应控制指令 “.MREPEAT”记述本控制指令。

替换为宏实际参数的个数

..MACPARA

格　式　　..MACPARA

说　明　　表示宏调用的实际参数的个数。
能在 “.MACRO”定义的宏主体内记述本控制指令。

例　　判断宏实际参数的个数，进行条件汇编。

```
        .GLB    mem
name    .MACRO  f1,f2
        .IF     ..MACPARA == 2
        ADD  f1,f2
        .ELSE
        ADD  R3,f1
        .ENDIF
        .ENDM

name  mem    ; 调用宏。

.ELSE        ; 宏展开后的代码
ADD  R3,mem
.ENDIF
```

备　注　　能将本控制指令记述为表达式的项。
如果不给 “.MACRO”定义的宏主体记述本控制指令，值就为 0。

替换为当前宏的重复次数

..MACREP

格　式　　..MACREP

说　明　　表示重复宏的展开次数。
能在 “.MREPEAT” 定义的宏主体内记述本控制指令。
能给条件汇编程序的操作数记述本控制指令。

例

```
mac     .MACRO  value,reg
        .MREPEAT  value
        MOV.B   #0,..MACREP[reg]
        .ENDR
        .ENDM

        mac     3,R3        ; 宏调用

        .MREPEAT  3         ; 宏展开后的代码
        MOV.B   #0,1[R3]
        MOV.B   #0,2[R3]
        MOV.B   #0,3[R3]
        .ENDR
        .ENDM
```

备　注　　能将本控制指令记述为表达式的项。
如果不给 “.MACRO” 定义的宏主体记述本控制指令，值就为 0。

替换为指定的字符串长度

.LEN

格　式　　.LEN △ {"< 字符串 >"}
.LEN △ {'< 字符串 >'}

说　明　　表示操作数记述的字符串长度。

例

```
bufset   .MACRO  f1
buffer:  .BLKB   .LEN{'f1'}
                 .ENDM

         bufset  Sample     ; 调用宏。

buffer:          .BLKB   6  ; 宏展开后的代码
```

备　注　　必须用 {} 将操作数括起来记述。
能在本控制指令和操作数之间记述空白字符或者制表符。
能给字符串记述含有空白字符和制表符的字符。
必须用单引号或者双引号将字符串括起来记述。
能给表达式的项记述本控制指令。
在计算宏实际参数的字符串长度时，必须用单引号将形式参数名括起来记述。如果用双引号括起来记述，就表示作为形式参数指定的字符串长度。

替换为字符串的起始位置

.INSTR

格　式　　.INSTR △{ "<字符串>","<检索字符串>",<检索开始位置> }
.INSTR △{ '<字符串>','<检索字符串>',<检索开始位置> }

说　明　　表示在操作数指定的字符串中检索字符串的起始位置。
能指定开始检索字符串的位置。

例　　从指定的字符串（japanese）的开头（top）取出“se”字符串的位置(7)。

```
top     .EQU   1
point_set       .MACRO  source,dest,top
point   .EQU    .INSTR{'source','dest',top}
        .ENDM
 point_set  japanese,se,1   ; 调用宏。

 point        .EQU   7      ; 宏展开后的代码
```

备　注　　必须用{}将操作数括起来记述。
必须记述字符串、检索字符串和检索开始位置。
必须用逗号分开记述字符串、检索字符串和检索开始位置。
不能在逗号的前后记述空白字符和制表符。
能给检索开始位置记述符号。
如果检索开始位置为1，就表示字符串的开头。
能给表达式的项记述本控制指令。
当检索字符串长于字符串时，值为0；当字符串中不含检索字符串时，值为0；当检索开始位置的值大于字符串长度时，值为0。
要将宏的实际参数作为检测条件展开宏时，必须用单引号将形式参数名括起来记述。如果用双引号括起来记述，就将该字符串作为检测条件进行宏展开。

字符串的抽出

.SUBSTR

格　式　　.SUBSTR △ { "< 字符串 >",< 抽出开始位置 >,< 抽出字符数 > }
.SUBSTR △ { '< 字符串 >',< 抽出开始位置 >,< 抽出字符数 > }

说　明　　从字符串的指定位置开始抽出所指定的字符串。

例　　将宏实际参数的字符串长度提供给 “.MREPEAT” 的操作数。
每展开一次 “.BYTE” 的行，“..MACREP” 就按 1→2→3→4 增加。
因此，从宏实际参数的字符串的开头开始按顺序给 “.BYTE” 的操作数逐个提供字符。

```
name    .MACRO  data
        .MREPEAT   .LEN{'data'}
        .BYTE  .SUBSTR{'data',..MACREP,1}
        .ENDR
        .ENDM

        name  ABCD      ; 调用宏。

        .BYTE  "A"      ; 宏展开后的代码
        .BYTE  "B"
        .BYTE  "C"
        .BYTE  "D"
```

备　注　　必须用 {} 将操作数括起来记述。
必须记述字符串、抽出开始位置和抽出字符数。
必须用逗号分开记述字符串、抽出开始位置和抽出字符数。
能给抽出开始位置和抽出字符数记述符号。如果抽出开始位置为 1，就表示字符串的开头。
能给字符串记述含有空白字符和制表符的字符。
必须用单引号或者双引号将字符串括起来记述。
当抽出开始位置的值大于字符串长度时，值为 0；当抽出字符数的值大于字符串长度时，值为 0；当抽出字符数为 0 时，值为 0。
要将宏的实际参数作为抽出条件进行宏展开时，必须用单引号将形式参数名括起来记述。如果用双引号括起来记述，就将该字符串作为抽出条件进行宏展开。

10.3.8 编译程序的专用控制指令

在编译程序生成汇编语言源文件时，为了让汇编程序对 C 语言的功能进行适当的处理，有时输出下述控制指令。

在使用编译程序生成的汇编语言源文件时，请不要更改这些控制指令的内容而直接使用。在建立用户汇编程序时，不能使用这些控制指令。

表 10.37 编译程序的专用控制指令

控制指令	内容
._LINE_TOP	当展开了 #pragma inline_asm 指定的函数时，输出这些控制指令。
._LINE_END	
.SWSECTION	当 switch 语句使用了转移表时，输出这些控制指令。
.SWMOV	
.SWITCH	

11. 编译程序的错误信息

11.1 错误格式和错误级

本章节说明用以下格式输出的错误信息和错误内容。

错误号 (错误级) 错误信息
错误内容

根据错误的重要程度，错误级分为 5 种。

错误级		运行
(I)	信息	继续处理。
(W)	警告	继续处理。
(E)	错误	继续选项的解析处理，中止处理。
(F)	致命错误	中止处理。
(-)	内部错误	中止处理。

11.2 信息一览

C0005 (I) Precision lost
对于赋值表达式，在将右边表达式的值转换为左边的型时，有可能失去精度。

C0006 (I) Conversion in argument
将函数的参数表达式转换为函数原型指定的参数型。

C0008 (I) Conversion in return
将返回语句的表达式转换为函数返回值的型。

C0011 (I) Used before set symbol : " 变量名 " in " 函数名 "
参照了没有设定值的局部变量。

C0101 (I) Optimizing range divided in function " 函数名 "
“函数名”的优化范围被分割成多个。

C0102 (I) Register is not allocated to " 变量名 " in " 函数名 "
不能给有 register 存储类的变量分配寄存器。

C1026 (W) Address of packed member
取得了指定 pack=1 的结构体成员地址。

C1300 (W) Command parameter specified twice
至少指定了 2 次相同的编译程序选项。在相同的编译程序选项中，最后指定的选项有效。

C1301 (W) " 选项 " option ignored
忽视 “选项” 进行编译。

C1308 (W) Duplicate number specified in option "选项" : "序号"
在 “选项”中指定了相同的序号。

C1309 (W) Section name "SI" or "SU" specified
给 “段名”指定了 “SI”或者 “SU”。用指定的段名进行输出。

C1315 (W) File_inline "文件名" ignored by same file as source file
通过 file_inline 选项指定了编译对象文件。忽视 file_inline 选项进行编译。

C1316 (W) "相应宏" is not a valid predefined macro name
宏名 “<宏名>”不是预定义宏。将 undefine 选项的指定置为无效。

C1402 (W) #pragma section ignored
忽视 #pragama section 的指定。

C1410 (W) A struct/union/class has different pack specifications
在 1 个结构体、联合体或者类中存在不同的 pack 值。

C1600 (W) Debugging information describing location of "名称" is lost
不输出 “名称”的符号信息。

C1800 (W) Variable "变量名" type mismatch in files
在文件之间，用 “变量名”表示的变量型不同。
不能指定 file_inline 选项。

C1801 (W) Using function at influence the code generation of "NC" compiler
使用了对和 NC 编译程序的兼容性有影响的功能。

C1802 (W) Using function at influence the code generation of "H8" compiler
使用了对和 H8 编译程序的兼容性有影响的功能。

C1803 (W) Address taken "变量名". It may cause an upset endian indirect reference
取得了和 endian 选项不同字节序的 8 字节变量 “变量名”的地址。有可能引起字节序处理错误的间接参照。

C1804 (W) Using incompatible int type
因为在进行 C++ 编译时 int_to_short 选项无效，所以在 C++ 编译和 C 编译中 int 型的大小不同。当有可能在 C++ 程序中参照 C 程序的外部名时，输出此信息。

C1950 (W) Nothing to compile, assemble or link (input and output combination)
不需要进行编译、汇编或者连接处理。请确认输入文件的构成和 output 选项指定的组合。
Ignored argument(s)：以下显示不进行处理的一览表。

C2021 (E) Invalid number specified in option "选项" : "序号"
通过 “选项”指定了无效值。请确认值的范围。

C2022 (E) Error level message cannot be changed : "change_message"
不能更改 Error 级信息的信息级。

C2023 (E) Same register is used at base option.
给 base 选项的不同区域指定了相同的寄存器。

C2024 (E) Base register is already used at fint_register option.
通过 base 选项指定了被 fint_register 选项禁止使用的寄存器。

C2025 (E) Base option address constant overflow
base 选项的地址值超出了 0x00000000 ～ 0xffffffff 的范围。

C2026 (E) Illegal register of base option
base 选项的寄存器号有误。指定了 R8 ～ R13 以外的寄存器。

C2027 (E) Cannot read specified file " 文件名 "
不能正常地读文件。请确认文件的指定是否有误。

C2203 (E) Illegal member reference for "."
运算符 “.” 左侧表达式的型不是结构体型或者联合体型。

C2240 (E) Illegal section naming
段的命名有误。给不同用途的段起了相同的名称。

C2450 (E) Illegal #pragma option declaration
#pragma option 声明有误。

C2700 (E) Function " 函数名 " in #pragma interrupt already declared
中断函数声明 #pragma interrupt 指定的函数已被声明为一般函数。

C2701 (E) Multiple interrupt for one function
对 1 个函数重复声明了中断函数声明 #pragma interrupt。

C2703 (E) Illegal #pragma interrupt declaration
中断函数声明 #pragma interrupt 的规格有误。

C2704 (E) Illegal reference to interrupt function
非法参照了中断函数。

C2710 (E) Section name too long
指定段名的字符数超过了极限值。

C2711 (E) Section name table overflow
指定段的个数超过了极限值。

C2714 (E) Usable stack area overflow

在存取堆栈时，因为存取了 SP 相对寻址中不能参照的范围，所以不能生成指令。
可能给数组的下标设定了负数或者自变量区域太大。请检查源码记述。

C2800 (E) Illegal parameter number in in-line function

在内部函数中使用的参数个数不同。

C2801 (E) Illegal parameter type in in-line function

在内部函数中参数的型不同。

C2802 (E) Parameter out of range in in-line function

在内部函数中参数大小超过了能指定的范围。

C2803 (E) Invalid offset value in in-line function

在内部函数中参数的指定不恰当。

C2806 (E) Multiple #pragma for one function

对 1 个函数指定了多种矛盾的 #pragma。

C2831 (E) Multiple #pragma entry declaration

存在多个 #pragma entry 声明。

C2833 (E) Multiple #pragma stacksize declaration

存在多个指定 “si” 或者 “su” 的 #pragma stacksize 声明。

C2854 (E) Illegal address in #pragma address

指定的地址出现了以下任意一种情况：
(1) 给不同的变量指定了相同的地址。
(2) 对于不同的变量，变量的地址重叠。

C2860 (E) Missing #pragma oscall for " 函数名 "

没有函数 “服务调用名” 所需的 #pragma oscall 指定。

C3009 (F) String literal too long

字符串的字符数超过了极限值。字符串的字符数是指连接连续指定的字符串之后的字节数。此字符串的字符数不是源程序中的长度，而是包含在字符串数据中的字节数，转义序列也计为 1 个字符。

C3019 (F) Cannot open source file " 文件名 "

不能打开源文件。

C3020 (F) Source file input error " 文件名 "

不能读源文件或者 include 文件。

C3021 (F) Memory overflow

不能分配编译程序内部使用的存储区。

C3023 (F) Type nest too deep
限定基本型的型 （指针型、数组型、函数型）的个数超过了极限值。

C3024 (F) Array dimension too deep
数组的维数超过了极限值。

C3025 (F) Source file not found
在命令行中没有指定源文件名。

C3030 (F) Too many compound statements
在 1 个函数中复合语句的个数超过了极限值。

C3031 (F) Data size overflow
数组或者结构体的大小超过了极限值。

C3203 (F) Assembly source line too long
输出的汇编源程序的 1 行过长。

C3204 (F) Illegal stack access
函数内使用的堆栈大小 （包含局部变量区、寄存器保存区以及其他函数调用的参数进栈区等）或者该函数调用的参数区超过了 2G 字节。

C3300 (F) Cannot open internal file
可能发生了以下某种错误：
(1) 不能打开编译程序内部生成的中间文件。
(2) 已存在和中间文件同名的文件。
(3) 不能打开编译程序内部使用的文件。

C3301 (F) Cannot close internal file
不能关闭编译程序内部生成的中间文件。请确认编译程序的安装步骤是否有误。

C3302 (F) Cannot input internal file
不能读编译程序内部生成的中间文件。请确认编译程序的安装步骤是否有误。

C3303 (F) Cannot output internal file
不能写编译程序内部生成的中间文件。请增大磁盘空间的容量。

C3304 (F) Cannot delete internal file
不能删除编译程序内部生成的中间文件。请确认是否正在存取编译程序生成的中间文件。

C3305 (F) Invalid command parameter " 选项 "
编译程序选项的指定方法有误。

C3306 (F) Interrupt in compilation
在编译处理过程中从标准输入终端检测到 (Ctrl)+C 命令的中断。

C3307 (F) Compiler version mismatch
构成编译程序的文件与文件的版本不同。请参照安装指南的组装方法，重新安装编译程序。

C3308 (F) Cannot create file " 文件名 "
不能建立编译程序生成的文件。

C3320 (F) Command parameter buffer overflow
命令行的指定超过了 4096 个字符。

C3321 (F) Illegal environment variable
在以下某种情况下发生了错误：
(1) 没有设定环境变量 BIN_RX。
(2) 没有给环境变量 BIN_RX 指定编译程序的执行文件路径名。
(3) 在设定环境变量 BIN_RX 时，指定了违反规约的文件名或者路径名的长度超过了 118 个字符。
(4) 给环境变量 CPU_RX 设定了 “RX600” 以外的内容。

C3322 (F) Lacking cpu specification
没有指定 CPU。必须通过 cpu 选项或者环境变量 CPU_RX 指定 CPU。

C3900 (E) Input file not found. – " 文件名 "
没有指定的输入文件名。

C3901 (E) Input file read error. – " 文件名 "
输入文件发生了读错误。

C3902 (E) Invalid file name. – " 文件名 "
给输入文件名指定了不能使用的字符。

C3903 (E) Invalid option. – " 选项的指定 "
选项的指定不正确。

C3905 (E) Cannot build temporary file.
不能建立临时文件。请确认编译程序的环境设定是否有问题。

C3906 (E) Memory overflow.
编译程序使用的存储器容量不够。

C3907 (E) Tool execute error.
编译程序、汇编程序或者优化连接编辑程序的启动失败。

C3908 (E) Cannot delete temporary file.
不能删除临时文件。请确认编译程序的环境设定是否有问题。

C4000-C4999 (-) Internal error
在编译程序的内部处理过程中发生了某种故障。请和购入本编译程序的营业部门或者代理店联系，反映错误的发生情况。

C5001 (E) Last line of file ends without a newline
文件的最后 1 行的末尾没有换行字符。

C5002 (E) Last line of file ends with a backslash
文件的最后 1 行的末尾没有反斜线符号。

C5003 (F) #include file " 文件名 " includes itself
包含了 “文件名” 的自身文件。

C5004 (F) Out of memory
编译所需的存储器容量不够。请增大系统的存储器容量或者结束其他应用程序。

C5005 (F) Could not open source file " 名称 "
不能打开 “名称” 的文件。请确认文件名是否正确。

C5006 (E) Comment unclosed at end of file
没有指定注释的结束 “*/”。

C5007 (E) (I) Unrecognized token
有不能识别的字句 （在宏时为 (I)）。

C5008 (E) (I) Missing closing quote
没有指定字符串的结束 “"”（在宏时为 (I)）。

C5009 (I) Nested comment is not allowed
“/* */” 注释有嵌套。

C5010 (E) "#" not expected here
在行的开头或者预处理器外指定了 “#”。

C5011 (E) (W) Unrecognized preprocessing directive
有不能识别的预处理器的关键字。

C5012 (E) (W) Parsing restarts here after previous syntax error
重新开始解析字句。

C5013 (E) (F) Expected a file name
需要文件名。 #include 语句为 (F)， #line 语句为 (E)。

C5014 (E) Extra text after expected end of preprocessing directive
在预处理器语句之后记述了文本。

C5016 (F) " 名称 " is not a valid source file name
“名称”的文件无效。

C5017 (E) Expected a "]"
没有 “]”。

C5018 (E) Expected a ")"
没有 “)”。

C5019 (E) Extra text after expected end of number
在数值之后记述了文本。

C5020 (E) Identifier " 名称 " is undefined
没有定义 “名称”的符号。

C5021 (W) Type qualifiers are meaningless in this declaration
指定了没有含义的型限定词。将型限定词置为无效。

C5022 (E) Invalid hexadecimal number
16 进制数的记述有误。

C5023 (E) Integer constant is too large
整数常数的值太大。

C5024 (E) Invalid octal digit
8 进制数的记述有误。

C5025 (E) Quoted string should contain at least one character
字符常数为空。

C5026 (E) Too many characters in character constant
字符常数中的字符数太多。

C5027 (W) Character value is out of range
字符的值超出了范围。舍去超过的值。

C5028 (E) Expression must have a constant value
表达式的值不是常数。

C5029 (E) Expected an expression
需要表达式。

C5030 (E) Floating constant is out of range
浮点型的值超出了范围。

C5031 (E) (W) Expression must have integral type
表达式的型必须是整数型。

C5032 (E) Expression must have arithmetic type
表达式的型必须是算术型。

C5033 (E) Expected a line number
#line 语句需要行号。

C5034 (E) Invalid line number
#line 语句的行号无效。

C5035 (F) #error directive: " 行号 "
使用了 #error 语句。

C5036 (E) The #if for this directive is missing
#if 语句的指定方法有误。

C5037 (E) The #endif for this directive is missing
#endif 行的指定方法有误。

C5038 (E) (W) Directive is not allowed -- an #else has already appeared
已出现 #else 语句。忽视此指定。

C5039 (E) (W) Division by zero
发生了被零除。

C5040 (E) Expected an identifier
需要标识符。

C5041 (E) Expression must have arithmetic or pointer type
表达式的型必须是算术型或者指针型。

C5042 (E) (W) Operand types are incompatible (" 型 1" and " 型 2")
“型 1”和 “型 2”的操作数的型不同。

C5044 (E) Expression must have pointer type
表达式的型必须是指针型。

C5045 (W) #undef may not be used on this predefined name
不能取消系统定义的宏名。将 #undef 的指定置为无效。

C5046 (W) “宏名” is predefined; attempted redefinition ignored
不能重新定义系统定义的宏名。将 #define 的指定置为无效。

C5047 (W) Incompatible redefinition of macro "名称" (declared at line "行号")
“名称”宏的重新定义和前面的定义不同。将重新定义的宏置为有效。

C5049 (E) Duplicate macro parameter name
重复定义了宏的参数名。

C5050 (E) "##" may not be first in a macro definition
在 #define 宏的最初位置指定了 “##”。

C5051 (E) "##" may not be last in a macro definition
在 #define 宏的结束位置指定了 “##”。

C5052 (E) Expected a macro parameter name
在 “#” 之后没有宏参数。

C5053 (E) Expected a ":"
需要 “:”。

C5054 (W) Too few arguments in macro invocation
展开宏时的实际参数不够。

C5055 (W) Too many arguments in macro invocation
展开宏时的实际参数太多。

C5056 (E) Operand of sizeof may not be a function
不能给 sizeof 运算的操作数指定函数。

C5057 (E) This operator is not allowed in a constant expression
不能在常数表达式中指定此运算符。

C5058 (E) This operator is not allowed in a preprocessing expression
不能在预处理器的表达式中指定此运算符。

C5059 (E) Function call is not allowed in a constant expression
不能在常数表达式中调用函数。

C5060 (E) This operator is not allowed in an integral constant expression
不能在整数型常数表达式中指定此运算符。

C5061 (W) Integer operation result is out of range
整数运算结果超出了值的范围。假设为忽视上溢高位的值。

C5062 (W) Shift count is negative
移位计数是负值。按指定的内容进行运算。

C5063 (W) Shift count is too large
移位计数超过了有效位数。按指定的内容进行运算。

C5064 (W) Declaration does not declare anything
没有指定声明的符号。忽视声明。

C5065 (E) Expected a ";"
需要“;”。

C5066 (E) Enumeration value is out of "int" range
枚举型成员的值超出了 int 型的范围。

C5067 (E) Expected a "}"
需要“}”。

C5068 (W) Integer conversion resulted in a change of sign
进行了有符号转换的整数型转换。照样设定位串。

C5069 (W) Integer conversion resulted in truncation
进行了舍去高位字节的整数型转换。设定舍去高位字节后的值。

C5070 (E) Incomplete type is not allowed
指定了不完全的型。

C5071 (E) Operand of sizeof may not be a bit field
给 sizeof 运算符的操作数指定了位字段。

C5075 (E) Operand of "*" must be a pointer
“*”运算符操作数的型不是指针型。

C5076 (W) Argument to macro is empty
没有给函数宏指定参数。

C5077 (E) This declaration has no storage class or type specifier
没有指定存储类或者型。

C5078 (E) A parameter declaration may not have an initializer
不能给参数的声明指定初始表达式。

C5079 (E) Expected a type specifier
需要型说明符。

C5080 (E) (W) A storage class may not be specified here
在此不能指定存储类。

C5081 (E) More than one storage class may not be specified
不能指定多个存储类。

C5082 (W) Storage class is not first
没有在数据型前指定存储类。

C5083 (W) Type qualifier specified more than once
指定了多个 const/volatile 限定词。忽视多余的指定。

C5084 (E) Invalid combination of type specifiers
型的组合不正确。

C5085 (W) Invalid storage class for a parameter
给形式参数指定了非法的存储类。

C5086 (E) Invalid storage class for a function
给函数指定了非法的存储类。

C5087 (E) A type specifier may not be used here
不能指定型。

C5088 (E) Array of functions is not allowed
不能指定以函数为元素的数组。

C5089 (E) Array of void is not allowed
不能指定以 void 型为元素的数组。

C5090 (E) Function returning function is not allowed
不能指定以函数型为返回型的函数。

C5091 (E) Function returning array is not allowed
不能指定以数组为返回型的函数。

C5092 (E) Identifier-list parameters may only be used in a function definition
不能在函数定义以外的位置使用标识符列表参数。

C5093 (E) Function type may not come from a typedef
不能使用进行了 typedef 声明的函数型。

C5094 (E) The size of an array must be greater than zero
数组的值必须大于 0。

C5095 (E) Array is too large
数组太大。

C5096 (W) A translation unit must contain at least one declaration
翻译单位内至少需要 1 个声明。

C5097 (E) A function may not return a value of this type
函数不能返回此型的值。

C5098 (E) An array may not have elements of this type
不能将此型作为数组的元素。

C5099 (E) (W) A declaration here must declare a parameter
此函数声明需要声明参数。

C5100 (E) Duplicate parameter name
形式参数的名称有重复。

C5101 (E) " 名称 " has already been declared in the current scope
在相同作用域中已存在 “名称” 的声明。

C5102 (E) Forward declaration of enum type is nonstandard
enum 型的前方声明不是标准格式。

C5103 (E) Class is too large
类太大。

C5104 (E) Struct or union is too large
结构体或者联合体太大。

C5105 (E) Invalid size for bit field
这是非法的位字段的大小。

C5106 (E) Invalid type for a bit field
这是非法的位字段的型。

C5107 (E) (W) Zero-length bit field must be unnamed
不能给长度为 0 的位字段命名。

C5108 (W) Signed bit field of length 1
指定了带符号整数型的长度为 1 的位字段。用指定的型进行处理。

C5109 (E) Expression must have (pointer-to-) function type
表达式的型必须是指向函数型的指针型。

C5110 (E) Expected either a definition or a tag name
需要声明的定义或者标记符名。

C5111 (W) Statement is unreachable
这是不被执行的语句，有可能通过优化删除。

C5112 (E) Expected "while"
需要 “while” 关键字。

C5114 (E) (W) Entity-kind " 名称 " was referenced but not defined
没有定义被参照的 “名称”。

C5115 (E) A continue statement may only be used within a loop
continue 语句只在循环语句中有效。

C5116 (E) A break statement may only be used within a loop or switch
break 语句只在循环语句或者 switch 语句中有效。

C5117 (W) Non-void entity-kind " 名称 " should return a value
非 void 型函数没有返回值。返回值为不定值。

C5118 (E) A void function may not return a value
返回 void 型的函数不能有返回值。

C5119 (E) Cast to type " 型 " is not allowed
不能指定指向 “型” 的型转换。

C5120 (E) Return value type does not match the function type
返回值和函数的型不同。

C5121 (E) A case label may only be used within a switch
在 switch 语句外使用了 case 标号。

C5122 (E) A default label may only be used within a switch
在 switch 语句外使用了 default 标号。

C5123 (E) Case label value has already appeared in this switch
switch 语句中已存在 case 标号的值。

C5124 (E) Default label has already appeared in this switch
switch 语句中已存在 default 标号的值。

C5125 (E) Expected a "("
需要 “(”。

C5126 (E) Expression must be an lvalue
表达式必须是左边值。

C5127 (E) Expected a statement
需要语句。

C5128 (W) Loop is not reachable from preceding code
这是不被执行的循环语句。

C5129 (E) A block-scope function may only have extern storage class
在块内声明的函数必须是 extern 存储类。

C5130 (E) Expected a "{"
需要 “{”。

C5131 (E) Expression must have pointer-to-class type
表达式的型必须是指向类的指针型。

C5132 (E) Expression must have pointer-to-struct-or-union type
表达式的型必须是指向结构体或者联合体的指针型。

C5133 (E) Expected a member name
需要成员名。

C5134 (E) Expected a field name
需要字段名。

C5135 (E) Entity-kind " 名称 " has no member " 成员名 "
“名称”没有 “成员名”。

C5136 (E) Entity-kind " 名称 " has no field " 字段名 "
“名称”没有 “字段名”。

C5137 (E) (W) Expression must be a modifiable lvalue
表达式必须是能修正的左边值。

C5138 (E) (W) Taking the address of a register field is not allowed
不能参照寄存器字段的地址。

C5139 (E) Taking the address of a bit field is not allowed
不能参照位字段的地址。

C5140 (E) (W) Too many arguments in function call
函数调用的实际参数个数太多。

C5141 (E) Unnamed prototyped parameters not allowed when body is present
被定义函数的原型声明的参数没有名称。

C5142 (E) Expression must have pointer-to-object type
表达式的型必须是指向目标的指针型。

C5143 (F) Program too large or complicated to compile
程序太大或者太复杂。

C5144 (E) A value of type "型1" cannot be used to initialize an entity of type "型 2"
初始值的 “型 1”和变量的 “型 2”不同。

C5145 (E) Entity-kind "名称" may not be initialized
不能对 “名称”进行初始化。

C5146 (E) Too many initializer values
初始值的个数太多。

C5147 (E) (W) Declaration is incompatible with "名称" (declared at line "行号")
和前面声明的 “名称”的型不同。

C5148 (E) Entity-kind "名称" has already been initialized
已设定 “名称”的初始值。

C5149 (E) A global-scope declaration may not have this storage class
不能在全局作用域的声明中指定此存储类。

C5150 (E) A type name may not be redeclared as a parameter
不能在形式参数中重新声明型名。

C5151 (E) A typedef name may not be redeclared as a parameter
不能在形式参数中重新声明型名。

C5152 (W) Conversion of nonzero integer to pointer
要将零以外的整数转换为指针。

C5153 (E) Expression must have class type
表达式的型必须是类（class）型。

C5154 (E) Expression must have struct or union type
表达式的型必须是结构体型或者联合体型。

C5155 (W) Old-fashioned assignment operator
使用了旧式的赋值运算符。

C5156 (W) Old-fashioned initializer
使用了旧式的初始表达式。

C5157 (E) (W) Expression must be an integral constant expression
表达式必须是整数型常数表达式。

C5158 (E) Expression must be an lvalue or a function designator
表达式必须是左边值或者函数名。

C5159 (E) Declaration is incompatible with previous "名称" (declared at line "行号")
和前面使用的 “名称” 的型不同。

C5160 (E) Name conflicts with previously used external name "名称"
前面使用的外部名 “名称” 和名称有重复。

C5161 (W) Unrecognized #pragma
指定了不能识别的 #pragma。忽视 #pragma 的指定。

C5163 (F) Could not open temporary file "名称"
不能打开 “名称” 的临时文件。请确认编译程序的环境设定和主机环境的文件系统是否异常。

C5164 (F) Name of directory for temporary files is too long ("名称")
临时文件的 “名称” 太长。

C5165 (E) Too few arguments in function call
函数调用的实际参数个数不够。

C5166 (E) Invalid floating constant
这是非法的浮点常数的指定。

C5167 (E) Argument of type "型1" is incompatible with parameter of type "型2"
实际参数的型 “型 1” 和形式参数的型 “型 2” 不同。

C5168 (E) A function type is not allowed here
不允许函数型。

C5169 (E) (W) Expected a declaration
需要声明。

C5170 (W) Pointer points outside of underlying object
指针指向的区域超出了目标的范围。

C5171 (E) Invalid type conversion
这是非法的型转换的型。

C5172 (W) (I) External/internal linkage conflict with previous declaration
前面的声明和外部 / 内部连接不同。假设内部连接。

C5173 (E) (W) Floating-point value does not fit in required integral type
在将浮点型的值转换为整数型时，超出了值的范围。

C5174 (I) Expression has no effect
这是没有效果的表达式。有可能通过优化删除。

C5175 (E) (W) Subscript out of range
数组的下标超出了范围。用指定的下标继续处理。

C5177 (W) Entity-kind " 名称 " was declared but never referenced
有不被参照的声明。

C5178 (W) "&" applied to an array has no effect
数组名的前头有 “&”。忽视 “&”。

C5179 (W) Right operand of "%" is zero
“%” 运算符的右边值是 0。用指定的表达式进行评价。

C5180 (W) (I) Argument is incompatible with formal parameter
参数和旧式的参数不同。

C5181 (W) Argument is incompatible with corresponding format string conversion
参数和对应的字符串转换格式不同。

C5182 (F) Could not open source file " 名称 " (no directories in search list)
不能打开 “名称” 的文件。请确认文件夹是否存在。

C5183 (E) Type of cast must be integral
型转换的型必须是整数型。

C5184 (E) Type of cast must be arithmetic or pointer
型转换的型必须是算术型或者指针型。

C5185 (I) Dynamic initialization in unreachable code
不执行初始化表达式。在执行时不设定初始值。

C5186 (W) Pointless comparison of unsigned integer with zero
对 0 和无符号的整数进行了无意义的比较。按指定的内容评价表达式。

C5187 (I) Use of "=" where "==" may have been intended
“==” 是有意图的表达式，使用了 “=”。按指定的内容评价表达式。

C5188 (W) Enumerated type mixed with another type
枚举型被转换为其他枚举型或者数据型。

C5189 (F) Error while writing "文件名" file
写文件失败。

C5190 (F) Invalid intermediate language file
这是非法的中间语言文件。

C5191 (W) Type qualifier is meaningless on cast type
给型转换的型指定了无含义的型限定词。忽视指定的型。

C5192 (W) Unrecognized character escape sequence
指定了不能识别的转义序列字符。照样使用该值。

C5193 (I) Zero used for undefined preprocessing identifier
将 0 用于预处理器语句的表达式的评价。按指定的内容评价表达式。

C5194 (E) Expected an asm string
需要 asm 字符串。

C5195 (E) An asm function must be prototyped
需要对 asm 函数进行原型声明。

C5196 (E) An asm function may not have an ellipsis
不能在 asm 函数的参数中使用省略符号 （...）

C5219 (F) Error while deleting file "文件名"
不能删除 “文件名” 的文件。

C5220 (E) Integral value does not fit in required floating-point type
不能将整数值转换为要求的浮点型。

C5221 (E) Floating-point value does not fit in required floating-point type
不能将浮点型转换为要求的浮点型。视为无穷大的值。

C5222 (E) Floating-point operation result is out of range
浮点运算结果超出了值的范围。假设忽视上溢高位的值。

C5223 (W) Function 函数名 declared implicitly
隐含声明了函数。

C5224 (W) The format string requires additional arguments
实际参数的个数少于格式字符串中要求的参数。

C5225 (W) The format string ends before this argument
实际参数的个数多于格式字符串中要求的参数。

C5226 (W) Invalid format string conversion
格式转换的格式和实际参数的型不同。

C5227 (E) Macro recursion
递归的宏的展开级超过 300 层。

C5228 (W) Trailing comma is nonstandard
在列表的最后元素值之后加了逗号，这不是标准格式。

C5229 (W) Bit field cannot contain all values of the enumerated type
位字段不能保持全部枚举型的值。舍去结果。

C5230 (W) Nonstandard type for a bit field
位字段使用了非标准格式的数据型。

C5231 (W) Declaration is not visible outside of function
从函数外看不到函数原型声明内的型声明。

C5232 (W) Old-fashioned typedef of "void" ignored
旧式的 void 的 typedef 无效。

C5233 (W) Left operand is not a struct or union containing this field
给左操作数的结构体或者联合体指定了不存在的字段。

C5234 (W) Pointer does not point to struct or union containing this field
给指针指向的结构体或者联合体指定了不存在的字段。

C5235 (E) Variable " 名称 " was declared with a never-completed type
声明了不完全型的 “名称” 变量。

C5236 (W) (I) Controlling expression is constant
控制表达式是常数 (I)，控制表达式是地址常数 (W)。按指定的内容评价表达式。

C5237 (I) Selector expression is constant
switch 语句的控制表达式是常数。

C5238 (E) Invalid specifier on a parameter
在参数声明中使用了非法的说明符。

C5239 (E) Invalid specifier outside a class declaration
在类声明以外的位置使用了非法的说明符。

C5240 (E) Duplicate specifier in declaration
在 1 个声明内重复使用了说明符。

C5241 (E) A union is not allowed to have a base class
union 型不能有基类。

C5242 (E) Multiple access control specifiers are not allowed
重复使用了存取说明符。

C5243 (E) Class or struct definition is missing
没有对应 class 定义的括弧。

C5244 (E) Qualified name is not a member of class "型" or its base classes
限定名不是类或者基类成员的 “型”。

C5245 (E) A nonstatic member reference must be relative to a specific object
非静态成员的参照没有对应目标。

C5246 (E) A nonstatic data member may not be defined outside its class
不能在类的外面定义非静态数据成员。

C5247 (E) Entity-kind "名称" has already been defined
已定义了 “名称”。

C5248 (E) Pointer to reference is not allowed
不允许指向参照型的指针型。

C5249 (E) Reference to reference is not allowed
不允许指向参照型的参照型。

C5250 (E) Reference to void is not allowed
不允许指向 void 型的参照型。

C5251 (E) Array of reference is not allowed
不允许参照型数组。

C5252 (E) Reference entity-kind "名称" requires an initializer
参照型的定义 “名称” 需要初始值。

C5253 (E) Expected a ","
需要逗号 “,”。

C5254 (E) Type name is not allowed
不允许型名。

C5255 (E) Type definition is not allowed
不允许型的定义。

C5256 (E) Invalid redeclaration of type name "名称" (declared at line "行号")
不能重新定义 “名称” 的型名。

C5257 (E) Const entity-kind "名称" requires an initializer
const 型的定义 “名称” 需要初始值。

C5258 (E) "this" may only be used inside a nonstatic member function
在非静态成员函数外使用了 “this”。

C5259 (E) Constant value is not known
const 型的值不明确。

C5260 (W) Explicit type is missing ("int" assumed)
没有指定型。假设为 int 型。

C5261 (I) Access control not specified ("名称" by default)
没有指定基类的存取控制。假设指定存取控制为 “名称”。

C5262 (E) (W) Not a class or struct name
没有基类指定的类或者结构体。

C5263 (E) Duplicate base class name
重复指定了基类。

C5264 (E) Invalid base class
这是非法的基类。

C5265 (E) Entity-kind "名称" is inaccessible
不能存取 “名称”。

C5266 (E) "名称" is ambiguous
指定的 “名称” 不明确。

C5268 (E) Declaration may not appear after executable statement in block
声明不在块的执行语句之后。

C5269 (E) Conversion to inaccessible base class "型" is not allowed
无法转换为不能参照的基类的 “型”。

C5274 (E) Improperly terminated macro invocation
在调用宏的过程中文件结束。

C5276 (E) Name followed by "::" must be a class or namespace name
“::” 之后的名称必须是类名或者 namespace 名。

C5277 (E) Invalid friend declaration
友元声明的指定不正确。

C5278 (E) A constructor or destructor may not return a value
构造函数和析构函数没有返回值。

C5279 (E) Invalid destructor declaration
析构函数的声明不正确。

C5280 (E) (W) Declaration of a member with the same name as its class
声明了和类名同名的成员名。
(W) 非 static 变量名
(E) static 变量名、 typedef 名和 enum 成员等

C5281 (E) Global-scope qualifier (leading "::") is not allowed
不允许全局作用域决定的运算符。

C5282 (E) The global scope has no " 名称 "
没有给全局作用域声明 “名称”。

C5283 (E) Qualified name is not allowed
不允许限定名。

C5284 (E) (W) NULL reference is not allowed
不允许参照 NULL。按指定的内容评价表达式。

C5285 (E) Initialization with "{...}" is not allowed for object of type " 型 "
在 “型” 的目标中，不允许 {} 格式的初始化。

C5286 (E) Base class " 型 " is ambiguous
基类的型不明确。

C5287 (E) Derived class " 型 " contains more than one instance of class " 型 "
派生型包含多个相同 “型” 的类。

C5288 (E) Cannot convert pointer to base class " 型 1" to pointer to derived class " 型 2" -- base class is virtual
不能将虚拟基类 “型 1” 的指针型转换为派生类 “型 2” 的指针型。

C5289 (E) No instance of constructor " 名称 " matches the argument list
构造函数 “名称” 的参数不同。

C5290 (E) Copy constructor for class " 型 " is ambiguous
类 “型” 的复制构造函数不明确。

C5291 (E) No default constructor exists for class "型"
不存在类 “型” 的默认构造函数。

C5292 (E) "名称" is not a nonstatic data member or base class of class "型"
“名称” 不是非静态数据成员或者基类 “型”。

C5293 (E) Indirect nonvirtual base class is not allowed
不允许非虚拟的间接基类。

C5294 (E) Invalid union member -- class "型" has a disallowed member function
存在不能给 union 成员指定的类 “型” 的成员函数。

C5296 (E) (W) Invalid use of non-lvalue array
非法使用了非左边值的数组。

C5297 (E) Expected an operator
需要运算符。

C5298 (E) Inherited member is not allowed
不能使用继承的成员。

C5299 (E) Cannot determine which instance of entity-kind "名称" is intended
不能决定重载函数的 “名称”。

C5300 (E) (W) A pointer to a bound function may only be used to call the function
将指向成员函数的指针用于函数调用以外的用途。

C5301 (E) Typedef name has already been declared (with same type)
已经以相同的型定义了 typdef 的名称。

C5302 (E) Entity-kind "名称" has already been defined
已定义了函数 “名称”。

C5304 (E) No instance of entity-kind "名称" matches the argument list
函数 “名称” 的参数不同。

C5305 (E) Type definition is not allowed in function return type declaration
不能在函数返回型的声明中定义型。

C5306 (E) Default argument not at end of parameter list
默认参数的声明不在参数列表的最后。

C5307 (E) Redefinition of default argument
重新定义了默认参数。

C5308 (E) More than one instance of "名称" matches the argument list:
因为参数列表相同，所以重载函数 “名称” 不明确。

C5309 (E) More than one instance of constructor "名称" matches the argument list:
因为参数列表相同，所以构造函数 “名称” 不明确。

C5310 (E) Default argument of type "型1" is incompatible with parameter of type "型2"
默认值的 “型1” 和参数的 “型2” 不同。

C5311 (E) Cannot overload functions distinguished by return type alone
不能重载不同返回型的函数。

C5312 (E) No suitable user-defined conversion from "型1" to "型2" exists
不存在从 “型1” 到 “型2” 的适当的用户定义转换。

C5313 (E) Type qualifier is not allowed on this function
不能给函数指定型限定词 （const、 volatile）。

C5314 (E) Only nonstatic member functions may be virtual
给静态成员函数指定了 virtual。

C5315 (E) The object has cv-qualifiers that are not compatible with the member function
目标的型限定词 （const、 volatile）和成员函数的型限定词不同。

C5316 (E) Program too large to compile (too many virtual functions)
虚拟函数的个数太多。

C5317 (E) Return type is not identical to nor covariant with return type "型" of overridden virtual function entity-kind "名称"
虚拟函数 “名称” 的返回型的 “型” 不同。

C5318 (E) Override of virtual entity-kind "名称" is ambiguous
虚拟函数 “名称” 的替换不明确。

C5319 (E) Pure specifier ("= 0") allowed only on virtual functions
在非虚拟函数中指定了纯说明符 “=0”。

C5320 (E) Badly-formed pure specifier (only "= 0" is allowed)
纯说明符的格式不正确。只允许 “=0”。

C5321 (E) Data member initializer is not allowed
数据成员的初始化指定不正确。

C5322 (E) Object of abstract class type "型" is not allowed:
不能定义抽象类 “型” 的目标。

C5323 (E) Function returning abstract class "型" is not allowed:
不能定义抽象类返回 “型” 的函数。

C5324 (I) Duplicate friend declaration
重复指定了友元声明。

C5325 (E) Inline specifier allowed on function declarations only
inline 说明符只在函数声明中有效。

C5326 (E) (W) "inline" is not allowed
不允许 inline 的指定。

C5327 (E) Invalid storage class for an inline function
这是非法的 inline 函数的存储类。

C5328 (E) Invalid storage class for a class member
这是非法的类成员的存储类。

C5329 (E) Local class member entity-kind "名称" requires a definition
没有定义局部类成员 “名称”。

C5330 (E) Entity-kind "名称" is inaccessible
不能存取 “名称”。

C5332 (E) Class "型" has no copy constructor to copy a const object
在类 “型” 中没有复制 const 型目标的复制构造函数。

C5333 (E) Defining an implicitly declared member function is not allowed
不能定义隐含声明的成员函数。

C5334 (E) Class "型" has no suitable copy constructor
不存在适用于类 “型” 的复制构造函数。

C5335 (E) (W) Linkage specification is not allowed
不能指定连接说明符。

C5336 (E) Unknown external linkage specification
指定了不能识别的连接指定。

C5337 (E) Linkage specification is incompatible with previous "名称" (declared at line "行号")
和前面指定的连接说明符 “名称” 不同。

C5338 (E) More than one instance of overloaded function "名称" has "C" linkage
存在多个有 C 连接的重载函数 “名称”。

C5339 (E) Class "型" has more than one default constructor
类 “型” 有多个默认构造函数。

C5340 (E) Value copied to temporary, reference to temporary used
值被复制到局部区域。使用了局部区域的参照。

C5341 (E) "operator 运算符" must be a member function
“运算符” 的运算符函数必须是成员函数。

C5342 (E) Operator may not be a static member function
不允许静态成员函数的运算符函数。

C5343 (E) No arguments allowed on user-defined conversion
对于用户定义转换，不允许参数。

C5344 (E) Too many parameters for this operator function
运算符函数的参数个数太多。

C5345 (E) Too few parameters for this operator function
运算符函数的参数个数不够。

C5346 (E) Nonmember operator requires a parameter with class type
非成员函数的运算符函数需要类（class）型的参数。

C5347 (E) Default argument is not allowed
不允许默认参数。

C5348 (E) More than one user-defined conversion from "型 1" to "型 2" applies:
从 “型 1” 到 “型 2” 的用户定义型转换不明确。

C5349 (E) No operator "运算符" matches these operands
运算符函数 “运算符” 的操作数不同。

C5350 (E) More than one operator "运算符" matches these operands:
运算符函数 “运算符” 的操作数不明确。

C5351 (E) First parameter of allocation function must be of type "size_t"
operator new 的第 1 参数必须是 size_t 型。

C5352 (E) Allocation function requires "void *" return type
operator new 的返回型必须是 void * 型。

C5353 (E) Deallocation function requires "void" return type
operator delete 的返回型必须是 void 型。

C5354 (E) First parameter of deallocation function must be of type "void *"
operator delete 的第 1 参数必须是 void * 型。

C5356 (E) Type must be an object type
型必须是目标型。

C5357 (E) Base class " 型 " has already been initialized
基类已被初始化。

C5359 (E) Entity-kind " 名称 " has already been initialized
“名称”已被初始化。

C5360 (E) Name of member or base class is missing
成员名或者基类有误。

C5363 (E) Invalid anonymous union -- nonpublic member is not allowed
无名 union 的成员不是公开成员。

C5364 (E) Invalid anonymous union -- member function is not allowed
在无名 union 中不允许成员函数。

C5365 (E) Anonymous union at global or namespace scope must be declared static
全局或者 namespace 作用域的无名 union 需要 static 声明。

C5366 (E) Entity-kind " 名称 " provides no initializer for:
不能给 “名称”指定初始化。

C5367 (E) Implicitly generated constructor for class " 型 " cannot initialize:
不能对隐含生成的类 “型”的构造函数进行初始化。

C5368 (W) Entity-kind " 名称 " defines no constructor to initialize the following:
“名称”没有定义初始化的构造函数。

C5369 (E) Entity-kind " 名称 " has an uninitialized const or reference member
“名称”的 const 或者参照成员没有被初始化。

C5370 (W) Entity-kind " 名称 " has an uninitialized const field
“名称”的 const 字段没有被初始化。

C5371 (E) Class " 型 " has no assignment operator to copy a const object
没有定义复制 const 目标的类 “型”的赋值运算符函数。

C5372 (E) Class "型" has no suitable assignment operator
没有给类 “型” 定义适当的赋值运算。

C5373 (E) Ambiguous assignment operator for class "型"
类 “型” 的赋值运算符函数不明确。

C5375 (E) Declaration requires a typedef name
需要 typedef 名的声明。

C5377 (W) "virtual" is not allowed
不能指定 virtual。

C5378 (E) "static" is not allowed
不能指定 static。

C5380 (E) Expression must have pointer-to-member type
表达式的型必须是指向成员的指针型。

C5381 (I) Extra ";" ignored
忽视多余的 “;”。

C5382 (W) In-class initializer for nonstatic member is nonstandard
非静态成员的初始化不是标准格式。

C5384 (E) No instance of overloaded "名称" matches the argument list
重载函数 “名称” 的参数列表不同。

C5386 (E) No instance of entity-kind "名称" matches the required type
不存在被要求的型的重载函数 “名称”。

C5388 (E) "operator->" for class "型1" returns invalid type "型2"
类 “型1” 的 operator-> 运算函数的返回型 “型2” 不正确。

C5389 (E) A cast to abstract class "型" is not allowed:
不允许向抽象类 “型” 的型转换。

C5390 (E) Function "main" may not be called or have its address taken
不能调用 main 函数或者取得地址。

C5391 (E) A new-initializer may not be specified for an array
不能通过 new 对数组进行初始化。

C5392 (E) Member function "名称" may not be redeclared outside its class
在类的外面重新声明了成员函数 “名称”。

C5393 (E) Pointer to incomplete class type is not allowed
不允许指向不完全类的指针型。

C5394 (E) Reference to local variable of enclosing function is not allowed
不允许参照包含局部类的函数的局部变量。

C5397 (E) Implicitly generated assignment operator cannot copy:
隐含生成的赋值运算符函数不能正确地复制目标。

C5398 (W) Cast to array type is nonstandard (treated as cast to "型")
数组型的型转换不是标准格式（假设向“型”的型转换）。

C5399 (I) Entity-kind "名称" has an operator newxxxx() but no default operator deletexxxx()
“名称”有 operator new，但是没有默认的 operator delete。

C5400 (I) Entity-kind "名称" has a default operator deletexxxx() but no operator newxxxx()
“名称”有默认的 operator delete，但是没有 operator new。

C5401 (E) Destructor for base class "型" is not virtual
基类“型”的析构函数不是 virtual。

C5403 (E) Invalid redeclaration of member "函数名"
这是成员函数非法的重新声明。

C5404 (E) Function "main" may not be declared inline
不能对 main 函数进行 inline 声明。

C5405 (E) Member function with the same name as its class must be a constructor
和类名同名的成员函数必须是构造函数。

C5407 (E) A destructor may not have parameters
析构函数不能有参数。

C5408 (E) Copy constructor for class "型1" may not have a parameter of type "型2"
类“型 1”的复制构造函数不能有“型 2”的参数。

C5409 (E) Entity-kind "名称" returns incomplete type "型"
函数“名称”的返回型不是不完全型“型”。

C5410 (E) Protected entity-kind "名称" is not accessible through a "型" pointer or object
不能在经过指向“型”的指针或者目标后存取限定公开名的“名称”。

C5411 (E) A parameter is not allowed
不允许形式参数。

C5412 (E) An "asm" declaration is not allowed here
不允许 asm 声明。

C5413 (E) No suitable conversion function from "型 1" to "型 2" exists
不存在从“型 1”到“型 2”的适当的转换函数。

C5414 (W) Delete of pointer to incomplete class
删除了指向不完全型类的指针。

C5415 (E) No suitable constructor exists to convert from "型 1" to "型 2"
不存在从“型 1”到“型 2”的适当的转换构造函数。

C5416 (E) More than one constructor applies to convert from "型 1" to "型 2":
从“型 1”转换到“型 2”的构造函数不明确。

C5417 (E) More than one conversion function from "型 1" to "型 2" applies:
从“型 1”到“型 2”的转换函数不明确。

C5418 (E) More than one conversion function from "型" to a built-in type applies:
从“型”到内部型的转换函数不明确。

C5424 (E) A constructor or destructor may not have its address taken
不能参照构造函数或者析构函数的地址。

C5427 (E) Qualified name is not allowed in member declaration
不能在成员声明中使用限定名。

C5429 (E) The size of an array in "new" must be non-negative
不允许 new 指定的数组大小是负值。

C5430 (W) Returning reference to local temporary
在函数内将局部区的参照设定到返回值。

C5432 (E) "enum" declaration is not allowed
不允许枚举型的声明。

C5433 (E) Qualifiers dropped in binding reference of type "型 1" to initializer of type "型 2"
给参照型“型 1”的初始值指定了 const/volatile 限定的型“型 2”。

C5434 (E) A reference of type "型 1" (not const-qualified) cannot be initialized with a value of type "型 2"
不能用 “型 2” 的值对非 const 型限定的型 “型 1” 的参照进行初始化。

C5435 (E) A pointer to function may not be deleted
不能删除指向函数的指针。

C5436 (E) Conversion function must be a nonstatic member function
转换函数必须是非静态成员函数。

C5437 (E) Template declaration is not allowed here
不允许在此作用域中进行模板声明。

C5438 (E) Expected a "<"
需要 “<”。

C5439 (E) Expected a ">"
需要 “>”。

C5440 (E) Template parameter declaration is missing
模板的参数声明不正确。

C5441 (E) Argument list for entity-kind "名称" is missing
模板 “名称” 的实际参数列表不正确。

C5442 (E) Too few arguments for entity-kind "名称"
模板 “名称” 的实际参数不够。

C5443 (E) Too many arguments for entity-kind "名称"
模板的实际参数太多。

C5445 (E) Entity-kind "名称 1" is not used in declaring the parameter types of entity-kind "名称 2"
不使用模板 “名称 2” 的参数 “名称 1”。

C5449 (E) More than one instance of entity-kind "名称" matches the required type
重载函数 “名称” 不明确。

C5450 (E) The type "long long" is nonstandard
long long 型不是标准格式。

C5451 (E) Omission of "class" is nonstandard
无 “class” 的 friend 声明不是标准格式。

C5452 (E) Return type may not be specified on a conversion function
没有指定转换函数的返回型。

C5456 (E) Excessive recursion at instantiation of entity-kind "名称"
递归生成模板 “名称” 的示例。

C5457 (E) "名称" is not a function or static data member
“名称” 不是函数或者静态数据成员。

C5458 (E) Argument of type "型1" is incompatible with template parameter of type "型2"
实际参数的型 “型1” 和模板的参数 “型2” 不同。

C5459 (E) Initialization requiring a temporary or conversion is not allowed
不允许在初始化时要求临时和转换。

C5460 (W) Declaration of "变量名" hides function parameter
函数内的变量声明隐藏了函数参数。

C5461 (E) Initial value of reference to non-const must be an lvalue
非 const 型参照的初始值必须是左边值。

C5463 (E) "template" is not allowed
不允许 “template” 的指定。

C5464 (E) "型" is not a class template
“型” 不是类模板。

C5466 (E) "main" is not a valid name for a function template
“main” 不能用于函数模板的名称。

C5467 (E) Invalid reference to entity-kind "名称" (union/nonunion mismatch)
这是非法的 “名称” 的参照。

C5468 (E) A template argument may not reference a local type
模板的实际参数不能参照局部型。

C5469 (E) Tag kind of "名称1" is incompatible with declaration of entity-kind "名称2" (declared at line "行号")
标记符名 “名称1” 的种类和 “名称2” 的声明不同。

C5470 (E) The global scope has no tag named "名称"
全局作用域中没有标记符名 “名称”。

C5471 (E) Entity-kind "名称 1" has no tag member named "名称 2"
“名称 1”没有标记符成员 “名称 2”。

C5473 (E) Entity-kind "名称" may be used only in pointer-to-member declaration
在指向成员的指针型的声明中，必须使用 typedef 名 “名称”。

C5475 (E) A template argument may not reference a non-external entity
模板的实际参数不能参照非外部名。

C5476 (E) Name followed by "::~" must be a class name or a type name
“::~”之后的名称必须是类名或者型名。

C5477 (E) Destructor name does not match name of class "型"
类名 “型”和析构函数名不同。

C5478 (E) Type used as destructor name does not match type "型"
用于析构函数名的型和 “型”不同。

C5479 (I) Entity-kind "名称" redeclared "inline" after being called
在调用函数后声明了 inline “名称”。将以后的 inline 指定置为有效。

C5481 (E) Invalid storage class for a template declaration
模板声明的存储类指定不正确。

C5484 (E) Invalid explicit instantiation declaration
这是非法的模板的实际参数。

C5485 (E) Entity-kind "名称" is not an entity that can be instantiated
不能对模板 “名称”进行实体化。

C5486 (E) Compiler generated entity-kind "名称" cannot be explicitly instantiated
不能对编译程序生成的函数进行实体化。

C5487 (E) (I) Inline entity-kind "名称" cannot be explicitly instantiated
不能对 inline 函数 “名称”进行实体化。

C5489 (E) Entity-kind "名称" cannot be instantiated -- no template definition was supplied
因为没有定义模板，所以不能对 “名称”进行实体化。

C5490 (E) Entity-kind "名称" cannot be instantiated -- it has been explicitly specialized
不能对 “名称”进行实体化。

C5493 (E) No instance of entity-kind " 名称 " matches the specified type
重载函数 “名称” 和指定的型不同。

C5494 (E) (W) Declaring a void parameter list with a typedef is nonstandard
被 typedef 定义的 void 参数列表的声明不是标准格式。

C5496 (E) Template parameter " 名称 " may not be redeclared in this scope
在作用域中重新声明了模板参数 “名称”。

C5497 (W) Declaration of " 名称 " hides template parameter
“名称” 的声明隐藏了模板参数。

C5498 (E) Template argument list must match the parameter list
模板的实际参数和形式参数不同。

C5500 (E) Extra parameter of postfix "operatorxxxx" must be of type "int"
后缀运算函数的第 2 参数的型必须是 int 型。

C5501 (E) An operator name must be declared as a function
必须将运算符名声明为函数。

C5502 (E) Operator name is not allowed
不允许运算符名。

C5503 (E) Entity-kind " 名称 " cannot be specialized in the current scope
在作用域中 “名称” 不明确。

C5504 (E) Nonstandard form for taking the address of a member function
成员函数地址的取得不是标准格式。

C5505 (E) Too few template parameters -- does not match previous declaration
模板的参数不够。

C5506 (E) Too many template parameters -- does not match previous declaration
模板的参数太多。

C5507 (E) Function template for operator delete(void *) is not allowed
不允许 operator delete(void *) 的函数模板。

C5508 (E) Class template and template parameter may not have the same name
类模板和模板的参数同名。

C5510 (E) A template argument may not reference an unnamed type
参照了没有命名的模板实际参数的型。

C5511 (E) Enumerated type is not allowed
不允许枚举型。

C5512 (W) Type qualifier on a reference type is not allowed
不能给参照型指定 const/volatile 限定。

C5513 (E) (W) A value of type "型 1" cannot be assigned to an entity of type "型 2"
因为型不同，所以 “型 1” 的值不能赋值到 “型 2” 的实体。
(W) 型 1 和型 2 是指向没有兼容性的型的指针。
(E) 型 1 和型 2 是没有兼容性的型。

C5514 (W) Pointless comparison of unsigned integer with a negative constant
将负常数和无符号的整数进行了比较。

C5515 (E) Cannot convert to incomplete class "型"
不能向不完全型 “型” 进行型转换。

C5516 (E) Const object requires an initializer
const 型的目标需要初始值。

C5517 (E) Object has an uninitialized const or reference member
目标有未初始化的 const 型成员或者参照型成员。

C5518 (E) Nonstandard preprocessing directive
有非标准格式的预处理器的关键字。

C5519 (E) Entity-kind "名称" may not have a template argument list
“名称” 不能有模板的实际参数。

C5520 (E) (W) Initialization with "{...}" expected for aggregate object
必须用 {...} 的格式对集合型的目标进行初始化。

C5521 (E) Pointer-to-member selection class types are incompatible ("型1" and "型 2")
成员指针型的类的型 “型 1” 和 “型 2” 不同。

C5522 (W) Pointless friend declaration
对自身进行了友元声明。

C5523 (W) "." used in place of "::" to form a qualified name
使用了 “.” 代替 “::” 的作用域限定名。

C5525 (W) A dependent statement may not be a declaration
条件表达式没有作用域。

C5526 (E) A parameter may not have void type
不能指定 void 型的参数。

C5529 (E) This operator is not allowed in a template argument expression
不允许模板的实际参数表达式指定的运算。

C5530 (E) Try block requires at least one handler
没有对应 try 语句的 catch 语句。

C5531 (E) Handler requires an exception declaration
Catch 语句的 (...) 需要异常声明。

C5532 (E) Handler is masked by default handler
通过默认处理程序屏蔽了处理程序。

C5533 (W) Handler is potentially masked by previous handler for type "型"
有可能根据前面的 “型” 的处理程序屏蔽处理程序。

C5534 (I) Use of a local type to specify an exception
指定了使用局部型的异常处理。

C5535 (I) Redundant type in exception specification
在异常处理中指定了冗余型。

C5536 (E) Exception specification is incompatible with that of previous
entity-kind "名称" (declared at line "行号"):
异常处理的指定和前面的指定 “名称” 不同。

C5540 (E) Support for exception handling is disabled
没有指定异常处理的选项 （exception）。

C5541 (W) Omission of exception specification is incompatible with previous
entity-kind "名称" (declared at line "行号")
异常处理的省略形式和前面的 “名称” 不同。

C5542 (F) Could not create instantiation request file "名称"
不能建立用于对模板进行实体化的文件 “名称”。

C5543 (E) Non-arithmetic operation not allowed in nontype template argument
在对应的模板实际参数中不允许非算术型转换。

C5544 (E) Use of a local type to declare a nonlocal variable
给非局部变量指定了局部型。

C5545 (E) Use of a local type to declare a function
给函数声明指定了局部型。

C5546 (E) Transfer of control bypasses initialization of:
不进行初始化处理。

C5548 (E) Transfer of control into an exception handler
执行异常处理程序。

C5549 (I) Entity-kind "名称" is used before its value is set
在给 “名称” 设定值前使用了 “名称”。

C5550 (W) Entity-kind "名称" was set but never used
没有使用 “名称”。

C5551 (E) Entity-kind "名称" cannot be defined in the current scope
不能在此作用域中定义 “名称”。

C5552 (W) Exception specification is not allowed
不允许异常处理的指定。将异常处理置为无效。

C5553 (W) External/internal linkage conflict for entity-kind "名称" (declared at line "行号")
“名称” 的外部 / 内部连接的指定发生冲突。设定外部连接。

C5554 (W) Entity-kind "名称" will not be called for implicit or explicit conversions
不能既隐含又明确地调用转换函数 “名称”。

C5555 (E) Tag kind of "名称" is incompatible with template parameter of type "型"
标记符 “名称” 的种类和模板的参数的 “型” 不同。

C5556 (E) Function template for operator new(size_t) is not allowed
不允许 operator new(size_t) 的函数模板。

C5558 (E) Pointer to member of type "型" is not allowed
指向成员的指针型 “型” 有误。

C5559 (E) Ellipsis is not allowed in operator function parameter list
不能给运算符函数的参数列表指定省略符号 （...）。

C5560 (E) "关键字" is reserved for future use as a keyword
关键字是将来安装的保留字。

C5563 (F) Invalid preprocessor output file
这是不能用于预处理器输出的文件名。

C5598 (E) A template parameter may not have void type
不能给模板的参数指定 void 型。

C5599 (E) Excessive recursive instantiation of entity-kind "名称" due to instantiate-all mode
通过指定 instantiate-all 模式，递归生成模板 “名称” 的示例。

C5601 (E) A throw expression may not have void type
不能给 throw 表达式指定 void 型。

C5603 (E) Parameter of abstract class type "型" is not allowed:
不允许抽象类 “型” 的参数。

C5604 (E) Array of abstract class "型" is not allowed:
不允许抽象类 “型” 的数组。

C5605 (E) Floating-point template parameter is nonstandard
浮点的模板参数不是标准格式。

C5606 (E) This pragma must immediately precede a declaration
必须在声明前记述此 pragma。

C5607 (E) This pragma must immediately precede a statement
必须在表达式之前记述此 pragma。

C5608 (E) This pragma must immediately precede a declaration or statement
必须在声明或者表达式之前记述此 pragma。

C5609 (E) This kind of pragma may not be used here
不能在此使用此类 pragma。

C5611 (W) Overloaded virtual function "名称 1" is only partially overridden in entity-kind "名称 2"
“名称 1” 的重载虚拟函数在 “名称 2” 中只有部分虚拟函数为替换对象。按指定的内容继续处理。

C5612 (E) Specific definition of inline template function must precede its first use
必须在调用前定义 inline 指定的模板函数

C5615 (E) Parameter type involves pointer to array of unknown bound
参数的型包括指向没有指向元素个数的数组的指针。

C5616 (E) Parameter type involves reference to array of unknown bound
参数的型包括对没有指定元素个数的数组的参照。

C5617 (W) Pointer-to-member-function cast to pointer to function
进行了成员函数指针到函数指针型的型转换。

C5618 (I) Struct or union declares no named members
在结构体或者联合体中不包括命名成员。

C5619 (E) Nonstandard unnamed field
这是非标准格式的未命名字段。

C5620 (E) Nonstandard unnamed member
这是非标准格式的未命名成员。

C5624 (E) " 名称 " is not a type name
“名称”不是型的名称。

C5641 (F) " 名称 " is not a valid directory
“名称”不是正确的文件夹。

C5642 (F) Cannot build temporary file name
不能建立编译程序使用的临时文件。

C5643 (E) "vestrict" is not allowed
不能指定 “restrict”。

C5644 (E) A pointer or reference to function type may not be qualified by "restrict"
不能通过 “restrict” 限定指向函数的指针或者参照型。

C5647 (E) Conflicting calling convention modifiers
调用规约限定词发生冲突。

C5650 (W) Calling convention specified here is ignored
忽视在此指定的调用规约。

C5651 (E) A calling convention may not be followed by a nested declarator
嵌套的说明符不能接在调用规约之后。

C5652 (I) Calling convention is ignored for this type
忽视对此型的调用规约。

C5654 (E) Declaration modifiers are incompatible with previous declaration
说明符和前面声明的说明符没有兼容性。

C5656 (E) Transfer of control into a try block
控制从外侧的块移到 try 块。

C5657 (W) Inline specification is incompatible with previous "名称" (declared at line "行号")
Inline 的指定和前面的声明 “名称” 不同。

C5658 (E) Closing brace of template definition not found
模板定义没有右括弧。

C5660 (E) Invalid packing alignment value
这是非法的 Pack 值。

C5661 (E) Expected an integer constant
没有整数常数。

C5662 (W) Call of pure virtual function
纯虚拟函数调用了函数。

C5663 (E) Invalid source file identifier string
#pragma 指定的语法有误。

C5664 (E) A class template cannot be defined in a friend declaration
不能在友元声明中定义类模板。

C5665 (E) "asm" is not allowed
不能使用 asm 说明符。

C5666 (E) "asm" must be used with a function definition
必须在定义函数的同时指定 asm 说明符。

C5667 (E) "asm" function is nonstandard
asm 函数不是标准格式。

C5668 (E) Ellipsis with no explicit parameters is nonstandard
只有指定省略符号 （...）的参数不是标准格式。

C5669 (E) "&..." is nonstandard
“&...” 的参数不是标准格式。

C5670 (E) Invalid use of "&..."
“&...” 的使用有误。

C5673 (E) A reference of type "型 1" cannot be initialized with a value of type "型 2"
不能用 “型 2” 的值对 const/volatile 型 “型 1” 的参照进行初始化。

C5674 (E) Initial value of reference to const volatile must be an lvalue
const/volatile 型参照的初始值必须是左边值。

C5676 (W) Using out-of-scope declaration of "符号名"
Using 声明在符号的作用域外。

C5678 (I) Call of entity-kind "名称" (declared at line "行号") cannot be inlined
不能对函数调用 “名称” 进行 inline 展开。

C5679 (I) Entity-kind "名称" cannot be inlined
不能对函数 “名称” 进行 inline 展开。

C5691 (E) (W)"符号", required for copy that was eliminated, is inaccessible
不能存取复制构造函数。

C5692 (E) (W) "符号", required for copy that was eliminated, is not callable because reference parameter cannot be bound to rvalue
不能调用复制构造函数。

C5693 (E) <typeinfo> must be included before typeid is used
为了使用 typeid，必须包含 <typeinfo>。

C5694 (E) "名称" cannot cast away const or other type qualifiers
失去了 “名称” 的数据型转换的结果 const 等的属性。

C5695 (E) The type in a dynamic_cast must be a pointer or reference to a complete class type, or void *
dynamic_cast 的型必须是指向完全类 （class）型的指针型、参照型或者 void * 型。

C5696 (E) The operand of a pointer dynamic_cast must be a pointer to a complete class type
dynamic_cast 指针的操作数必须是指向完全类 （class）型的指针型。

C5697 (E) The operand of a reference dynamic_cast must be an lvalue of a complete class type
dynamic_cast 参照的操作数必须是完全类 （class）型的左边值。

C5698 (E) The operand of a runtime dynamic_cast must have a polymorphic class type
执行时 dynamic_cast 的操作数必须是多态的类 （class）型。

C5701 (E) An array type is not allowed here
不允许数组型。

C5702 (E) Expected an "="
需要赋值表达式。

C5703 (E) Expected a declarator in condition declaration
需要说明符。

C5704 (E) "名称", declared in condition, may not be redeclared in this scope
不能在此作用域中重新声明 “名称”。

C5705 (E) Default template arguments are not allowed for function templates
不能给函数模板指定默认的实际参数。

C5706 (E) Expected a "," or ">"
需要 “,” 或者 “>”。

C5707 (E) Expected a template parameter list
需要模板的参数列表。

C5708 (W) Incrementing a bool value is deprecated
bool 型的值递增。值递增后继续处理。

C5709 (E) bool type is not allowed
bool 型的值不能递减。

C5710 (E) Offset of base class "名称1" within class "名称2" is too large
类 “名称 2” 内的基类 “名称 1” 太大。

C5711 (E) Expression must have bool type (or be convertible to bool)
表达式的型必须是 bool 型或者是能转换为 bool 型的型。

C5717 (E) The type in a const_cast must be a pointer, reference, or pointer to member to an object type
const_cast 的型必须是指针型、参照型或者指向成员的指针型。

C5718 (E) A const_cast can only adjust type qualifiers; it cannot change the underlying type
const_cast 不能调整 const/volatile 以外的型。

C5719 (E) mutable is not allowed
不允许 mutable 的指定。

C5720 (W) Redeclaration of entity-kind "名称" is not allowed to alter its access
不能通过重新声明 “名称” 更改存取的指定。前面声明的存取指定有效。

C5722 (W) Use of alternative token "<:" appears to be unintended
使用了 2 字符的记述 “<:”，解释为 “[”。

C5723 (W) Use of alternative token "%:" appears to be unintended
使用了 2 字符的记述 “%:”，解释为 “#”。

C5724 (E) namespace definition is not allowed
允许在文件作用域或者 namespace 作用域中定义 namespace。

C5725 (E) Name must be a namespace name
Namespace 的名称不正确。

C5726 (E) Namespace alias definition is not allowed
在此不允许 namespace 的别名定义。

C5727 (E) namespace-qualified name is required
需要 namespace 的限定名。

C5728 (E) A namespace name is not allowed
不允许 namespace 名。

C5730 (E) Entity-kind "名称" is not a class template
“名称”不是类模板的成员。

C5731 (E) Array with incomplete element type is nonstandard
有不完全元素型的数组不是标准格式。

C5732 (E) Allocation operator may not be declared in a namespace
在 namespace 内声明了 operator new 函数。

C5733 (E) Deallocation operator may not be declared in a namespace
在 namespace 内声明了 operator delete 函数。

C5734 (E) Entity-kind "名称1" conflicts with using-declaration of entity-kind "名称2"
名称 “名称 1” 和 using 声明名 “名称 2” 发生冲突。

C5735 (E) Using-declaration of entity-kind "名称1" conflicts with entity-kind "名称2" (declared at line "行号")
using 声明的名称发生冲突。

C5737 (W) Using-declaration ignored -- it refers to the current namespace
对当前 namespace 作用域的名称进行了 using 声明。忽视 using 声明。

C5738 (E) A class-qualified name is required
需要类的限定名。

C5741 (W) Using-declaration of entity-kind "名称" ignored
using 声明 “名称”无效。

C5742 (E) Entity-kind "名称1" has no actual member "名称2"
在 “名称 1” 中不存在 “名称 2” 的成员。

C5748 (W) Calling convention specified more than once
至少指定了 1 次调用规约。

C5749 (E) A type qualifier is not allowed
不能指定型限定词。

C5750 (E) Entity-kind " 名称 " (declared at line " 行号 ") was used before its template was declared
在声明模板前使用了 “名称”。

C5751 (E) Static and nonstatic member functions with same parameter types cannot be overloaded
不能重载有相同参数的型的静态成员函数和非静态成员函数。

C5752 (E) No prior declaration of entity-kind " 名称 "
没有声明 namespace 模板函数 “名称”。

C5753 (E) A template-id is not allowed
在此不允许模板 （template 名 <template 实际参数 >）。

C5754 (E) A class-qualified name is not allowed
在此不允许类限定名。

C5755 (E) Entity-kind " 名称 " may not be redeclared in the current scope
不能在此作用域中重新声明 “名称”。

C5756 (E) Qualified name is not allowed in namespace member declaration
不允许在 namespace 成员声明中指定的限定名。

C5757 (E) Entity-kind " 名称 " is not a type name
“名称” 不是型名。

C5758 (E) Explicit instantiation is not allowed in the current scope
不能在当前的作用域范围内明确生成示例。

C5759 (E) " 符号名 " cannot be explicitly instantiated in the current scope
不能在当前的作用域中对符号进行明确的示例化。

C5760 (W) " 符号 " explicitly instantiated more than once
不能对符号进行具体化。

C5761 (E) Typename may only be used within a template
只能在模板内使用 typename 关键字。

C5765 (E) Nonstandard character at start of object-like macro definition
在目标宏定义的开头含有非标准的字符串。

C5766 (W) Exception specification for virtual entity-kind " 名称 1" is incompatible with that of overridden entity-kind " 名称 2"

虚拟函数的异常指定 “名称 1” 和 “名称 2” 不同。

C5767 (W) Conversion from pointer to smaller integer

将指针转换为小于指针大小的型。

C5768 (W) Exception specification for implicitly declared virtual entity-kind " 名称 1" is incompatible with that of overridden entity-kind " 名称 2"

编译程序生成的隐含虚拟函数 “名称 1” 的异常指定和 “名称 2” 不同。

C5769 (E) " 符号 1", implicitly called from " 符号 2", is ambiguous

operator delete 的调用不明确。

C5771 (E) "explicit" is not allowed

只能给类声明内的构造函数指定 explicit。

C5772 (E) Declaration conflicts with " 名称 " (reserved class name)

和保留的类名 type_info 发生冲突。

C5773 (E) Only "()" is allowed as initializer for array entity-kind " 名称 "

数组 “名称” 的初始化指定不正确。

C5774 (E) "virtual" is not allowed in a function template declaration

不能在函数模板中指定 virtual。

C5775 (E) Invalid anonymous union -- class member template is not allowed

无名 union 的指定不正确。

C5776 (E) Template nesting depth does not match the previous declaration of entity-kind " 名称 "

模板参数的嵌套和前面的声明 “名称” 不同。

C5777 (E) This declaration cannot have multiple "template <...>" clauses

此声明不能声明多个模板。

C5779 (E) " 名称 ", declared in for-loop initialization, may not be redeclared in this scope

不能在此作用域中重新声明由 for 语句的初始化表达式声明的 “名称”。

C5780 (W) Reference is to "符号1" -- under old for-init scoping rules it would have been " 符号 2"

参照了 “符号 1”。

C5782 (E) Definition of virtual entity-kind " 名称 " is required here

需要虚拟函数的定义 “名称”。

C5783 (W) Empty comment interpreted as token-pasting operator "##"
空的注释假设为字句连接运算符 “##”。

C5784 (E) A storage class is not allowed in a friend declaration
不能在友元声明中指定存储类。

C5785 (E) Template parameter list for "名称" is not allowed in this declaration
在此声明中不允许 “名称”的模板的参数排列。

C5786 (E) entity-kind "名称" is not a valid member class or function template
“名称”不是有效的成员或者函数模板。

C5787 (E) Not a valid member class or function template declaration
不是有效的成员或者函数模板声明。

C5788 (E) A template declaration containing a template parameter list may not be followed by an explicit specialization declaration
不能在定义模板函数后指定包含模板参数排列的模板声明。

C5789 (E) Explicit specialization of entity-kind "名称1" must precede the first use of entity-kind "名称2"
显式模板实体的定义 “名称1”必须在使用最初的模板 “名称2”之前。

C5790 (E) Explicit specialization is not allowed in the current scope
不允许在此作用域中定义显式模板实体。

C5791 (E) Partial specialization of entity-kind "名称" is not allowed
不允许模板 “名称”的部分定义。

C5792 (E) Entity-kind "名称" is not an entity that can be explicitly specialized
“名称”不是模板的示例。

C5793 (E) Explicit specialization of entity-kind "名称" must precede its first use
必须在最初使用前定义显式模板实体 “名称”。

C5794 (W) Template parameter "模板参数" may not be used in an elaborated type specifier
不能对 class 指定使用模板参数。在将 Class 的指定置为无效后将模板置为有效。

C5795 (E) Specializing "名称" requires "template<>" syntax
“名称”模板实体的定义需要 tempalte<> 格式。

C5799 (E) Specializing "符号名" without "template<>" syntax is nonstandard
没有 “template<>”的符号特殊化语法不是标准格式。

C5800 (E) This declaration may not have extern "C" linkage
此声明不能有 extern "C" 的连接。

C5801 (E) " 名称 " is not a class or function template name in the current scope
在此作用域中 “名称” 不是类模板或者函数模板。

C5802 (W) Specifying a default argument when redeclaring an unreferenced function template is nonstandard
在重新声明未参照的函数模板时指定了默认参数。忽视默认参数。

C5803 (E) Specifying a default argument when redeclaring an already referenced function template is not allowed
在重新声明已被参照的函数模板时指定了默认参数。

C5804 (E) Cannot convert pointer to member of base class " 型 1" to pointer to member of derived class " 型 2" -- base class is virtual
不能将虚拟基类 “型 1” 的成员指针转换为派生类 “型 2” 的成员指针。

C5805 (E) Exception specification is incompatible with that of entity-kind " 名称 " (declared at line " 行号 "):
Throw 的异常指定和 “名称” 的异常指定不同。

C5806 (W) Omission of exception specification is incompatible with entity-kind " 名称 " (declared at line " 行号 ")
Throw 异常指定的省略和 “名称” 的异常指定不同。将 “名称” 置为有效。

C5807 (E) Unexpected end of default argument expression
默认参数表达式不正确。

C5808 (E) Default-initialization of reference is not allowed
不允许参照型的默认初始化。

C5809 (E) Uninitialized entity-kind " 名称 " has a const member
未初始化的 “名称” 有 const 型成员。

C5810 (E) Uninitialized base class " 型 " has a const member
未初始化的基类 “型” 有 const 型成员。

C5811 (E) Const entity-kind " 名称 " requires an initializer -- class " 型 " has no explicitly declared default constructor
const 型的 “名称” 需要初始化指定。类 “型” 没有明确声明的默认构造函数。

C5812 (E) (W) Const object requires an initializer -- class " 型 " has no explicitly declared default constructor
const 型目标需要初始化指定。类 “型” 没有明确声明的默认构造函数。

C5815 (I) Type qualifier on return type is meaningless
给模板实体化的返回型指定了无意义的限定型。将限定型置为有效。

C5816 (E) In a function definition a type qualifier on a "void" return type is not allowed
不能在函数定义中给 “void” 型的返回值指定型限定词。

C5817 (E) Static data member declaration is not allowed in this class
局部类不能有静态数据成员。

C5818 (E) Template instantiation resulted in an invalid function declaration
被模板实体化的函数声明不正确。

C5819 (E) "..." is not allowed
不能使用疑 “...”。

C5822 (E) Invalid destructor name for type "型"
“型” 的析构函数名不正确。

C5824 (E) Destructor reference is ambiguous -- both entity-kind "名称 1" and entity-kind "名称 2" could be used
使用了 “名称 1” 和 “名称 2”。析构函数的参照不明确。

C5825 (W) Virtual inline entity-kind "名称" was never defined
没有定义虚拟 inline 成员函数 “名称”。

C5826 (W) Entity-kind "名称" was never referenced
不参照函数的参数 “名称”。

C5827 (E) Only one member of a union may be specified in a constructor initializer list
在构造函数的初始化中，只能指定联合体的一个成员。

C5828 (E) Support for "new[]" and "delete[]" is disabled
不支持 “new[]” 和 “delete[]”。

C5829 (W) "double" used for "long double" in generated C code
在生成 C 码时将 “long double” 转换为 “double”。

C5830 (W) "符号"has no corresponding operator deletes (to be called if an exception is thrown during initialization of an allocated object)
没有对应的 operator delete。

C5831 (W) (I) Support for placement delete is disabled
operator delete 函数的型不正确。继续处理。

C5832 (E) No appropriate operator delete is visible
找不到合适的 operator delete 函数。

C5833 (E) Pointer or reference to incomplete type is not allowed
不允许指向不完全型的指针或者参照型。

C5834 (E) Invalid partial specialization -- entity-kind " 名称 " is already fully specialized
对被特殊化的 “名称” 进行了部分特殊化。

C5835 (E) Incompatible exception specifications
异常指定的型不同。

C5836 (W) Returning reference to local variable
将局部变量的参照指定为返回值。继续指定的处理。

C5837 (W) Omission of explicit type is nonstandard ("int" assumed)
没有指定型。假设 int 型。

C5838 (E) More than one partial specialization matches the template argument list of entity-kind " 名称 "
部分特殊模板 “名称” 的模板实际参数不明确。

C5840 (E) A template argument list is not allowed in a declaration of a primary template
不能在主模板声明中指定模板实际参数。

C5841 (E) Partial specializations may not have default template arguments
部分特殊化的模板不能有默认的模板参数。

C5842 (E) Entity-kind " 名称 1" is not used in template argument list of entity-kind " 名称 2"
部分特殊化模板 “名称 1” 不能用于 “名称 2” 的模板实际参数。

C5843 (E) The type of partial specialization template parameter entity-kind " 名称 " depends on another template parameter
部分特殊化模板 “名称” 的模板形式参数取决于其他模板的形式参数。

C5844 (E) The template argument list of the partial specialization includes a nontype argument whose type depends on a template parameter
部分特殊化模板的模板实际参数包含了取决于模板形式参数的非型实际参数。

C5845 (E) This partial specialization would have been used to instantiate entity-kind " 名称 "
此部分特殊化模板正要对主模板 “名称” 进行实体化。

C5846 (E) This partial specialization would have been made the instantiation of entity-kind "名称" ambiguous

此部分特殊化模板为不明确的 “名称” 的实体化。

C5847 (E) Expression must have integral or enum type

表达式的型必须是整数型或者枚举型。

C5848 (E) Expression must have arithmetic or enum type

表达式的型必须是算术型或者枚举型。

C5849 (E) Expression must have arithmetic, enum, or pointer type

表达式的型必须是算术型、枚举型或者指针型。

C5850 (E) Type of cast must be integral or enum

型转换的型必须是整数型或者枚举型。

C5851 (E) Type of cast must be arithmetic, enum, or pointer

型转换的型必须是算术型、枚举型或者指针型。

C5852 (E) Expression must be a pointer to a complete object type

表达式的型必须是指向完全目标型的指针型。

C5854 (E) A partial specialization nontype argument must be the name of a nontype parameter or a constant

部分特殊化模板的非型模板实际参数必须是非型的形式参数名或者常数。

C5855 (E) (W) Return type is not identical to return type "型" of overridden virtual function entity-kind "名称"

函数的返回型和被覆盖的虚拟函数 “名称” 的返回型 “型” 不同。

C5857 (E) A partial specialization of a class template must be declared in the namespace of which it is a member

必须在包含该成员的 namespace 中声明部分特殊化的模板。

C5858 (E) Entity-kind "名称" is a pure virtual function

“名称” 是纯虚拟函数。

C5859 (E) Pure virtual entity-kind "名称" has no overrider

不覆盖纯虚拟函数 “名称”。

C5861 (E) Invalid character in input line

行中出现了非法字符。

C5862 (E) Function returns incomplete type "型"

函数的返回型 “型” 是不完全型。

C5863 (I) Effect of this "#pragma pack" directive is local to "符号"
#pragma pack 指示的影响只局限于符号。

C5864 (E) "名称" is not a template
“名称”不是模板。

C5865 (E) A friend declaration may not declare a partial specialization
不能在友元声明中指定部分特殊化模板。

C5866 (I) Exception specification ignored
忽视异常指定。

C5867 (W) Declaration of "size_t" does not match the expected type "型"
size_t 型和期待的 “型”不同。

C5868 (E) Space required between adjacent ">" delimiters of nested template argument lists (">>" is the right shift operator)
在 2 个模板实际参数列表的最后指定的 “>>”之间需要间隔。

C5869 (E) Could not set locale to allow processing of multibyte characters
不能给多字节字符设定区域。

C5870 (W) Invalid multibyte character sequence
有非法的 2 字节字符。

C5871 (E) Template instantiation resulted in unexpected function type of "型 1" (the meaning of a name may have changed since the template declaration -- the type of the template is "型 2")
有 “型 2”的模板实体化的结果，建立了没有期待的型 “型 1”的函数。

C5872 (E) Ambiguous guiding declaration -- more than one function template no matches type "型"
模板函数不明确。

C5873 (E) Non-integral operation not allowed in nontype template argument
在非型的模板实际参数中，不允许非整数型运算。

C5875 (E) Embedded C++ does not support templates
Embedded C++ 规格不支持模板功能。

C5876 (E) Embedded C++ does not support exception handling
Embedded C++ 规格不支持异常处理功能。

C5877 (E) Embedded C++ does not support namespaces
Embedded C++ 规格不支持 namespace 功能。

C5878 (E) Embedded C++ does not support run-time type information
Embedded C++ 规格不支持运行时型信息功能。

C5879 (E) Embedded C++ does not support the new cast syntax
Embedded C++ 规格不支持新格式的型转换功能。

C5880 (E) Embedded C++ does not support using-declarations
Embedded C++ 规格不支持 using 声明功能。

C5881 (E) Embedded C++ does not support "mutable"
Embedded C++ 规格不支持 mutable 功能。

C5882 (E) Embedded C++ does not support multiple or virtual inheritance
Embedded C++ 规格不支持多重继承 / 虚拟继承功能。

C5885 (E) "型 1" cannot be used to designate constructor for "型 2"
“型 1”不能用于构造函数的 “型 2”。

C5886 (E) Invalid suffix on integral constant
这是非法的整数常数的后缀。

C5890 (E) Variable length array with unspecified bound is not allowed
没有指定可变长数组的大小。

C5891 (E) An explicit template argument list is not allowed on this declaration
在此声明中，不允许显式模板实际参数。

C5892 (E) An entity with linkage cannot have a type involving a variable length array
有连接说明符的声明不能有包含可变长数组的型。

C5893 (E) A variable length array cannot have static storage duration
可变长数组不能有静态存储期间。

C5894 (E) Entity-kind "名称" is not a template
“名称”不是模板。

C5896 (E) Expected a template argument
期待模板的实际参数。

C5898 (E) Nonmember operator requires a parameter with class or enum type
非成员运算符函数需要类或者枚举型的形式参数。

C5900 (E) Using-declaration of entity-kind "名称" is not allowed
不允许 “名称”的 using 声明。

C5901 (E) Qualifier of destructor name "型 1" does not match type "型 2"
“型 1”的析构函数的限定名和 “型 2”不同。

C5902 (W) Type qualifier ignored
这是非法的型限定名。将型限定名置为无效。

C5907 (E) Option "nonstd_qualifier_deduction" can be used only when compiling C++
只能在 C++ 编译时使用 “nonstd_qualifier_deduction”选项。

C5912 (W) Ambiguous class member reference – "符号 1" used in preference to "符号 2"
这是不明确的类成员的参照。在参照符号 1 后参照符号 2。

C5915 (E) A segment name has already been specified
这是已指定的段名。

C5916 (E) Cannot convert pointer to member of derived class "型 1" to pointer to member of base class "型 2" -- base class is virtual
不能将指向派生类 “型 1”成员的指针型转换为指向虚拟基类 “型 2”成员的指针型。

C5919 (F) Invalid output file: "名称"
这是非法的模板信息文件的 “名称”。请确认编译程序的环境设定和主机环境的文件系统是否异常。

C5920 (F) Cannot open output file: "名称"
不能打开模板信息文件 “名称”。请确认编译程序的环境设定和主机环境的文件系统是否异常。

C5925 (W) Type qualifiers on function types are ignored
忽视函数型的型限定词。

C5926 (F) Cannot open definition list file: "名称"
不能打开 “名称”文件。请确认编译程序的环境设定和主机环境的文件系统是否异常。

C5928 (E) Incorrect use of va_start
va_start 的使用方法有误。

C5929 (E) Incorrect use of va_arg
va_arg 的使用方法有误。

C5930 (E) Incorrect use of va_end
va_end 的使用方法有误。

C5934 (E) A member with reference type is not allowed in a union
参照型不能是联合体成员。

C5935 (E) "typedef" may not be specified here
不能指定 typedef。

C5936 (W) Redeclaration of entity-kind "名称" alters its access
通过 “名称 “的重新声明中更改了存取的指定。将重新定义的存取指定置为有效。

C5937 (E) A class or namespace qualified name is required
需要类或者 namespace 的限定名。

C5938 (E) Return type "int" omitted in declaration of function "main"
在 main 函数的声明中 int 型的返回值被除外。

C5939 (E) pointer-to-member representation "符号 1" is too restrictive for "符号 2"
指向成员的指针的声明不正确。

C5940 (W) Missing return statement at end of non-void entity-kind "名称"
非 void 型返回的函数 “名称 “没有 return 语句。 return 值为不定值。

C5941 (W) Duplicate using-declaration of "名称" ignored
重复指定了 using 声明 “名称 “。将重复的 using 声明置为无效。

C5942 (W) enum bit-fields are always unsigned, but enum "名称" includes negative enumerator
枚举型的位字段总是 unsigned，但是在枚举型 “名称”中包含了负的枚举常数值。

C5946 (E) Name following "template" must be a member template
“template”之后的名称必须是成员模板。

C5947 (E) Name following "template" must have a template argument list
“template”之后的名称必须是模板实际参数。

C5948 (E) (W) Nonstandard local-class friend declaration -- no prior declaration in the enclosing scope
这是非标准格式的局部类友元声明。在类的定义中没有前方声明。

C5949 (I) Specifying a default argument on this declaration is nonstandard
在此声明中指定的默认参数不是标准格式。

C5951 (E) (W) Return type of function "main" must be "int"
main 函数的返回值必须是 int。

C5952 (E) A template parameter may not have class type
不能给模板的形式参数指定类 （class）型名。

C5953 (E) A default template argument cannot be specified on the declaration of a member of a class template
不能给类模板的成员声明指定默认的模板实际参数。

C5954 (E) A return statement is not allowed in a handler of a function try block of a constructor
在构造函数的 try 块的处理程序中，不允许返回语句。

C5955 (E) Ordinary and extended designators cannot be combined in an initializer designation
指示符不正确。

C5956 (E) The second subscript must not be smaller than the first
第 2 个下标必须大于第 1 个下标。

C5959 (W) Declared size for bit field is larger than the size of the bit field type; truncated to " 位数 " bits
指定的位数超过了位字段的型的 “长度”。按照位字段的型的长度，继续处理位数。

C5960 (E) Type used as constructor name does not match type " 型 "
用作构造函数名的型和 “型” 不同。

C5961 (W) Use of a type with no linkage to declare a variable with linkage
用未连接的型声明了连接的变量。视为连接的变量。

C5962 (W) Use of a type with no linkage to declare a function
用未连接的型声明了连接的函数。视为连接的变量。

C5963 (E) Return type may not be specified on a constructor
不能给构造函数指定返回型。

C5964 (E) Return type may not be specified on a destructor
不能给析构函数指定返回型。

C5965 (E) Incorrectly formed universal character name
universal character 的格式不正确。

C5966 (E) Universal character name specifies an invalid character
universal character 指定的字符不正确。

C5967 (E) A universal character name cannot designate a character in the basic character set
在基本字符集合中，不能将 universal character 指定为字符。

C5968 (E) This universal character is not allowed in an identifier
在标识符中不允许此 universal character。

C5969 (E) The identifier __VA_ARGS__ can only appear in the replacement lists of variadic macros

不能在有可变个数参数的宏的替换列表外记述 __VA_ARGS__ 标识符。

C5970 (W) The qualifier on this friend declaration is ignored

忽视此友元声明的限定词。

C5971 (E) Array range designators cannot be applied to dynamic initializers

数组范围名不能用于动态初始表达式。

C5972 (E) Property name cannot appear here

此处不能存在属性名。

C5973 (W) "inline" used as a function qualifier is ignored

忽视被用作函数限定词的 “inline”。

C5975 (E) A variable-length array type is not allowed

不能使用可变长数组型。

C5976 (E) A compound literal is not allowed in an integral constant expression

不能在整数常数表达式中使用复合文字。

C5977 (E) A compound literal of type "型" is not allowed

不能使用指定的复合文字型。

C5978 (E) A template friend declaration cannot be declared in a local class

不能在局部类中声明模板的友元函数。

C5979 (E) Ambiguous "?" operation: second operand of type "型1" can be converted to third operand type "型2", and vice versa

三元运算符 “?:” 的第 2 表达式的 “型 1” 和第 3 表达式的 “型 2” 是能相互转换的型，但是转换不明确。

C5980 (E) Call of an object of a class type without appropriate operator() or conversion functions to pointer-to-function type

调用了目标，但是没有定义指向 operator() 函数或者函数指针型的转换函数。

C5982 (E) There is more than one way an object of type "型" can be called for the argument list

至少有 2 个能从实际参数列表调用的 “型” 的目标。

C5983 (E) typedef name has already been declared (with similar type)

已用同等的型声明了 typedef 名。

C5984 (W) Operator new and operator delete cannot be given internal linkage

用 static 定义了 operator new/operator delete。

C5985 (E) Storage class "mutable" is not allowed for anonymous unions
不能将 mutable 指定为无名联合体。

C5987 (E) Abstract class type "型" is not allowed as catch type:
不能用 catch 接受抽象类。

C5988 (E) A qualified function type cannot be used to declare a nonmember function or a static member function
不能将带限定的函数型用于非成员函数和 static 成员函数的声明。

C5989 (E) A qualified function type cannot be used to declare a parameter
不能将带限定的函数型用于函数参数的指定。

C5990 (E) Cannot create a pointer or reference to qualified function type
不能建立指向带限定的函数型的指针型和参照型。

C5991 (W) Extra braces are nonstandard
在集合型的初始表达式列表中有多余的 “{”。

C5992 (E) Invalid macro definition:
这是非法的宏定义。

C5993 (W) Subtraction of pointer types "符号名 1" and "符号名 2" is nonstandard
指针型的符号 1 和符号 2 的减法运算不是标准格式。

C5994 (E) An empty template parameter list is not allowed in a template parameter declaration
不能将有空模板参数的模板指定为模板参数。

C5995 (E) Expected "class"
给模板参数指定的类模板需要类。

C5996 (E) The "class" keyword must be used when declaring a template parameter
给模板参数指定的类模板必须是结构体。

C5997 (W) "函数名 1" is hidden by "函数名 2" -- virtual function override intended?
“函数名 1”隐藏了 “函数名 2”。请确认是否覆盖了虚拟函数。

C5998 (E) A qualified name is not allowed for a friend declaration that is a function definition
不能在指定 friend 的函数定义中指定带名称空间的函数名。

C5999 (E) "型 1" is not compatible with "型 2"
指定的类模板和模板参数的格式不同。

C6000 (W) A storage class may not be specified here
在此不能指定存储区类的说明符。

C6001 (E) Class member designated by a using-declaration must be visible in a direct base class
在能参照的直接基类中，必须指定类成员的 using。

C6006 (E) A template parameter cannot have the same name as one of its template parameters
指定为模板参数的类模板名和自身的模板参数名相同。

C6007 (E) Recursive instantiation of default argument
递归生成模板函数的默认参数的示例。

C6009 (E) " 示例名 " is not an entity that can be defined
要生成没有实体的示例。

C6010 (E) Destructor name must be qualified
这是非法的析构函数名。

C6011 (E) Friend class name may not be introduced with "typename"
不能在 “typename” 之后记述友元类的名称。

C6012 (E) A using-declaration may not name a constructor or destructor
不能在 using 声明中指定构造函数或者析构函数。

C6013 (E) A qualified friend template declaration must refer to a specific previously declared template
需要在参照前预先定义限定友元模板。

C6014 (E) Invalid specifier in class template declaration
在类模板的声明中含有非法的说明符。

C6015 (E) Argument is incompatible with formal parameter
参数和已定义的参数没有兼容性。

C6017 (E) Loop in sequence of "operator->" functions starting at class " 符号 "
operator-> 不正确。

C6018 (E) " 类名 " has no member class " 成员名 "
使用了类中没有的成员。

C6019 (E) The global scope has no class named " 类名 "
在类的名称中使用了文件作用域运算符。

C6020 (E) Recursive instantiation of template default argument
在模板的默认参数中递归生成了示例。

C6021 (E) Access declarations and using-declarations cannot appear in unions
不能在 union 中使用 using 指定。

C6022 (E) " 名称 " is not a class member
不是类的成员。

C6023 (E) Nonstandard member constant declaration is not allowed
不能声明非标准格式的 const 成员。

C6028 (W) Invalid redeclaration of nested class
在类中二重定义了类。

C6029 (E) Type containing an unknown-size array is not allowed
有未确定大小数组的结构体或者联合体不能成为成员。

C6030 (W) A variable with static storage duration cannot be defined within an inline function
不能在 inline 函数中声明有静态作用域的变量。

C6031 (W) An entity with internal linkage cannot be referenced within an inline function with external linkage
不能在有外部连接的 inline 函数中参照有内部连接的标识符。

C6032 (E) Argument type " 型 " does not match this type-generic function macro
参数的型和通用函数生成宏的型不同。

C6034 (E) Friend declaration cannot add default arguments to previous declaration
在声明友元函数的情况下，不能在定义友元函数时插入默认参数。

C6035 (E) " 模板名 " cannot be declared in this scope
不能在此作用域中声明模板。

C6036 (E) The reserved identifier " 符号 " may only be used inside a function
在函数外使用了 __FUNC__。

C6037 (E) This universal character cannot begin an identifier
标识符名不能以此通用字符开头。

C6038 (E) Expected a string literal
没有字符串文字。

C6039 (E) Unrecognized STDC pragma
这是不能识别的 STDC 附注。

C6040 (E) Expected "ON", "OFF", or "DEFAULT"
没有 “ON”、“OFF” 或者 “DEFAULT”。

C6041 (E) A STDC pragma may only appear between declarations in the global scope or before any statements or declarations in a block scope
只能在全局作用域中的声明之间、表达式之间或者块作用域中的声明之间出现 STDC 附注。

C6042 (E) Incorrect use of va_copy
这是非法的 va_copy 宏的使用方法。

C6043 (E) " 型 " can only be used with floating-point types
“型” 用作浮点型以外的型。

C6044 (E) Complex type is not allowed
不能使用复数型。

C6045 (E) Invalid designator kind
这是非法的字段标识符。

C6046 (W) Floating-point value cannot be represented exactly
浮点数值有误差。

C6047 (E) Complex floating-point operation result is out of range
复数型浮点运算结果超出了能表示的值的范围。

C6048 (E) Conversion between real and imaginary yields zero
实数和虚数相互转换后的值为 0。

C6049 (E) An initializer cannot be specified for a flexible array member
不能给可变长数组成员指定初始表达式。

C6050 (W) imaginary *= imaginary sets the left-hand operand to zero
虚数 *= 虚数的左边值为 0。

C6051 (E) (W) Standard requires that " 符号 " be given a type by a subsequent declaration ("int" assumed)
不能使用隐含的型。

C6052 (E) A definition is required for inline " 符号 "
没有定义 inline 函数。

C6053 (W) Conversion from integer to smaller pointer
整数被转换为更小的指针。

C6054 (E) A floating-point type must be included in the type specifier for a _Complex or _Imaginary type

在复数或者虚数型的说明符中，必须包含浮点型。

C6055 (E) Types cannot be declared in anonymous unions

不能在无名联合体中声明型。

C6056 (W) Returning pointer to local variable

返回了指向局部变量的指针。

C6057 (W) Returning pointer to local temporary

返回了指向局部区域的指针。

C6061 (E) Declaration of "符号名" is incompatible with a declaration in another translation unit

“符号名”的声明和另一个翻译单位内的声明没有兼容性。

C6062 (E) The other declaration is "行"

有其他声明。

C6065 (E) A field declaration cannot have a type involving a variable length array

字段声明不能包含有可变长数组的型。

C6066 (E) declaration of "示例" had a different meaning during compilation of "符号"

编译时的声明不同。

C6067 (E) Expected "template"

没有“template”。

C6072 (E) (W) A declaration cannot have a label

声明不能有标号。

C6075 (E) "示例名" already defined during compilation of "符号"

已在编译时被定义。

C6076 (E) "符号" already defined in another translation unit

已被其他的翻译单位定义。

C6081 (E) A field with the same name as its class cannot be declared in a class with a user-declared constructor

不能声明和类名同名的成员。

C6083 (F) Exported template file "文件名" is corrupted

输出的模板文件破损。

C6086 (E) the object has cv-qualifiers that are not compatible with the member "符号"

有目标的 CV 限定词和成员 “符号”没有兼容性。

C6087 (E) No instance of "类名" matches the argument list and object (the object has cv-qualifiers that prevent a match)

“类名 “的示例和参数列表、目标不同 （有目标的 CV 限定词抑制了相同）。

C6089 (E) There is no type with the width specified

没有指定宽度的型。

C6105 (W) #warning directive: "字符串"

输出了 “字符串”。

C6139 (E) The "template" keyword used for syntactic disambiguation may only be used within a template

只能在 template 中将关键字 “template”用于消除语法中不明确的问题。

C6144 (E) Storage class must be auto or register

存储类必须是 auto 或者 register。

C6145 (W) "型 1" would have been promoted to “型 2” when passed through the ellipsis parameter; use the latter type instead

型 1 被扩展为型 2。使用型 2。

C6146 (E) "符号" is not a base class member

不是基类的成员。

C6151 (F) Mangled name is too long

被 mangle 的名称太长。

C6158 (E) void return type cannot be qualified

不能限定 void 型的返回值。

C6161 (E) A member template corresponding to "符号" is declared as a template of a different kind in another translation unit

模板声明和其他翻译单位不同。

C6163 (E) va_start should only appear in a function with an ellipsis parameter

只限有省略参数的函数能使用 va_start。

C6192 (W) Null (zero) character in input line ignored

忽视了输入行中的 null 字符。

C6193 (W) Null (zero) character in string or character constant

在字符串或者字符常数内含有 null 字符。

C6194 (W) Null (zero) character in header name
头文件名含有 null 字符。

C6197 (W) The prototype declaration of " 符号 " is ignored after this unprototyped redeclaration
忽视函数原型。

C6201 (E) Typedef " 符号 " may not be used in an elaborated type specifier
不能用于详述型说明符。

C6203 (E) Parameter "参数名" may not be redeclared in a catch clause of function try block
不能在 try 块的 catch 语句中重新声明 “参数名”。

C6204 (E) The initial explicit specialization of " 符号名 " must be declared in the namespace containing the template
必须在包含模板的名称空间中对符号进行最初显式特殊化的声明。

C6206 (E) "template" must be followed by an identifier
“template”之后需要标识符。

C6211 (W) Nonstandard cast to array type ignored
忽视了向非标准格式数组型的型转换。

C6212 (E) This pragma cannot be used in a _Pragma operator (a #pragma directive must be used)
不能在 _Pragma operator 内使用此附注 （必须使用 #pragma 指示）。

C6213 (W) Field uses tail padding of a base class
字段使用了基类的末尾填充。

C6218 (W) Base class " 类名 1" uses tail padding of base class " 类名 2"
基类 1 使用了基类 2 的末尾填充。

C6222 (W) Invalid error number
这是非法的错误号。

C6223 (W) Invalid error tag
这是非法的错误标记符。

C6224 (W)Expected an error number or error tag
没有错误号或者错误标记符。

C6227 (E) Transfer of control into a statement expression is not allowed
控制不能转到表达式语句。

C6229 (E) This statement is not allowed inside of a statement expression
不能在表达式语句中有此表达式。

C6230 (E) A non-POD class definition is not allowed inside of a statement expression
不能在表达式语句中定义非 POD 类。

C6235 (W) Nonstandard conversion between pointer to function and pointer to data
在指针函数和不完全的目标之间进行非标准格式的转换。

C6254 (E) Integer overflow in internal computation due to size or complexity of "型"
由于数据型的大小或者复杂性，内部计算结果发生了整数上溢。

C6255 (E) Integer overflow in internal computation
内部计算结果发生了整数上溢。

C6273 (W) Alignment-of operator applied to incomplete type
运算符的调整不适用于不完全的型。

C6280 (E) Conversion from inaccessible base class "类名" is not allowed
不能将派生类中私有继承的基类（class）型指针转换为继承型的指针。

C6282 (E) String literals with different character kinds cannot be concatenated
不能连接不同种类的字符串文字。

C6285 (W) Nonstandard qualified name in namespace member declaration
在名称空间的成员声明中使用了非标准格式的限定词名。

C6290 (W) Non-POD class type passed through ellipsis
非 POD 类型被传递给了省略符号。

C6291 (E) A non-POD class type cannot be fetched by va_arg
不能通过 va_arg 取得非 POD 型的类。

C6292 (E) The 'u' or 'U' suffix must appear before the 'l' or 'L' suffix in a fixed-point literal
在定点文字中，‘u’或者 ‘U’型的后缀必须出现在 ‘l’或者 ‘L’的后缀之前。

C6294 (W) Integer operand may cause fixed-point overflow
整数操作数有可能引起定点上溢。

C6295 (E) Fixed-point constant is out of range
定点常数超出了能表示的范围。

C6296 (W) Fixed-point value cannot be represented exactly
定点不能完整地表示 16 进制数。

C6297 (W) Constant is too large for long long; given unsigned long long type (nonstandard)
作为 long long 型，常数太大。更改为 Unsigned 的 long long 型（非标准格式）。

C6301 (W) “符号” declares a non-template function -- add <> to refer to a template instance
声明了非模板函数。

C6302 (W) Operation may cause fixed-point overflow
运算有可能引起定点上溢。

C6303 (E) Expression must have integral, enum, or fixed-point type
表达式必须含有整数型、枚举型或者定点型。

C6304 (E) Expression must have integral or fixed-point type
表达式必须含有整数型或者定点型。

C6307 (W) Class member typedef may not be redeclared
不能重新声明类成员的 typedef。

C6308 (W) Taking the address of a temporary
取得了局部区域的地址。

C6310 (W) Fixed-point value implicitly converted to floating-point type
定点值被隐含地转换为浮点型。

C6311 (E) Fixed-point types have no classification
没有区分浮点型。

C6312 (E) A template parameter may not have fixed-point type
不能给模板参数指定定点型。

C6313 (E) Hexadecimal floating-point constants are not allowed
不能使用 16 进制数的浮点常数。

C6315 (E) Floating-point value does not fit in required fixed-point type
浮点数值不在要求的定点型范围内。

C6316 (W) Value cannot be converted to fixed-point value exactly
如果将值变为定点值，就会产生误差。

C6317 (E) Fixed-point conversion resulted in a change of sign
将负整数值转换为定点型后的值变为正值。

C6318 (E) Integer value does not fit in required fixed-point type
整数值不在要求的定点型范围内。

C6319 (E) (W) Fixed-point operation result is out of range
定点运算结果超出了能表示的值的范围。

C6320 (E) Multiple named address spaces
存在多个同名的地址空间。

C6321 (E) Variable with automatic storage duration cannot be stored in a named address space
不能将有局部作用域的变量保存到命名地址空间。

C6322 (E) Type cannot be qualified with named address space
不能通过命名地址空间识别型。

C6323 (E) Function type cannot be qualified with named address space
不能通过命名地址空间识别函数型。

C6324 (E) Field type cannot be qualified with named address space
不能通过命名地址空间识别字段型。

C6325 (E) Fixed-point value does not fit in required floating-point type
定点值不在要求的浮点型范围内。

C6326 (E) Fixed-point value does not fit in required integer type
定点值不在要求的整数型范围内。

C6327 (E) Value does not fit in required fixed-point type
值不在要求的定点值范围内。

C6335 (F) Cannot open predefined macro file: "文件名"
不能打开已定义的宏文件。

C6336 (F) Invalid predefined macro entry at line "行数": "宏名"
在“行数”中有非法定义的宏的 entry 声明。

C6337 (F) Invalid macro mode name "宏模式名"
这是非法的宏模式名。

C6338 (F) Incompatible redefinition of predefined macro "宏名"
这是没有兼容性的已定义宏的重新定义。

C6342 (W) const_cast to enum type is nonstandard
通过 const_cast 转换为枚举型的型转换不是标准格式。

C6344 (E) A named address space qualifier is not allowed here
在此不能使用命名地址空间的标识符。

C6345 (E) An empty initializer is invalid for an array with unspecified bound
通过空的初始表达式对未指定边界的数组进行初始化，这是非法的。

C6346 (W) Function returns incomplete class type "类名"
函数返回了非法的类（class）型。

C6348 (I) Declaration hides "变量名"
局部变量被其他局部变量的声明隐藏。

C6349 (E) A parameter cannot be allocated in a named address space
不能在命名地址空间中分配参数。

C6350 (E) Invalid suffix on fixed-point or floating-point constant
在定点常数和浮点常数之后有非法的后缀。

C6351 (E) A register variable cannot be allocated in a named address space
不能在命名地址空间中分配寄存器变量。

C6352 (E) Expected "SAT" or "DEFAULT"
没有“SAT”或者“DEFAULT”。

C6353 (I) "符号名" has no corresponding member operator delete "符号名" (to be called if an exception is thrown during initialization of an allocated object)
符号没有和 new 运算符配对的 delete 运算符（在对取得的目标进行初始化时发生异常的情况下调用）

C6355 (E) A function return type cannot be qualified with a named address space
不能通过命名地址空间限定函数的返回值。

C6361 (W) Negation of an unsigned fixed-point value
将无符号的定点置为无效。

C6365 (E) Named-register variables cannot have void type
命名寄存器变量不能为 void 型。

C6372 (E) Nonstandard qualified name in global scope declaration
在全局作用域中声明了非标准格式的被限定的名称。

C6373 (W) Implicit conversion of a 64-bit integral type to a smaller integral type (potential portability problem)
64 位整数型被隐含地转换为更小的整数型。有可能产生移植性的问题。

C6374 (W) Explicit conversion of a 64-bit integral type to a smaller integral type (potential portability problem)

64 位整数型被明确地转换为更小的整数型。有可能产生移植性的问题。

C6375 (W) Conversion from pointer to same-sized integral type (potential portability problem)

从指针装换为相同长度的整数型。有可能产生移植性的问题。

C6380 (E) (I) Virtual "函数名" was not defined (and cannot be defined elsewhere because it is a member of an unnamed namespace)

没有定义虚拟函数，而且因为是无名空间的成员，所以不能在其他位置进行定义。

C6381 (E) (I) Carriage return character in source line outside of comment or character/string literal

在注释或者字符串文字以外的位置有换行字符。

C6382 (E) Expression must have fixed-point type

表达式必须含有定点。

C6386 (W) Storage specifier ignored

忽视存储类说明符。

C6396 (W) White space between backslash and newline in line splice ignored

忽视函数行接合部的反斜线符号和换行字符之间的间隔。

C6398 (E) Invalid member for anonymous member class -- class "符号" has a disallowed member function

对无名的成员类声明了非法的成员函数。

C6400 (W) Positional format specifier cannot be zero

不能给位置格式说明符指定 0。

C6403 (E) A variable-length array is not allowed in a function return type

不能将可变长数组作为函数的返回值的型。

C6404 (E) Variable-length array type is not allowed in pointer to member of type "型"

作为指向类成员的指针，禁止指向可变长数组型成员。

C6405 (E) The result of a statement expression cannot have a type involving a variable-length array

表达式语句的运算结果不能含有可变长数组型。

C6420 (E) (W) Some enumerator values cannot be represented by the integral type underlying the enum type

这是不能用整数型表示的枚举值。

C6421 (E) Default argument is not allowed on a friend class template declaration
不能在友元类的模板声明中指定默认参数。

C6422 (W) Multicharacter character literal (potential portability problem)
这是多字符文字，有可能引起移植性的问题。

C6424 (E) Second operand of offsetof must be a field
宏 offsetof 的第二个操作数必须是字段。

C6425 (E) Second operand of offsetof may not be a bit field
宏 offsetof 的第二个操作数不能是字段。

C6426 (E) Cannot apply offsetof to a member of a virtual base
不能将宏 offsetof 用于虚拟基类的成员。

C6427 (W) Offsetof applied to non-POD types is nonstandard
用于非 POD 型的宏 offsetof 不是标准格式。

C6428 (E) Default arguments are not allowed on a friend declaration of a member function
不能在友元声明的成员函数中指定默认参数。

C6429 (E) Default arguments are not allowed on friend declarations that are not definitions
不能在没有定义的友元声明中指定默认参数。

C6430 (E) Redeclaration of "函数名" previously declared as a friend with default arguments is not allowed
不能重新声明已被声明为有默认参数的友元函数。

C6431 (E) Invalid qualifier for "符号" (a derived class is not allowed here)
限定词不正确。

C6432 (E) Invalid qualifier for definition of class "类名"
给类的定义指定了非法的限定词。

C6439 (E) Template argument list of "符号" must match the parameter list
必须和模板参数列表相同。

C6440 (E) An incomplete class type is not allowed
这是不完全的类（class）型。

C6445 (E) Invalid redefinition of "符号名"
重新定义了枚举型。

C6449 (E) Explicit specialization of "符号" must precede its first use "符号 2"
已将模板具体化。

C6623 (W) The destructor for "类1" has been suppressed because the destructor for "类2" is inaccessible
因为不能存取类 2 的析构函数，所以抑制了类 1 的析构函数。

C6648 (W) '=' assumed following macro name "宏名" in command-line definition
视为在命令行定义内的宏名之后附加了 '='。

C6649 (E) (W) White space is required between the macro name "宏名" and its replacement text
在 "宏名" 及其替换文本之间需要间隔。

C6655 (E) "符号" cannot be declared inline after its definition "定义名"
因为抑制了 inline，所以不能将符号声明为 inline 函数。

C6671 (W) __assume expression with side effects discarded
放弃了有副作用的 __assume 表达式。

C6674 (E) __evenaccess qualifier is applied to only integer type
只能给整数型指定 __evenaccess 限定词。

C6675 (E) Expected a section name string
__sectop/__secend/__secsize 没有段名。

C6676 (E) Expected a section name
没有段名。

C6677 (E) Invalid pragma declaration
这是非法的 #pragma 语法。

C6678 (E) "符号名" has already been specified by other pragma
其他 #pragma 已指定了此符号。

C6679 (E) Pragma may not be specified after definition
不能在符号定义后的声明中指定 #pragma。

C6680 (E) Invalid kind of pragma is specified to this symbol
指定了非法的 #pragma。

C6681 (I) This pragma has no effect
此 #pragma 无效。

C6682 (E) "符号名" must be qualified for function type
符号必须是函数型。

C6683 (E) Illegal "附注名" specifier
这是非法的 #pragma。

C6684 (E) Multiple pointer qualifiers
指针型限定词有重复。

C6685 (E) __ptr16 must be qualified for data pointer type
__ptr16 不能限定数据指针型以外的型。

C6686 (E) Invalid binary digit
这是非法的 2 进制数。

C6687 (W) This pragma "名称" is ignored
忽视 “名称” 的 #pragma。

C6688 (E) "this" pointer of "类名" is cast implicitly to near pointer
用 near 指针隐含转换 “this”。

C6689 (E) Can not specify near or far for member
不能将成员函数指定为 near 或者 far。

C6690 (E) A member "函数名" qualified with near or far is declared
不能将成员函数指定为 near 或者 far。

C6691 (E) near or far specifier on a reference type is not allowed
不能指定 near 或者 far 为参照型。

C6692 (E) can not specify near or far for member function
不能将成员函数指定为 near 或者 far。

C6693 (E) can not specify near or far for function types
不能将函数型指定为 near 或者 far。

11.3 C 标准库函数的错误信息

在库函数中，如果在执行库函数的过程中发生了错误，就可能将错误号设定到标准库的头文件 <errno.h> 定义的宏 errno。

给错误号定义了对应的错误信息，能输出错误信息。输出错误信息的程序例子如下所示：

例

```
#include        <stdio.h>
#include        <string.h>
#include        <stdlib.h>
#include        <errno.h>

main()
{
    FILE *fp;

    fp=fopen("file", "w");
    fp=NULL;

    fclose(fp);                         /* error occurred      */

    printf("%s\n", strerror(errno));    /* print error message */
}
```

说明

(1) 由于将值为 NULL 的文件指针作为实际参数传递给 fclose 函数，所以出错。此时，给 errno 设定对应的错误号。

(2) 如果将错误号作为实际参数传递给 strerror 函数， strerror 函数就返回对应的错误信息的字符串指针。通过 printf 函数的字符串输出指定，输出错误信息。

标准库错误信息一览

错误号	错误信息 / 说明	设定错误号的函数
0x22 (ERANGE)	Data out of range 发生上溢。	Frexp、 ldexp、 modf、 ceil、 floor、 fmod、 atof、 atoi、 atol、 atoll、 atolfixed、 atolaccum、 strtod、 strtol、 strtoul、 strtoll、 stroull、 strtolfixed、 strtolaccum、 perror、 fprintf、 fscanf、 printf、 scanf、 sprintf、 sscanf、 vfprintf、 vprintf、 vsprintf、 acos、 acosf、 asin、 asinf、 atan、 atan2、 atan2f、 atanf、 ceilf、 cos、 cosf、 cosh、 coshf、 exp、 expf、 floorf、 fmodf, ldexpf、 log、 log10、 log10f、 logf、 modff、 pow、 powf、 sin、 sinf、 sinh、 sinhf、 sqrt、 sqrtf、 tan、 tanf、 tanh、 tanhf、 fabs、 fabsf、 frexpf
0x21 (EDOM)	Data out of domain 没有定义数学函数参数的结果值。	Acos、 acosf、 asin、 asinf、 atan、 atan2、 atan2f、 atanf、 ceil、 ceilf、 cos、 cosf、 cosh、 coshf、 exp、 expf、 floor、 floorf、 fmod、 fmodf、 ldexp、 ldexpf、 log、 log10、 log10f、 logf、 modf、 modff、 pow、 powf、 sin、 sinf、 sinh、 sinhf、 sqrt、 sqrtf、 tan、 tanf、 tanh、 tanhf、 fabs、 fabsf、 frexp、 frexpf
0x450 (ESTRN)	Too long string 字符串的字符数超过了 512 字符。	Atof、 atoi、 atol、 atoll、 atolfixed、 atolaccum、 strtod、 strtol、 strtoul、 strtoll、 stroull、 strtolfixed、 strtolaccum
0x04B0 (ECBASE)	Invalid radix 基数的指定有误。	Strtol、 strtoul、 strtoll、 strtoull
0x04B2 (ETLN)	Number too long 表示数值的字符串的字符数超过了有效位数。	Atof、 atolfixed、 atolaccum、 strtod、 strtolfixed、 strtolaccum、 fscanf、 scanf、 sscanf
0x04B4 (EEXP)	Exponent too large 指数的位数超过了 3 位。	Atof、 strtod、 fscanf、 scanf、 sscanf
0x04B6 (EEXPN)	Normalized exponent too large 在进行 1 次字符串的 IEEE 规格 10 进制格式的正规化时，指数的位数超过了 3 位。	Atof、 strtod、 fscanf、 scanf、 sscanf
0x04BA (EFLOATO)	Overflow out of float float 型的 10 进制数值超出了 float 型的范围（上溢）。	Fscanf、 scanf、 sscanf
0x04C4 (EFLOATU)	Underflow out of float float 型的 10 进制数值超出了 float 型的范围（下溢）。	Fscanf、 scanf、 sscanf
0x04E2 (EDBLO)	Overflow out of double double 型的 10 进制数值超出了 double 型的范围 （上溢）。	Fscanf、 scanf、 sscanf
0x04EC (EDBLU)	Underflow out of double double 型的 10 进制数值超出了 double 型的范围 （下溢）。	Fscanf、 scanf、 sscanf
0x04F6 (ELDBLO)	Overflow out of long double long double 型的 10 进制数值超出了 long double 型的范围 （上溢）。	Fscanf、 scanf、 sscanf
0x0500 (ELDBLU)	Underflow out of long double long double 型的 10 进制数值超出了 long double 型的范围 （下溢）。	Fscanf、 scanf、 sscanf

12. 汇编程序的错误信息

12.1 错误格式和错误级

本章节说明用以下格式输出的错误信息和错误内容。

错误号 （错误级） 错误信息
错误内容

根据错误的重要程度，错误级分为 3 种。

错误级		运行
(W)	警告	继续处理。
(E)	错误	中止处理。
(F)	致命错误	中止处理。

12.2 信息一览

A1000 (W) '.ALIGN' with not 'ALIGN' specified relocatable section
在没有指定 ALIGN 的段中记述了控制指令 “.ALIGN”。
请确认控制指令 “.ALIGN”的记述位置。请在记述控制指令 “.ALIGN”的段定义行记述 ALIGN 的指定。

A1001 (W) Destination address may be changed
有可能和转移目标期待的位置不同。
为了不选择最佳的寻址方式，请记述转移指令的操作数。

A1002 (W) Floating point value is out of range
浮点数不在范围内。
请确认浮点数的记述。忽视范围外的内容。

A1003 (W) Location counter exceed
定位计数器超过了 0FFFFFFFFh。
请确认 .ORG 操作数的值。请重新记述源文件。

A1004 (W) '.ALIGN' size is different
调整值不同。
请确认调整值。

A1006 (W) Data in 'CODE' section align in 4byte
在 endian=big 时，将 CODE 段中的数据区的起始位置调整为 4 字节边界。

A1007 (W) Data size in 'CODE' section align in 4byte
在 endian=big 时，将 CODE 段中的数据区的大小调整为 4 的倍数。

A1009 (W) Multiple symbols
用 .STACK 重复指定了符号的堆栈值。

A1010 (W) Section attribute mismatch
段的属性不同。

A1011 (W) Use PM instruction
使用了特权指令。

A1012 (W) Use FPU instruction
使用了浮点运算指令。

A1013 (W) Use DSP instruction
使用了 DSP 功能指令。

A1014 (W) Too many actual macro parameters
宏的实际参数的个数太多。
忽视多余的实际参数。

A1015 (W) Actual macro parameters are not enough
宏的实际参数的个数少于宏的形式参数的个数。
没有对应的实际参数的形式参数无效。

A1016 (W) '.END' statement is in include file
在 include 文件中记述了 .END。
不能在 include 文件中记述 .END。请删除记述。
忽视 .END。

A2000 (E) No space after mnemonic or directive
助记符和汇编控制指令之后没有空白字符。
请在指令和操作数之间记述空白字符。

A2001 (E) ',' is missing
没有记述 “,”。
请在操作数之间记述逗号。

A2002 (E) Characters exist in expression
在指令或者表达式中有多余的字符。
请确认表达式的记述规则。

A2003 (E) Size specifier is missing
没有长度说明符。
请记述长度说明符。

A2004 (E) Invalid operand(s) exist in instruction
指令中有无效的操作数。
请确认指令操作数的记述方法并且重新记述。

A2005 (E) Operand type is not appropriate

操作数的种类有误。

请确认述操作数的记述方法并且重新记。

A2006 (E) Size specifier is not appropriate

长度说明符的记述有误。

请重新记述长度说明符。

A2007 (E) Operand label is not in the same section

转移目标不在同一个段中。

只能转移到同一个段中的转移目标。请重新记述助记符。

A2008 (E) Illegal displacement value

位移量的值有误。

当长度说明符为 W 时，请指定 2 的倍数；当长度说明符为 L 时，请指定 4 的倍数。

A2022 (E) Symbol name is missing

在 EQU 控制指令行没有记述符号名。

A2023 (E) Illegal directive command is used

记述了非法的控制指令。

请重新记述正确的控制指令。

A2024 (E) No ';' at the top of comment

没有在注释的开头记述 “;”。

请在注释的开头记述分号。请确认助记符或者操作数的记述是否有误。

A2026 (E) 'CODE' section in big endian is not appropriate

在 endian=big 时，给绝对属性的 CODE 段的起始地址指定了不是 4 的倍数的值。

请给绝对属性的 CODE 段的起始地址指定 4 的倍数的值。

A2027 (E) Illegal character code

字符代码不正确。

A2028 (E) Unrecognized character escape sequence

有不能识别的转义序列。

A2040 (E) Include nesting over

包含的嵌套层太深。

请重新记述，使包含的嵌套层小于等于 9。

A2041 (E) Can't open include file 'XXXX'

不能打开 include 文件。

请确认 include 文件名和 include 文件的保存目录。

A2042 (E) Including the include file in itself

在 include 文件中包含了自身。

请确认并且重新记述 include 文件名。

A2049 (E) Invalid reserved word exist in operand

在操作数中记述了保留字。

不能在操作数中记述保留字。请重新记述操作数。

A2050 (E) Operand value is not defined

操作数的值是未定义的值。

请重新给操作数记述确定的值。

A2051 (E) '{' is missing

没有记述 ‘{’。

A2052 (E) Addressing mode specifier is not appropriate

寻址方式说明符的记述有误。

请确认寻址方式说明符的记述方法。

A2053 (E) Reserved word is missing

没有记述保留字。

A2054 (E) ']' is missing

没有记述 ‘]’。

请记述 ‘[’ 对应的 ‘]’。

A2055 (E) Right quote is missing

没有右侧引用符。

请记述引用符。

A2056 (E) The value is not constant

在汇编时值不确定。

请记述能在汇编时确定的表达式、符号名或者标号名。

A2057 (E) Quote is missing

没有对字符串记述引用符。

请用引用符将字符串括起来记述。

A2058 (E) Illegal operand is used

操作数错误。

请确认操作数的记述方法并且重新记述。

A2059 (E) Operand number is not enough

操作数不够。

请确认操作数的记述方法并且重新记述。

A2060 (E) Too many macro nesting

宏的嵌套层太多。

宏的嵌套层不能超过 65535 个。请确认源记述。

A2061 (E) Too many macro local label definition

宏内的局部标号的定义太多。

在 1 个文件中宏内的局部标号个数不能超过 65535 个。

A2062 (E) '.MACRO' is missing for '.ENDM'

没有 .ENDM 对应的 .MACRO。

请确认 .ENDM 的记述位置。

A2063 (E) '.MREPEAT' is missing for '.ENDR'

没有 .ENDR 对应的 .MREPEAT。

请确认 .ENDR 的记述位置。

A2064 (E) '.MACRO' or '.MREPEAT' is missing for '.EXITM'

没有 .EXITM 对应的 .MACRO 或者 .MREPEAT。

请确认 .EXITM 的记述位置。

A2065 (E) No macro name

没有宏名。

请给宏定义记述宏名。

A2066 (E) Too many formal parameter

宏的形式参数的定义个数太多。

宏的形式参数的个数不能超过 80 个。

A2067 (E) Illegal macro parameter

宏参数中有非法的记述。

请确认宏参数的记述内容。

A2068 (E) Source line is too long

源行太长。

请确认源行的记述内容。

A2069 (E) '.MACRO' is missing for '.LOCAL'

没有 .LOCAL 对应的 .MACRO。

请确认 .LOCAL 的记述位置。只能在宏的块内记述 .LOCAL。

A2070 (E) No '.ENDM' statement

没有记述 .ENDM。

请确认 .ENDM 的记述位置并且记述 .ENDM。

A2071 (E) No '.ENDR' statement
没有记述 .ENDR。
请确认 .ENDR 的记述位置并且记述 .ENDR。

A2072 (E) ')' is missing
没有记述 ‘)’。
请记述 ‘(’ 对应的 ‘)’。

A2073 (E) Operand expression is not completed
操作数的记述不够。
请确认操作数的记述方法并且重新记述。

A2074 (E) Syntax error in expression
表达式的记述有误。
请确认表达式的记述方法并且重新记述。

A2075 (E) String value exist in expression
在表达式中记述了字符串表达式。
请重新记述表达式。

A2076 (E) Division by zero
进行了被 0 除的运算。
请重新记述表达式。

A2077 (E) No '.END' statement
没有记述 .END。
请在源程序的结束行记述 .END。

A2078 (E) The specified address overlaps at ' 地址值 '
在指定的 ‘地址值’ 中地址分配有重叠。
请检查 .ORG 和 .OFFSET 的指定内容。
在 C/C++ 源的情况下，‘地址值’ 的多个变量有重复。
请确认要分配给 ‘地址值’ 的变量。

A2080 (E) '.IF' is missing for '.ELSE'
没有 .ELSE 对应的 .IF。
请确认 .ELSE 的记述位置。

A2081 (E) '.IF' is missing for '.ELIF'
没有 .ELIF 对应的 .IF。
请确认 .ELIF 的记述位置。

A2082 (E) '.IF' is missing for '.ENDIF'
没有 .ENDIF 对应的 .IF。
请确认 .ENDIF 的记述位置。

A2083 (E) Too many nesting level of condition assemble
条件汇编的嵌套太多。
请确认条件汇编的记述。

A2084 (E) No '.ENDIF' statement
在源文件中没有 IF 语句对应的 ENDIF。
请确认源文件的记述。

A2088 (E) Can't open '.ASSERT' message file 'XXXX'
不能打开 .ASSERT 的输出文件。
请确认文件名。

A2089 (E) Can't write '.ASSERT' message file 'XXXX'
不能写 .ASSERT 的输出文件。
请确认此文件的权限。

A2090 (E) Too many temporary label
临时标号的个数太多。
请在将临时标号替换为标号名后进行记述。

A2091 (E) Temporary label is undefined
未定义临时标号。
请定义临时标号。

A2100 (E) Value is out of range
值不在范围内。
请根据寄存器等的位长记述值。

A2111 (E) Symbol is undefined
未定义符号。
不能使用未定义的符号名。不能记述前方参照的符号名。
请确认符号名。

A2112 (E) Symbol is missing
没有记述符号。
请记述符号名。

A2113 (E) Symbol definition is not appropriate
符号的定义有误。
请确认符号的定义方法并且重新记述。

A2114 (E) Symbol has already defined as another type
已由不同的控制指令定义了同名的符号。
请更改符号名。

A2115 (E) Symbol has already defined as the same type

已定义了符号。

请更改符号名。

A2116 (E) Symbol is multiple defined

二重定义了符号。宏名和其他名称有重复。

请更改符号名。

A2117 (E) Invalid label definition

记述了无效的标号。

请重新记述标号的定义。

A2118 (E) Invalid symbol definition

记述了无效的符号。

请重新记述符号的定义。

A2119 (E) Reserved word is used as label or symbol

在标号或者符号中使用了保留字。

请重新记述标号或者符号名。

A2130 (E) No '.SECTION' statement

没有记述 ‘.SECTION’。

请给源程序至少记述 1 个 .SECTION。

A2131 (E) Section type is not appropriate

段属性的记述错误。

请重新记述段的属性。

A2132 (E) Section has already determined as attribute

段已被确定为相对属性。不能记述控制指令 “.ORG”。

请确认段的属性。

A2133 (E) Section attribute is not defined

未定义段的属性。不能在此段中记述控制指令 “.ALIGN”。

请在绝对地址属性的段或者指定了 ALIGN 的相对地址属性的段中记述控制指令 “.ALIGN”。

A2134 (E) Section name is missing

没有段名。

请给操作数记述段名。

A2135 (E) 'ALIGN' is multiple specified in '.SECTION'

在 .SECTION 定义行中指定了多个 ‘ALIGN’。

请删除多余的 ‘ALIGN’。

A2136 (E) Section type is multiple specified

在段定义行中重复指定了段属性。

只能在段定义行中记述 1 个 “CODE”、“DATA”、“ROMDATA” 的指定。

A2137 (E) Too many operand

有多余的操作数。

请确认操作数的记述内容。

A3000 (F) Can't create file 'filename'

不能生成 ‘filename’ 文件。

请确认目录容量。

A3001 (F) Can't open file 'filename'

不能打开 ‘filename’ 文件。

请确认文件名。

A3002 (F) Can't write file 'filename'

不能读 ‘filename’ 文件。

请确认此文件的权限。

A3003 (F) Can't read file 'filename'

不能读文件。

请确认此文件的权限。

A3004 (F) Can't create Temporary file

不能生成临时文件。

为了在当前目录以外的位置建立临时文件，请给环境变量 ‘TMP_RX’ 指定目录。

A3005 (F) Can't open Temporary file

不能打开临时文件。

请确认 ‘TMP_RX’ 指定的目录。

A3006 (F) Can't read Temporary file

不能读临时文件。

请确认 ‘TMP_RX’ 指定的目录。

A3007 (F) Can't write Temporary file

不能写临时文件。

请确认 ‘TMP_RX’ 指定的目录。

A3008 (F) Illegal file name 'filename'

这是非法的文件名。

请按文件名的记述规则指定文件名。

A3100 (F) Command line is too long
命令行的字符数太多。
请重新输入命令。

A3101 (F) Invalid option 'xx' is used
使用了无效的命令选项 xx。
不存在指定的选项。请重新输入命令。

A3102 (F) Ignore option 'xx'
指定了无效的选项。

A3103 (F) Option 'xx' is not appropriate
命令选项 xx 的记述不正确。
请重新指定命令选项。

A3104 (F) No input files specified
没有指定输入文件。
请指定输入文件。

A3105 (F) Source files number exceed 80
文件的个数超过了 80 个。
请分次执行汇编程序。

A3106 (F) Lacking cpu specification
没有指定 CPU。
请通过 cpu 选项或者环境变量 CPU_RX 指定 CPU。

A3110 (F) Multiple register base/fint_register
Base 和 fint_register 选项指定的寄存器有重复。

A3200 (F) Error occurred in executing 'xxx'
在执行 xxx 时发生了错误。
请重新执行 asrx。

A3201 (F) Not enough memory
存储器容量不够。
请在将文件分割后重新执行文件或者增加存储器容量。

A3202 (F) Can't find work dir
找不到工作目录。

A4000-A4999 (-) Internal error
在汇编程序的内部处理过程中发生了某种故障。请和购入本产品的营业部分或者代理店联系，反映错误的发生情况。

13. 优化连接编辑程序的错误信息

13.1 错误格式和错误级

本章节说明用以下格式输出的错误信息和错误内容。

错误号 (错误级) 错误信息
　　错误内容

根据错误的重要程度，错误级分为 5 种。

	错误级		运行
L0000 - L0999 P0000 - P0999	(I)	信息	继续处理。
L1000 - L1999 P1000 - P1999	(W)	警告	继续处理。
L2000 - L2999 P2000 - P2999	(E)	错误	继续选项的解析处理，中止处理。
L3000 - L3999 P3000 - P3999	(F)	致命错误	中止处理。
L4000 - P4000 -	(-)	内部错误	中止处理。

以 L 开头的错误号是优化连接编辑程序的输出信息。

以 P 开头的错误号是预连接程序的输出信息。不能通过 nomessage 选项或者 change_message 选项指定以 P 开头的错误号。

13.2 信息一览

L0001 (I) Section "段" created by optimization "优化"
　　因 “优化” 的优化而建立了 “段”。

L0002 (I) Symbol "符号" created by optimization "优化"
　　因 “优化” 的优化而建立了 “符号”。

L0003 (I) "文件"-"符号" moved to "段" by optimization
　　因 variable_access 的优化而移动了 “文件” 中的 “符号”。

L0004 (I) "文件"-"符号" deleted by optimization
　　因 symbol_delete 的优化而删除了 “文件” 中的 “符号”。

L0005 (I) The offset value from the symbol location has been changed by optimization : "文件"-"段"-"符号 ±offset"
　　在 “符号 ±offset” 的范围内，因优化使长度发生了变化，所以更改了 offset 值。请确认是否有问题。要想抑制 offset 值的变更时，请在汇编 “文件” 时解除 goptimize 选项的指定。

L0100 (I) No inter-module optimization information in "文件"
在“文件”中没有模块之间的优化信息，不将“文件”作为模块之间的优化对象。如果想成为模块之间的优化对象，就请在编译或者汇编时指定 goptimize 选项。但是， asmsh 没有 goptimize 选项。

L0101 (I) No stack information in "文件"
在“文件”中没有堆栈信息。“文件”可能是汇编程序的输出文件或者 SYSROF->ELF 转换文件。优化连接编辑程序输出的堆栈信息文件不包含该文件的内容。

L0102 (I) Stack size "大小" specified to the undefined symbol "符号" in "文件"
给“文件”中的未定义符号的“符号”指定了堆栈大小“大小”。

L0103 (I) Multiple stack sizes specified to the symbol "符号"
给符号的“符号”指定了多个堆栈大小。

P0200 (I) "示例" no longer needed in "文件"
“文件”中有不使用的“示例”。

P0201 (I) "示例" assigned to file "文件"
将“示例”分配到“文件”。

P0202 (I) Executing: "命令"
为了生成示例而执行了“命令”。

P0203 (I) "示例" adopted by file "文件"
“示例”被分配到“文件”。

L0300 (I) Mode type "模式类型 1" in "文件" differ from "模式类型 2"
输入了不同模式类型的文件。

L0400 (I) Unused symbol "文件"-"符号"
没有使用“文件”中的“符号”。

L0500 (I) Generated CRC code at "地址"
CRC 码被输出到“地址”。

L0510 (I) Section "段" was moved other area specified in option "cpu=< 存储器属性 >"
不分割段而根据 cpu=< 存储器属性 > 分配了“段”。

L0511 (I) Sections "段名"," 分割后的段名" are Non-contiguous
分割“段名”的段并且生成了“分割后的段名”的段。

L1000 (W) Option "选项" ignored
“选项”无效。忽视“选项”。

L1001 (W) Option "选项1" is ineffective without option "选项2"
“选项1”需要“选项2”。忽视“选项1”。

L1002 (W) Option "选项1" cannot be combined with option "选项2"
不能同时指定“选项1”和“选项2”。忽视“选项1”。

L1003 (W) Divided output file cannot be combined with option "选项"
在指定“选项”时，不能指定输出文件的分割。忽视选项的指定。将第1个输入的文件名用作输出文件名。

L1004 (W) Fatal level message cannot be changed to other level : "序号"
不能更改Fatal级信息的级。忽视“序号”的指定。能通过change_message选项更改Information/Warning/Error级的错误。

L1005 (W) Subcommand file terminated with end option instead of exit option
没有在end选项之后指定处理。假设exit选项进行处理。

L1006 (W) Options following exit option ignored
忽视了exit选项之后的选项。

L1007 (W) Duplicate option : "选项"
“选项”有重复。将最后指定的选项置为有效。

L1008 (W) Option "选项" is effective only in cpu type "单片机类型"
“选项”只在“单片机类型“时有效。忽视“选项”。

L1010 (W) Duplicate file specified in option "选项" : "文件名"
“选项”指定了2次相同的文件。忽视第2次的指定。

L1011 (W) Duplicate module specified in option "选项" : "模块"
“选项”指定了2次相同的模块。忽视第2次的指定。

L1012 (W) Duplicate symbol/section specified in option "选项" : "名称"
“选项”指定了2次相同的符号名或者段名。忽视第2次的指定。

L1013 (W) Duplicate number specified in option "选项" : "序号"
“选项”指定了相同的错误号。将最后指定的错误号置为有效。

L1100 (W) Cannot find "名称" specified in option "选项"
找不到“选项”指定的符号名或者段名。忽视“名称”的指定。

L1101 (W) "名称" in rename option conflicts between symbol and section
rename选项指定的“名称”有段名和符号名。
将符号名设定为变更对象。

L1102 (W) Symbol "符号" redefined in option "选项"
已定义了 “选项” 指定的符号。照样继续处理。

L1103 (W) Invalid address value specified in option "选项" : "地址"
“选项” 指定的 “地址” 是无效值。忽视 “地址” 的指定。

L1104 (W) Invalid section specified in option "选项" : "段"
不能给 “选项” 指定没有初始值的段。
忽视 “段” 的指定。

L1110 (W) Entry symbol "符号" in entry option conflicts
在编译或者汇编时将 entry 选项指定的 “符号” 以外的符号指定为入口符号。选项的指定优先。

L1120 (W) Section address is not assigned to "段"
没有指定 “段” 的地址。将 “段” 分配到最后。

L1121 (W) Address cannot be assigned to absolute section "段" in start option
“段” 是绝对地址段。忽视绝对地址段的地址指定。

L1122 (W) Section address in start option is incompatible with alignment : "段"
start 选项指定的 “段” 的地址和调整数发生矛盾。根据调整数校正段地址。

L1130 (W) Section attribute mismatch in rom option : "段 1, 段 2"
rom 选项指定的 “段 1” 和 “段 2” 的属性、调整数不同。作为 “段 2” 的调整数，将较大的一方置为有效。

L1140 (W) Load address overflowed out of record-type in option "选项"
指定了小于地址值的 record 格式。对于超出指定的 record 格式的范围，用其他 record 格式进行输出。

L1141 (W) Cannot fill unused area from "地址" with the specified value
空区域的大小不是 space 选项指定的值的倍数，所以不能将指定的数据输出到 “地址” 之后。

L1150 (W) Sections in fsymbol option have no symbol
fsymbol 选项指定的段中没有外部定义符号。忽视 fsymbol 选项。

L1160 (W) Undefined external symbol "符号"
参照了未定义的 “符号”。

L1170 (W) Specified SBR addresses conflict
指定了多个不同的 SBR 地址。作为 SBR=USER 进行处理。

L1171 (W) Least significant byte in SBR="常数" ignored
SBR 选项指定的地址 “常数” 的低 8bit 无效。

L1182 (W) Cannot generate vector table section "段"
在输入文件中有向量表 “段”。连接程序不自动生成 “段”。

L1183 (W) Interrupt number "向量号" of "段" is defined in input file
已在输入文件中定义了 VECTN 选项记述的向量号。输入文件的内容优先，继续处理。

L1190 (W) Section "段" was moved other area specified in option "cpu=<存储器属性>"
因为通过外部变量的存取优化更改了目标存储器容量，所以移动了下一个 cpu 指定范围的 “段”。

L1191 (W) Area of "FIX" is within the range of the area specified by "cpu=<存储器属性>" :"<start>-<end>"
在 cpu 选项中，存储器属性 FIX 和 FIX 以外的 <start>-<end> 范围有重叠，所以将 FIX 置为有效。

L1192 (W) Bss Section "段名" is not initialized
不能通过初始设定程序对没有初始值的数据段的 “段名” 进行初始化。请检查 -cpu 指定的范围和指针变量的大小。

L1193 (W) Section "段名" specified in option "选项" is ignored
对于通过 -cpu=stride 功能分割的段的后半部分，“选项” 的指定无效。请不要通过 “选项” 指定后半部分的段。

L1194 (W) Section "段" in relocation "文件"-"段"-"偏移量" is changed.
为了参照被分割的后半部分的段，更改了参照 “段”“文件”“偏移量” 位置中的 “段” 的再定位。要不想分割时，就请通过 contiguous_section 选项指定 “段”。

L1200 (W) Backed up file "文件 1" into "文件 2"
将 “文件 1” 被备份到 “文件 2”。

L1300 (W) No debug information in input files
在输入文件中没有调试信息。忽视 debug、sdebug 和 compress 选项的指定。请确认在编译或者汇编时是否指定了相应的选项。

L1301 (W) No inter-module optimization information in input files
在输入文件中没有模块之间的优化信息。忽视 optimize 选项。请在编译和汇编时指定 goptimize 选项。

L1302 (W) No stack information in input files
在输入文件中没有堆栈信息。忽视 stack 选项。当输入文件为汇编程序的输出文件或者 SYSROF->ELF 转换文件时， stack 选项无效。

L1303 (W) No rts information in input files
没有能生成 .rts 文件的输入文件。
不生成 .rts 文件而结束处理。

L1305 (W) Entry address in "文件" conflicts : "地址"
多次输入了不同入口地址的文件。

L1310 (W) "段" in "文件" is not supported in this tool
在 “文件” 中有不支持的段。忽视 “段”。

L1311 (W) Invalid debug information format in "文件"
“文件” 中的调试信息不是 dwarf2。删除 debug 信息。

L1320 (W) Duplicate symbol "符号" in "文件"
“符号” 有重复。先输入的文件中的符号优先。

L1321 (W) Entry symbol "符号" in "文件" conflicts
多次输入了定义入口符号的目标文件。将先输入的文件中的入口符号置为有效。

L1322 (W) Section alignment mismatch : "段"
输入了不同调整数的同名段。将指定的最大调整数置为有效。

L1323 (W) Section attribute mismatch : "段"
输入了不同属性的同名段。在绝对段和相对段的情况下，作为绝对段处理。如果 read/write 属性不同，两者就都允许。

L1324 (W) Symbol size mismatch : "符号" in "文件"
输入了不同大小的公共符号或者定义符号。定义符号优先。在都为公共符号时，先输入的文件中的符号优先。

L1325 (W) Symbol attribute mismatch : "符号": "文件"
“文件” 中的 “符号” 的属性和其他文件的同名符号的属性不同。请确认符号。

L1326 (W) Reserved symbol "符号” is defined in "文件”
在 “文件” 中定义了被保留的名称的符号 “符号”。

L1330 (W) Cpu type "单片机类型 1" in "文件" differ from "单片机类型 2"
输入了不同单片机类型的文件。将单片机类型作为 H8SX 继续处理。

L1400 (W) Stack size overflow in register optimization
在寄存器优化中，堆栈存取代码超过了编译程序的堆栈量极限值。忽视寄存器优化的指定。

L1401 (W) Function call nest too deep
因为函数的调用嵌套太深，所以不能进行寄存器的优化。

L1402 (W) Parentheses specified in option "start" with optimization
当通过 start 选项记述括弧 “()” 时，不能使用优化功能。
将优化功能置为无效。

L1410 (W) Cannot optimize "文件"-"段" due to multi label relocation operation
不能对有多个标号的再定位运算的段进行优化。不将 “文件”中的 “段”作为优化对象。

L1420 (W) "文件" is newer than "配置"
在 “配置”之后更新了 “文件”。忽视配置信息。

L1430 (W) Cannot generate effective bls file for compiler optimization
生成了无效的 bls 文件。在编译时，即使指定外部变量的存取优化 （map 选项），也不能进行此优化。
编译程序的外部变量存取优化 （map 选项）有以下限制。请确认是否有对应的内容并且检查段的分配。
在编译时使用了 base 选项的情况下，如果将数据段分配到代码段之后，就有可能不能进行外部变量的存取优化。
※ bls 文件是指 “外部符号的分配信息文件”，此信息文件用于编译程序的 map 选项。

L1500 (W) Cannot check stack size
因为没有堆栈段，所以不能检查编译时 stack 选项指定堆栈大小的一致性。为了检查编译时的 stack 选项的一致性，需要在编译时和汇编时指定 goptimize 选项。

L1501 (W) Stack size overflow : "堆栈大小"
堆栈段的大小超过了编译时 stack 选项指定的 “堆栈大小”。请更改编译时的选项，或者更改程序以便能减少堆栈量。

L1502 (W) Stack size in "文件" conflicts with that in another file
在多个文件中指定了不同的堆栈大小。请确认编译时的选项。

L1510 (W) Input file was compiled with option "smap" and option "map" is specified at linkage
有通过指定 “smap”进行编译的文件。对于指定 smap 的文件，不能在第 2 次创建时指定 map 选项进行编译。

P1600 (W) An error occurred during name decoding of "示例"
“示例”不能解码。用编码名输出信息。

L2000 (E) Invalid option : "选项"
不支持 “选项”。

L2001 (E) Option "选项" cannot be specified on command line
不能在命令行指定 “选项”。请在子命令文件中指定。

L2002 (E) Input option cannot be specified on command line
在命令行指定了 input 选项。请在命令行指定没有 input 选项的输入文件。

L2003 (E) Subcommand option cannot be specified in subcommand file
在子命令文件中指定了 subcommand 选项。不能嵌套 subcommand 选项。

L2004 (E) Option "选项 1" cannot be combined with option "选项 2"
不能同时指定 “选项 1” 和 “选项 2”。

L2005 (E) Option "选项" cannot be specified while processing "进程"
不能给 “进程” 处理指定 “选项”。

L2006 (E) Option "选项 1" is ineffective without option "选项 2"
“选项 1” 需要 “选项 2”。

L2010 (E) Option "选项" requires parameter
“选项” 需要指定参数。

L2011 (E) Invalid parameter specified in option "选项" : "参数"
“选项” 指定了无效的参数。

L2012 (E) Invalid number specified in option "选项" : "值"
“选项” 指定了无效值。请确认值的范围。

L2013 (E) Invalid address value specified in option "选项" : "地址"
“选项” 指定的 “地址” 是无效值。请用 0 ～ FFFFFFFF 之间的 16 进制数指定。

L2014 (E) Illegal symbol/section name specified in "选项" : "名称"
“选项” 指定的段或者符号名使用了非法字符。段 / 符号名能使用数字、英字、 _ 或者 $（不以数字开头）。

L2016 (E) Invalid alignment value specified in option "选项" : "调整数"
“选项” 指定的 “调整数” 无效。
请指定 1、 2、 4、 8、 16 或者 32。

L2017 (E) Cannot output “段” specified in option "选项"
不能输出由 “选项” 指定 “段” 的一部分代码。通过转换指令码的字节序，使 “段” 中的一部分指令码变为不连续状态。对于非连续部分的指令码所属的段，请通过 4 字节边界从连接列表确认段地址，然后确认要输出的段是和哪个段进行了字节序的转换。

L2020 (E) Duplicate file specified in option "选项" : "文件"
“选项” 指定了 2 次相同的文件。

L2021 (E) Duplicate symbol/section specified in option "选项" : "名称"
“选项” 指定了 2 次相同的符号名或者段名。

L2022 (E) Address ranges overlap in option "选项" : "地址范围"
“选项” 指定的 “地址范围” 有重叠。

L2100 (E) Invalid address specified in cpu option : "地址"
cpu 选项指定了 cpu 不能指定的地址。

L2101 (E) Invalid address specified in option "选项" : "地址"
"选项"指定的"地址"超过了 cpu 能指定的地址范围或者 cpu 选项指定的范围。

L2110 (E) Section size of second parameter in rom option is not 0 : "段"
给 rom 选项的第 2 个参数指定了长度不为 0 的"段"。

L2111 (E) Absolute section cannot be specified in rom option : "段"
rom 选项指定了绝对地址段。

L2120 (E) Library "文件" without module name specified as input file
将没有模块名的库文件指定为输入文件。

L2121 (E) Input file is not library file : "文件(模块)"
输入文件指定的"文件（模块）"不是库文件。

L2130 (E) Cannot find file specified in option "选项" : "文件"
找不到"选项"指定的文件。

L2131 (E) Cannot find module specified in option "选项" : "模块"
没有"选项"指定的模块。

L2132 (E) Cannot find "名称" specified in option "选项"
不存在"选项"指定的符号或者段。

L2133 (E) Cannot find defined symbol "名称" in option "选项"
不存在"选项"指定的外部定义符号。

L2140 (E) Symbol/section "名称" redefined in option "选项"
已定义了"选项"指定的符号或者段。

L2141 (E) Module "模块" redefined in option "选项"
已注册了"选项"指定的模块。

L2142 (E) Interrupt number "向量号" of "段" has multiple definition
多次输入了向量表"段"的向量号定义。只能给向量号设定一个地址。请检查源文件的记述。

L2200* (E) Illegal object file : "文件"
输入了非 ELF 格式。
* 有可能显示 P2200。

L2201 (E) Illegal library file : "文件"
"文件"不是库文件。

L2202 (E) Illegal cpu information file : "文件"
"文件"不是单片机信息文件。

L2203 (E) Illegal profile information file : "文件"
"文件"不是配置信息文件。

L2210 (E) Invalid input file type specified for option "选项" : "文件(种类)"
在指定 "选项"时输入了不能处理的 "文件 (种类)"。

L2211 (E) Invalid input file type specified while processing "进程" : "文件(种类)"
在 "进程"处理中输入了不能处理的 "文件 (种类)"。

L2212 (E) "选项" cannot be specified for inter-module optimization information in "文件"
因为 "文件"中有模块之间的优化信息，所以不能使用 "选项"的选项。不能在编译和汇编时使用 goptimize 选项。

L2220 (E) Illegal mode type "模式类型" in "文件"
输入了不同 "模式类型"的文件。

L2221 (E) Section type mismatch : "段"
输入了不同属性 (有无初始值)的同名段。

L2300 (E) Duplicate symbol "符号" in "文件"
"符号"有重复。

L2301 (E) Duplicate module "模块" in "文件"
"模块"有重复。

L2310 (E) Undefined external symbol "符号" referenced in "文件"
参照了 "文件"中未定义的 "符号"。

L2311 (E) Section "段1" cannot refer to overlaid section : "段2"-"符号"
在指定相同地址的覆盖段之间有符号参照。
请不要将 "段1"和 "段2"分配到相同的地址。

L2320 (E) Section address overflowed out of range : "段"
"段"的地址超出了能使用的地址范围。

L2321 (E) Section "段1" overlaps section "段2"
"段1"和 "段2"的地址有重叠。请更改 start 选项指定的地址。

L2322 (E) Section size too large: "段"
"段"的段太大。
$TBR 段的大小不能超过 1024 字节。

L2323 (E) Section "段 1(地址范围)" overlaps with section "段 2(地址范围)" in physical space

在物理存储器的分配上，“段 1”和 “段 2”有重叠。
请检查各段的分配地址。
< 地址范围 > : < 段的起始地址 >-< 段的结束地址 >

L2330 (E) Relocation size overflow : " 文件 "-" 段 "-" 偏移量 "

再定位运算结果超过了再定位长度。可能是因为不能到达转移目标或者参照了必须分配到特定地址的符号。请确认编译列表或者汇编列表中 “段”的 “偏移量”位置的参照符号是否分配到正确的位置。

L2331 (E) Division by zero in relocation value calculation : " 文件 "-" 段 "-" 偏移量 "

在再定位运算中发生了被 0 除的运算。请确认编译列表或者汇编列表中 “段”的 “偏移量”位置的运算是否有问题。

L2332 (E) Relocation value is odd number : " 文件 "-" 段 "-" 偏移量 "

再定位运算结果为奇数。请确认编译列表或者汇编列表中 “段”的 “偏移量”位置的运算是否有问题。

L2340 (E) Symbol name in section " 段 " is too long

fsymbol 指定的 “段”中的符号字符数超过了 8174 字符。

L2400 (E) Global register in " 文件 " conflicts : " 符号 "," 寄存器 "

“文件”中指定的全局寄存器已分配了其他符号。

L2401 (E) near8,near16 symbol " 符号 " is outside near memory area

没有将 “符号”分配到 near8 或者 near16 的范围。为了能正确地计算地址，请更改 start 的指定或者解除编译时的 near 指定。

L2402 (E) Number of register parameter conflicts with that in another file : " 函数 "

在多个文件中给 “函数”指定了不同的寄存器参数的个数。

L2403 (E) Fast interrupt register in " 文件 " conflicts with that in another file

“文件”中指定的高速中断通用寄存器的序号和其他文件不统一。请对照其他文件的高速中断通用寄存器的序号重新进行编译。

L2404 (E) Base register " 基址寄存器的种类 " in " 文件 " conflicts with that in another file

“文件”中指定的 “基址寄存器的种类”的寄存器序号和其他文件不统一。请对照其他文件的基址寄存器的序号重新进行编译。

L2410 (E) Address value specified by map file differs from one after linkage as to "符号"

作为 “符号” 的地址值，编译时使用的外部符号分配信息文件中的地址和连接后的地址不同。请确认下述 (1) ～ (3) 的内容：

(1) 不能在指定 map 选项 （编译时）的前后更改程序。

(2) 指定 map 选项 （编译时）前后的符号排列顺序有可能因 optlnk 的优化而发生变化。请将编译时的 map 选项置为无效或者将 optlnk 的优化选项置为无效。

(3) 在使用 tbr 选项或者 #pragma tbr 时，指定 map 选项 （编译时）后的符号有可能因编译程序的优化而被删除。请将编译时的 map 选项置为无效，或者将 tbr 选项或者 #pragma tbr 置为无效。

L2411 (E) Map file in "文件" conflicts with that in another file

在编译时，在输入文件之间使用了不同的外部符号分配信息文件。

L2412 (E) Cannot open file : "文件"

不能打开 “文件”（外部符号分配信息文件）。请确认文件名和存取权是否正确。

L2413 (E) Cannot close file : "文件"

不能关闭 “文件”（外部符号分配信息文件）。磁盘容量可能没有空间。

L2414 (E) Cannot read file : "文件"

不能读 “文件”（外部符号分配信息文件）。磁盘容量可能没有空间。

L2415 (E) Illegal map file : "文件"

这是非法的 “文件”（外部符号分配信息文件）格式。请确认文件名是否正确。

L2416 (E) Order of functions specified by map file differs from one after linkage as to "函数名"

在编译时使用的外部符号分配信息文件中的信息和连接后的分配信息中，函数 “函数名” 和其他函数的排列顺序不同。函数中 static 变量的地址有可能和外部符号分配信息文件以及连接后的结果不同。

L2417 (E) Map file is not the newest version: "文件名"

.map 文件不是最新的版本。

L2420 (E) "文件 1" overlap address "文件 2" : "地址"

文件 1 和文件 2 的地址有重叠。

P2500 (E) Cannot find library file : "文件"

没有指定的库 “文件”。

P2501 (E) "示例" has been referenced as both an explicit specialization and a generated instantiation

对于已定义的示例请求生成示例。

对于正在使用 “示例” 的文件，请确认是否通过 form=relocate 建立了可再定位文件。

P2502 (E) "示例" assigned to "文件 1" and "文件 2"
在 “文件 1” 和 “文件 2” 中重复定义了 “示例”。
对于正在使用 “示例” 的文件，请确认是否通过 form=relocate 建立了可再定位文件。

L3000 (F) No input file
没有输入文件。

L3001 (F) No module in library
库中的模块数为 0。

L3002 (F) Option "选项 1" is ineffective without option "选项 2"
“选项 1” 需要 “选项 2”。

L3004 (F) Unsupported inter-module optimization information type "类型" in "文件"
在文件中有不支持的模块之间优化信息 “类型”。请确认编译程序和汇编程序的版本是否正确。

L3100 (F) Section address overflow out of range : "段"
“段” 的地址超出了能使用的上限区域。
请更改 start 选项指定的地址。
有关地址空间的详细内容，请参照各单片机的硬件手册。

L3102 (F) Section contents overlap in absolute section "段"
绝对地址段的段内数据地址有重叠。请修正源程序。

L3110 (F) Illegal cpu type "单片机类型" in "文件"
输入了不同单片机类型的文件。

L3111 (F) Illegal encode type "字节序类型" in "文件"
输入了不同字节序类型的文件。

L3112 (F) Invalid relocation type in "文件"
“文件” 中有不支持的再定位类型。请确认编译程序和汇编程序的版本是否正确。

L3120 (F) Illegal size of the absolute code section : "段" in "文件"
“文件” 中存在非法的的绝对地址代码段 “段” 的大小。在 CPU 类型为 RX 族并且是大端法的情况下，请将绝对地址代码段的大小更改为 4 的倍数。

L3200 (F) Too many sections
段的个数超过了翻译限制。如果指定多个文件的输出，就可能解此决问题。

L3201 (F) Too many symbols
符号个数超过了翻译限制。如果指定多个文件的输出，就可能解此决问题。

L3202 (F) Too many modules
模块的个数超过了翻译限制。请分库建立。

L3203 (F) Reserved module name "optlnk_generates"
optlnk_generates_** （** 是 01 ～ 99 的数值）是优化连接编辑程序使用的保留名称，用作 .obj/.rel 文件名和库中的模块名。请在用于文件名和库中的模块名时进行更改。

L3300* (F) Cannot open file : " 文件 "
不能打开 “文件”。请确认文件名和存取权是否正确。
* 有可能显示 P3300。

L3301 (F) Cannot close file : " 文件 "
不能关闭 “文件”。磁盘容量可能没有空间。

L3302 (F) Cannot write file : " 文件 "
不能写 “文件”。磁盘容量可能没有空间。

L3303* (F) Cannot read file : " 文件 "
不能读 “文件”。有可能输入了空文件或者磁盘容量没有空间。
* 有可能显示 P3303。

L3310* (F) Cannot open temporary file
不能打开中间文件。请确认 HLNK_TMP 指定是否正确。
磁盘容量可能没有空间。
* 有可能显示 P3310。

L3311 (F) Cannot close temporary file
不能关闭中间文件。磁盘容量可能没有空间。

L3312 (F) Cannot write temporary file
不能写中间文件。磁盘容量可能没有空间。

L3313 (F) Cannot read temporary file
不能读中间文件。磁盘容量可能没有空间。

L3314 (F) Cannot delete temporary file
不能删除中间文件。磁盘容量可能没有空间。

L3320* (F) Memory overflow
优化连接编辑程序内部使用的存储器容量不够。请增加存储器容量。
* 有可能显示 P3320。

L3400 (F) Cannot execute " 加载模块 "
不能启动 “装入模块”。请确认是否设定了 “装入模块” 的路径。

L3410 (F) Interrupt by user
从标准输入终端检测到 “(Ctrl)+C” 键中断。

L3420 (F) Error occurred in " 加载模块 "
在执行 “装入模块” 的过程中发生了错误。

P3500 (F) Bad instantiation request file -- instantiation assigned to more than one file
示例的生成指定文件有误。
请重新对连接对象文件进行编译。

P3501 (F) Instantiation loop
示例生成处理进行了循环处理。
输入的文件名有可能和其他文件的示例生成要求文件相同。为了使文件名（扩展名除外）不相同，请更改文件名。

P3502 (F) Cannot create instantiation request file " 文件 "
不能建立示例的生成指定文件。
请确认目标建立文件夹的存取权是否正确。

P3503 (F) Cannot change to directory " 文件夹 "
不能移动到 “文件夹”。请确认 “文件夹” 是否存在。

P3504 (F) File " 文件 " is read-only
“文件” 是只读文件。请更改存取权。

L4000* (-) Internal error : (" 内部错误号 ") " 文件 行号 " / " 注释 "
在优化连接编辑程序的处理过程中发生了内部错误。
请与销售部门联系，告知信息中的内部错误号、文件、行号和注释的内容。
* 有可能显示 P4000。

14. 翻译限制

14.1 编译程序的翻译限制

编译程序的翻译限制如**表 14.1** 所示。
请在此翻译限制范围内建立源程序。

表 14.1 编译程序的翻译限制

	分类	项目	编译限制
1	启动	define 选项能指定的宏名总数	无限制 （取决于存储器容量）
2		文件名的字符数	无制限 （取决于 OS）
3	源程序	1 行的字符数	32768 个字符
4		每个文件的源程序的行数	无限制 （取决于存储器容量）
5		能编译的源程序的总行数	无限制 （取决于存储器容量）
6	预处理器	#include 语句的嵌套深度	无限制 （取决于存储器容量）
7		#define 语句的宏名总数	无限制 （取决于存储器容量）
8		宏定义和宏调用的参数的个数	无限制 （取决于存储器容量）
9		宏名的重新替换数	无限制 （取决于存储器容量）
10		条件编译的嵌套层数	无限制 （取决于存储器容量）
11		#if 语句和 #elif 语句中能指定的运算符和非运算符的合计数	无限制 （取决于存储器容量）
12	声明	函数定义的个数	无限制 （取决于存储器容量）
13		外部连接标识符 （外部名）的个数	无限制 （取决于存储器容量）
14		1 个函数中有效标识符 （内部名）的个数	无限制 （取决于存储器容量）
15		限定基本型的指针、数组和函数声明的个数	16 个
16		数组的维数	6 维
17		数组 / 结构体的大小	2147483647 字节
18	语句	复合语句的嵌套深度	无限制 （取决于存储器容量）
19		根据迭代语句 （while 语句、 do 语句、 for 语句）和选择语句 （if 语句、 switch 语句）的组合产生的嵌套深度	4096 层
20		1 个函数中能记述的复合语句的个数	2048 个
21		1 个函数中能指定的 goto 标号的个数	2147483646 个
22		switch 语句的个数	2048 个
23		switch 语句的嵌套深度	2048 层
24		1 个 switch 语句中能指定的 case 标号的个数	2147483646 个
25		for 语句的嵌套深度	2048 层
26	表达式	字符串的字符数	32766 字符
27		函数定义和函数调用的参数的个数	2147483646 个
28		1 个表达式中能指定的运算符和非运算符的合计数	约 500 个
29	标准库	open 函数能同时打开的文件个数	可变 *1
30	段	段名的长度 *2	8146 字符
31		1 个文件中 #pragma section 能指定的段的个数	2045 个

【注】 *1 详细内容请参照 “8.3.2 初始设定”。

*2 因为在生成目标时，受汇编程序的 1 行字符数的限制，所以 #pragma section 和 section 选项能指定的长度小于段名的长度。

14.2 汇编程序的翻译限制

汇编程序的翻译限制如表 14.2 所示。

表 14.2 汇编程序的翻译限制

	项目	翻译限制
1	1 行的字符数	8190 个字符
2	符号长度	1 行的字符数 *1
3	符号个数	无限制 （取决于存储器容量）
4	外部参照符号的个数	无限制 （取决于存储器容量）
5	外部定义符号的个数	无限制 （取决于存储器容量）
6	段的最大长度	0FFFFFFFFH 字节
7	段的个数	65265 个 （有调试信息）、 65274 个 （无调试信息）
8	文件 include	嵌套 30 层
9	字符串的长度	1 行的字符数 *1
10	文件名的字符数	1 行的字符数 *1
11	环境变量的设定字符数	2048 字节
12	宏定义的个数	65535 个

【注】 *1 根据同行中指定的字符串的长度，限制值小于此值。

15. 建立程序时的注意事项

对于编译程序，本章节说明编程时的注意事项以及从编译到调试的程序开发时的注意事项。

15.1 编程时的注意事项

(1) 函数原型声明

在调用函数时，必须对被调用的函数进行函数原型声明。在不进行函数原型声明时，有可能无法正常地接受和传递参数。

例 1
有 float 型参数的函数 （指定 dbl_size=8 的情况）

```
void g()
{
          float a;
          ...
          f(a);                                  // 将 a 转换为 double 型。
}
void f(float x)
{...}
```

例 2
有堆栈传递的 signed char、 (unsigned)char、 (signed)short 和 unsigned short 型参数的函数

```
void h();
void g()
{
          char a,b;
          ...
          h(1,2,3,4,a,b);                        // 将 a、 b 转换为 int 型。
}
void h(int a1, int a2, int a3, int a4, char a5, char a6)
{...}
```

(2) 参数中没有型信息的函数声明

在对相同的函数进行多次函数声明（包括函数定义）时，不能在参数排列中同时使用不记述型的格式和记述型的格式。否则，因为在调用函数和被调用函数中参数的解释不同，所以生成的代码有可能不能对型进行正确的处理。

如果在编译时显示 C5147 的错误信息，就可能遇到此问题，因此必须更改参数排列中记述型的格式，或者检查生成的代码来确认参数的接受和传递是否有问题。

例

因为用不同的格式记述了 old_style，在调用函数和被调用函数中参数 d 和 e 的型的含义不同，所以不能正确地接受和传递参数。

```
extern int old_style(int,int,int,short,short); /* 函数声明：参数排列中记述型的格式 */
int old_style(a,b,c,d,e)                        /* 函数定义：参数排列中不记述型的格式 */
 int a,b,c;
 short d,e;
{
    return a + b + c + d + e;
}
int result;
func()
{
    result = old_style(1,2,3,4,5);
}
```

(3) C/C++ 语言规格中未规定评价顺序的表达式

在使用 C/C++ 语言规格中未规定评价顺序的表达式时，如果编制的程序因评价顺序而产生不同的结果，就无法保证运行。

例

```
a[i]=a[++i]   ; 根据是先评价还是后评价赋值表达式的右边，左边值发生变化。

sub(++i, i)   ; 根据是先评价还是后评价函数的第 1 个参数，第 2 个参数的值发生变化。
```

(4) 上溢运算和被零除

即使发生上溢运算和浮点的被零除，也不输出错误信息。如果在 1 个常数或者常数之间的运算中发生上溢，就在编译时输出错误信息。

例

```
void main()
{
  int ia;
  int ib;
  float fa;
  float fb;

  ib=32767;
  fb=3.4e+38f;

  /* 在进行常数或者常数之间的运算时,                    */
  /* 如果检测到上溢，就输出编译错误信息。               */

  ia=99999999999;      /* (W) 检测到常数的上溢。        */
  fa=3.5e+40f;         /* (W) 检测到浮点运算的上溢。    */

  /* 如果在执行时发生上溢，就不输出错误信息。           */

  ib=ib+32767;         /* 忽视运算结果的上溢。          */
  fb=fb+3.4e+38f;      /* 忽视浮点运算结果的上溢。      */
}
```

(5) 写 const 型变量

需要注意：即使声明了 const 型的变量，当通过型转换将其转换为非 const 型后赋值或者在分割编译的程序之间不进行型统一的处理时，也无法通过编译程序检查 const 型变量的写操作。

例

```
const char *p;        /* 因为库函数 strcat 的第 1 个参数              */
:                     /* 是指向 char 型的指针型，所以                 */
strcat(p, "abc");     /* 有可能改写参数指向的区域。                   */
```

文件 1

```
const int i;
```

文件 2

```
extern int i;         /* 因为在文件 2 中没有用 const 型声明变量 i，   */
:                     /* 所以即使在文件 2 中进行写变量 i，            */
i=10;                 /* 也不会出错。                                 */
```

(6) 数学函数库的精度

需要注意：当 x ≈ 1 时， acos(x) 函数和 asin(x) 函数的误差变大。

误差范围如下：

acos(1.0 – ε) 的绝对误差	双精度	2^{-39} （$\varepsilon = 2^{-33}$）
	单精度	2^{-21} （$\varepsilon = 2^{-19}$）
asin(1.0 – ε) 的绝对误差	双精度	2^{-39} （$\varepsilon = 2^{-28}$）
	单精度	2^{-21} （$\varepsilon = 2^{-16}$）

(7) 有可能因优化而被删除的编程

如果记述了连续相同变量的参照或者不使用结果的表达式时，有可能因编译程序的优化而作为冗余码被删除。如果想随时保证存取，就必须在声明时指定 volatile。

例：

```
[1] b=a;          /* 第 1 行的表达式有可能作为冗余码被删除。          */
    b=a;
[2] while(1)a;    /* 变量 a 的参照和循环语句有可能作为冗余码被删除。  */
```

(8) C89 和 C99 的运行差异

在 C99 中，用 {} 将选择语句和重复语句括起来。因此在 C89 和 C99 的运行有可能不同。

例：

```
enum {a,b};
int g(void)
{
      if(sizeof(enum{b,a}))
          return a;
      return b;
}
```

如果指定 -lang=c99 对上述内容进行编译，就进行以下解释：

```
enum {a,b};
int g(void)
{
      if(sizeof(enum{b,a}))
      {
          return a;
      }
      return b;
}
```

当 -lang=c 时， g()=0 ；当 -lang=c99 时， g()=1。

15.2 通过 C++ 编译程序对 C 程序进行编译时的注意事项

(1) 函数原型声明

在使用函数前需要进行函数原型声明，此时还必须声明形式参数的型。

```
extern void func1();
void g()
{
    func1(1); // 错误
}
```

```
extern void func1(int);
void g()
{
    func1(1); // OK
}
```

(2) const 目标的连接

const 目标的连接在 C 程序中为外部连接，而在 C++ 程序中为内部连接。另外， const 目标需要初始值。

```
const cvalue1;       // 错误

const cvalue2 = 1; // 内部连接
```

```
const cvalue1=0;
   // 设定初始值。

extern const cvalue2 = 1;
   // 和C程序一样，为外部连接。
```

(3) void* 的赋值

在 C++ 程序中，如果不使用明确的型转换，就不能给其他指向目标型的指针 （指向函数的指针和指向成员指针除外）赋值。

```
void func(void *ptrv, int *ptri)
{
    ptri = ptrv; //错误
}
```

```
void func(void *ptrv, int *ptri)
{
    ptri = (int *)ptrv; //OK
}
```

15.3 有关选项的注意事项

(1) 需要指定统一的选项

需要指定统一的选项如下 (a)、 (b) 所示。如果连接这些选项指定的不同可再定位文件和库文件，就无法保证执行时的运行。

(a) 必须在编译程序、汇编程序和库编辑程序中，统一 cpu、 endian、 base 和 fint_register 这 4 个选项。

(b) 对于 “2.5 单片机选项”中的 (a) 以外的选项，必须在编译程序和库编辑程序中统一这些选项。

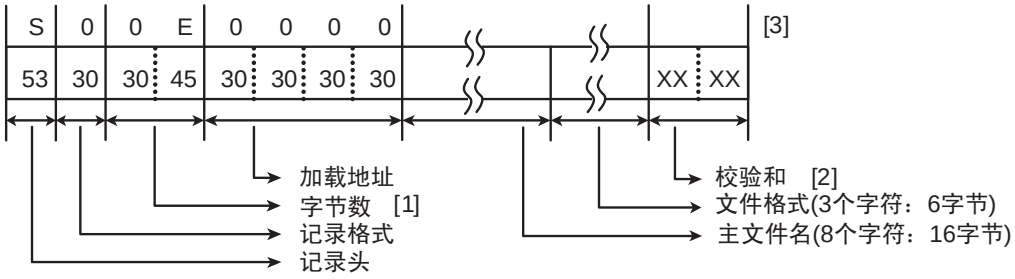
16. 附录

16.1 Motorola S 格式和 Intel HEX 格式的文件

本章节说明通过优化连接编辑程序输出的 Motorola S 格式和 Intel HEX 格式的文件。

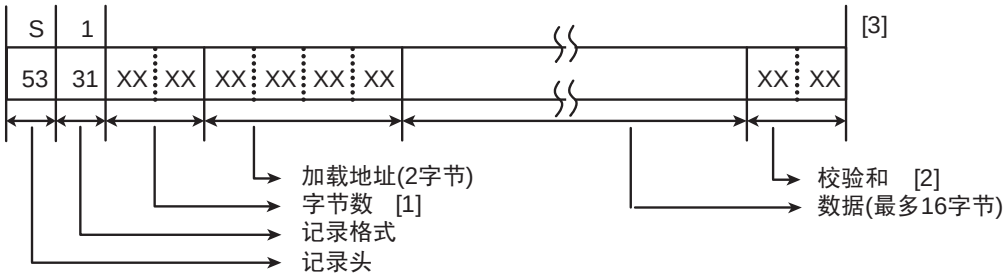
16.1.1 Motorola S 格式的文件

(a) 头记录(S0记录)

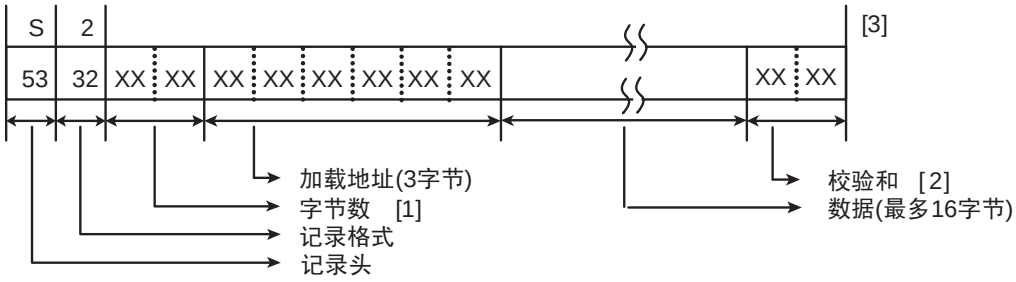


(b) 数据记录(S1、S2、S3记录)

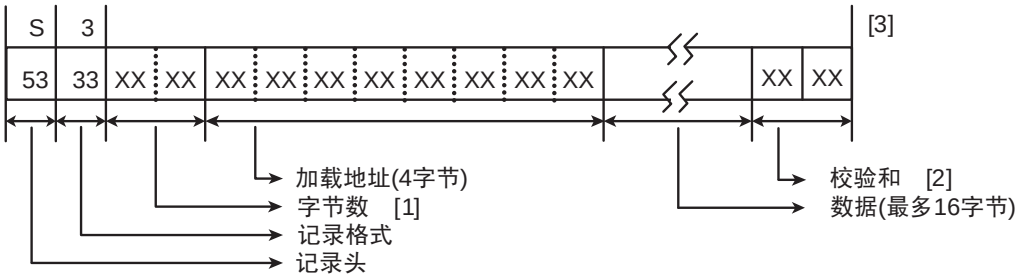
(i) 加载地址为0～FFFF的情况



(ii) 加载地址为10000～FFFFFF的情况

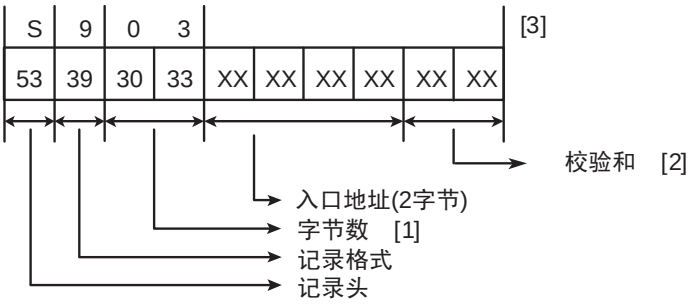


(iii) 加载地址为1000000～FFFFFFFF的情况

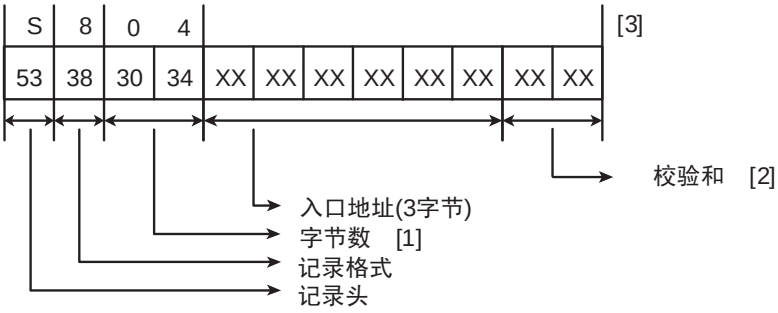


(c) 结束记录(S9、S8、S7记录)

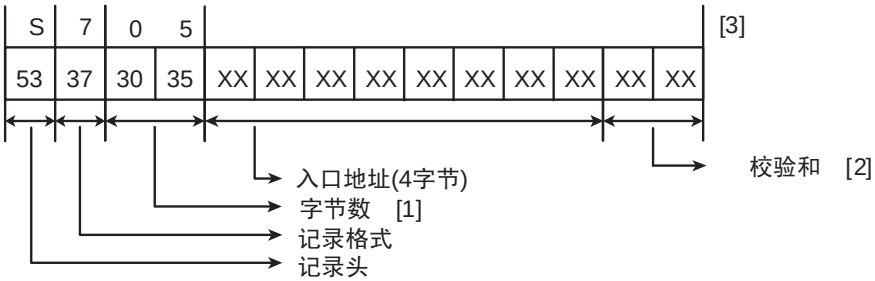
(i) 入口地址为0～FFFF的情况



(ii) 入口地址为10000～FFFFFF的情况



(iii) 入口地址为1000000～FFFFFFFF的情况



【注】 [1] 从加载地址(或者入口地址)到校验和的字节数

[2] 从字节数到校验和前的数据值的总和(以字节为单位)的1的补数

[3] 在校验和后附加换码。

16.1.2 Intel HEX 格式的文件

各数据记录的执行地址按以下方法计算：

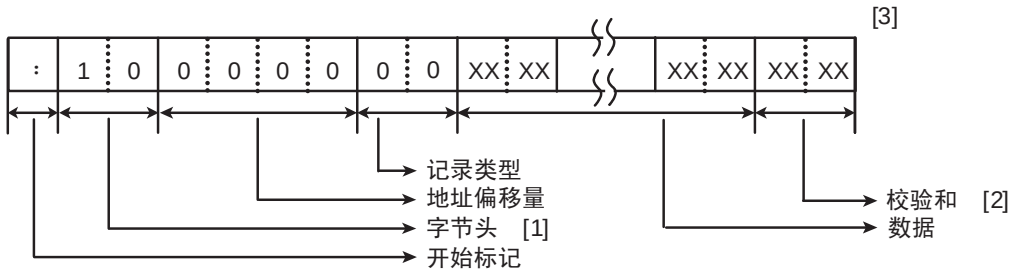
(1) 段地址

(段基址 <<4) + (数据记录的地址偏移量)

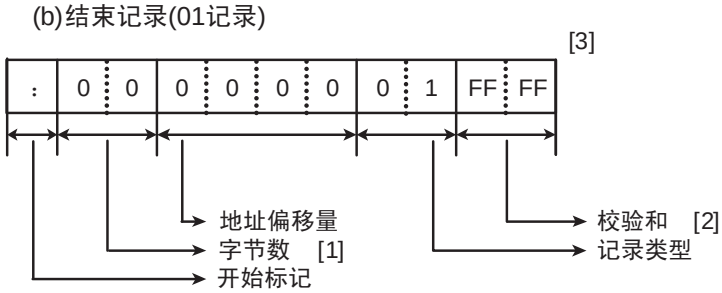
(2) 线性地址

(线性基址 <<16) + (数据记录的地址偏移量)

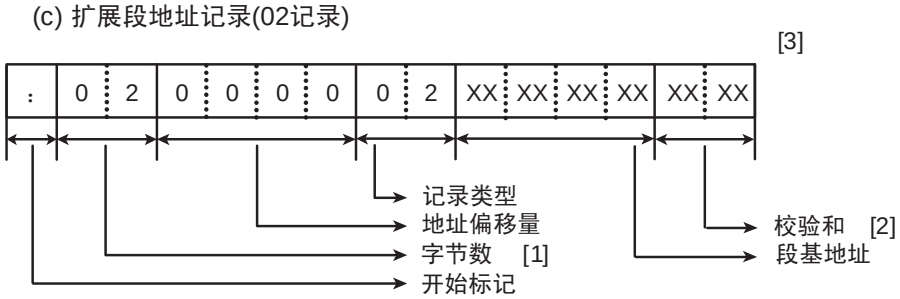
(a) 数据记录(00记录)



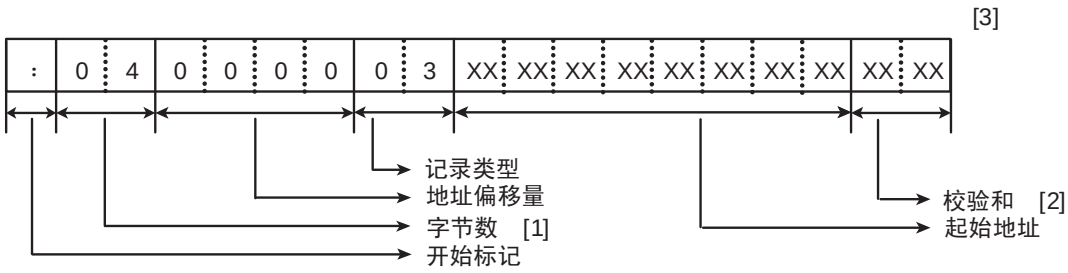
(b)结束记录(01记录)



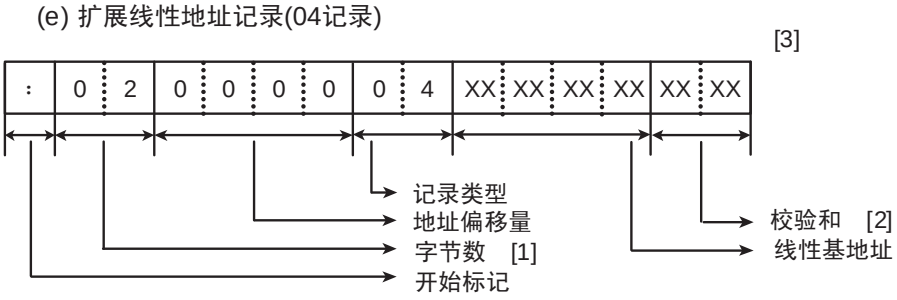
(c) 扩展段地址记录(02记录)



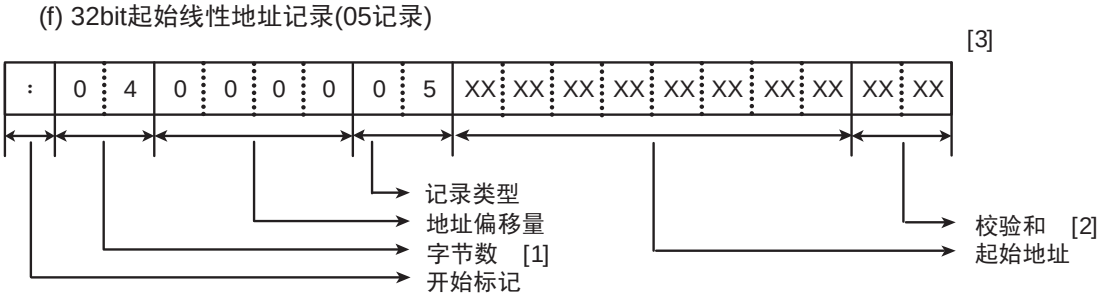
(d) 起始地址记录(03记录)



(e) 扩展线性地址记录(04记录)



(f) 32bit起始线性地址记录(05记录)



【注】 [1] 从记录类型的下一个数据到校验和的字节数

[2] 从字节数到校验和前的数据值的总和(用16进制数)的2的补数(低位8bit有效)

[3] 在校验和之后附加换行码。

16.2 ASCII 码一览表

表 16.1 ASCII 码一览表

低 4 位	高 4 位							
	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	“	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	‘	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

修订记录	RX 族 C/C++ 编译程序、汇编程序、优化连接编辑程序 编译程序包 用户手册

Rev.	发行日	修订内容	
		页	修订处
1.00	2011.03.30	—	初版发行

RX 族 C/C++ 编译程序、汇编程序、优化连接编辑程序
编译程序包 用户手册

Publication Date: Rev.1.00 Mar 30, 2011

Published by: Renesas Electronics Corporation

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
7F, No. 363 Fu Shing North Road Taipei, Taiwan, R.O.C.
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 1.0

RX族C/C++编译程序、汇编程序、优化连接编辑程序

瑞萨电子株式会社

R20UT0528CJ0100