

To our customers,

---

## Old Company Name in Catalogs and Other Documents

---

On April 1<sup>st</sup>, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: <http://www.renesas.com>

April 1<sup>st</sup>, 2010  
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (<http://www.renesas.com>)

Send any inquiries to <http://www.renesas.com/inquiry>.

## 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.

***USER'S MANUAL***

**RENESAS**

**Phase-out/Discontinued**

# CC78K SERIES C COMPILER

**OPERATION**

***USER'S MANUAL***

**NEC**

**Phase-out/Discontinued**

# **CC78K SERIES C COMPILER**

**OPERATION**



No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Corporation. NEC Corporation assumes no responsibility for any errors which may appear in this document.

NEC Corporation does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from use of a device described herein or any other liability arising from use of such device. No license, either express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Corporation or of others.



## INTRODUCTION

The CC78K Series C Compiler User's Manual for Operation (hereinafter referred to as this manual) has been prepared to give those who develop software by using this C compiler a correct understanding of the functions and operating methods of this C compiler.

This manual does not cover how to describe the source programs of the CC78K series C compiler. Therefore, before reading this manual, you should read the CC78K Series C Compiler User's Manual for Language published as a separate volume.



## [Target Devices]

Software for the following microcomputers can be developed with this C compiler.

Series	Target device
78K/0	uPD78001, uPD78002, uPD78011, uPD78012, uPD78013, uPD78014, uPD78P014, uPD78022, uPD78023, uPD78024, uPD78P024, uPD78042, uPD78043, uPD78044, uPD78P044
78K/II	uPD78210*, uPD78212, uPD78213, uPD78214, uPD78P214, uPD78217A, uPD78218A, uPD78P218A, uPD78220, uPD78224, uPD78P224, uPD78233, uPD78234, uPD78237, uPD78238, uPD78P238, uPD78243, uPD78244
78K/III	uPD78310*, uPD78312*, uPD78P312*, uPD78310A, uPD78312A, uPD78P312A, uPD78320, uPD78322, uPD78P322, uPD78323, uPD78324, uPD78P324, uPD78327, uPD78328, uPD78P328, uPD78330, uPD78334, uPD78P334, uPD78350, uPD78P352

\* Products for maintenance

Note: Of the above listed products, some are under development.

## [Readers of Manual]

Although this manual is intended for those who have read the user's manual of the microcomputer subject to software development and have experience in software programming, the readers need not necessarily have a knowledge of C compilers or C language. Therefore, this manual can also be read by those who use a C compiler for the first time.

However, because this C compiler consists of only the program files related to translation (compilation), the RA78K series assembler package is also required to develop programs by using this C compiler.

## [Organization of Manual]

This manual consists of the following nine chapters and appendixes:

### Chapter 1 - General

Outlines the functions of this C compiler including its role and standing in microcomputer development. Also introduces the features of the C compiler.

### Chapter 2 - Product Overview

Describes the program filenames offered by this C compiler package and the operating environment of each program.

### Chapter 3 - Execution of C Compiler

Explains the procedures required for executing this compiler using sample programs.

### Chapter 4 - I/O Files and Start-up of C Compiler

Details the I/O files of this C compiler and how to start up the C compiler.

### Chapter 5 - Compiler Options

Details how to specify the various options of this C compiler and precedence of options, together with explanation of an application example for each compiler option.

### Chapter 6 - Output Lists of C Compiler

Explains the formats of various lists to be output by this C compiler.

### Chapter 7 - C Compiler Utilization

Introduces some measures for effective utilization of this C compiler.

### Chapter 8 - Start-up routines and error handling routines

Explains the contents of the respective start-up and error handling routines and their usages using sample programs, and some important points for improving the sample programs.

### Chapter 9 - Error Messages

Describes error messages to be output by this C compiler.

## Appendixes

Contain examples of sample program lists, hints on use and restrictions, and a list of compiler options.

### [Recommended Usage of Manual]

For those who wish to actually operate this C compiler: First, read Chapter 3, Execution of C Compiler.

For those who have a general understanding of C compilers: You may skip Chapter 1, General.

Utilize various lists in the respective appendixes after you have familiarized yourself with the operation of the C compiler. Chapter 5, Compiler Options and Chapter 9, Error Messages also contain information for quick reference.

### [Symbols and Abbreviations]

The following symbols and abbreviations are used in this manual:

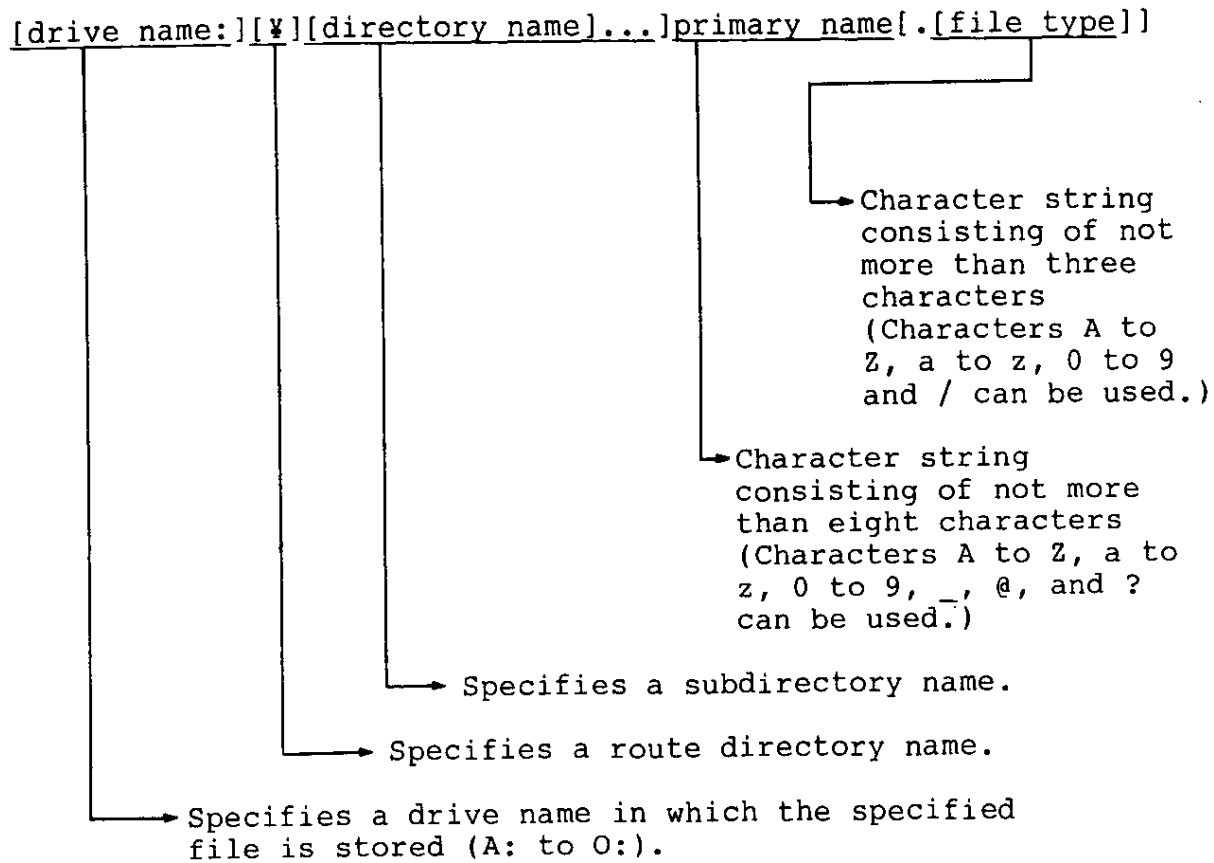
<u>Symbol</u>	<u>Meaning</u>
...	Continuation (repetition) of data in the same format
[ ]	Parameters in square brackets may be omitted.
" "	Characters enclosed in " " (double quotes) must be input as is.
' '	Characters enclosed in ' ' (single quotes) must be input as is.
( )	Characters enclosed in parentheses must be input as is.
_____	Important point or character string that must be input by the user in the input example
△	One or more spaces must be input.
⋮	This part of the program description is omitted.
/	Delimiter
Ⓕ	Return key input
\	Backslash

With PC-DOS, a backslash (\) is used in place of the ¥ sign.

# [Conventions of Filename Specification]

Conventions of input filename specification in the start-up command line of this C compiler are as follows:

## (1) Disk type file specification



Example: A:¥sample¥prime.c

- Note:
1. No blank (space) can be specified before or after ":" (semicolon), "." (period), and "¥".
  2. Uppercase and lowercase letters are not distinguished from each other.
  3. With PC-DOS, a backslash (\) is used in place of the ¥ sign.

## (2) Device type file specification

One of the following four logical devices can be specified:

Logical device	Description
CON	Outputs to Console
PRN	Outputs to Printer
AUX	Outputs to Auxiliary output device
NUL	Dummy output (Nothing will be output)

IBM-PC<sup>TM</sup>, IBM-PC/XT<sup>TM</sup>, IBM-PC/AT<sup>TM</sup>, and PC-DOS<sup>TM</sup> are trademarks of IBM Corp.

V30<sup>TM</sup> is a trademark of NEC Corporation.

80386<sup>TM</sup>, 80286<sup>TM</sup>, and 8086<sup>TM</sup> are trademarks of Intel Corp.

MS-DOS<sup>TM</sup> is a trademark of Microsoft Corp.

## CONTENTS

	<u>Page</u>
CHAPTER 1. GENERAL .....	1-1
1.1 What Is a C Compiler ? .....	1-1
1.1.1 C language and Assembly language .....	1-1
1.1.2 Development of microcomputer-applied products and role of this product .....	1-3
1.2 Program Development Procedure by C Compiler .....	1-5
1.2.1 Creating a source module file with the editor .....	1-6
1.2.2 C compiler .....	1-7
1.2.3 Assembler .....	1-8
1.2.4 Linker .....	1-9
1.2.5 Object converter .....	1-10
1.2.6 Librarian .....	1-11
1.2.7 Screen debugger .....	1-12
1.3 Reminders Before Program Development .....	1-13
1.4 Features of This C Compiler .....	1-15
 CHAPTER 2. PRODUCT OVERVIEW .....	2-1
2.1 Contents of Floppy Disks .....	2-1
2.1.1 System files .....	2-2
2.1.2 Library files .....	2-4
2.2 Forms of File Media Supplied .....	2-5
2.3 System Configuration .....	2-5
 CHAPTER 3. EXECUTION OF C COMPILER .....	3-1
3.1 Before Executing the C Compiler .....	3-1
3.1.1 Confirming the contents of the supplied disk ...	3-1
3.1.2 Sample program .....	3-2
3.2 Procedure for C Compiler Execution .....	3-3

	<u>Page</u>
CHAPTER 4. C COMPILER .....	4-1
4.1 Input/Output Files of This C Compiler .....	4-1
4.2 How to Start Up the C Compiler .....	4-4
4.2.1 Starting up the C compiler .....	4-4
4.2.2 Execution start and end messages .....	4-6
4.3 C Compiler Options .....	4-8
4.4 Optimization .....	4-9
4.5 ROMable Processing of Programs .....	4-12
4.5.1 At compile time .....	4-12
4.5.2 At linkage time .....	4-12
4.6 Error Check at Execution Time .....	4-14
4.6.1 Error handling routine .....	4-14
4.6.2 Error check library names .....	4-15
 CHAPTER 5. COMPILER OPTIONS .....	 5-1
5.1 Types of Compiler Options .....	5-1
5.2 How to Specify Compiler Options .....	5-4
5.3 Priority of Compiler Options .....	5-5
5.4 Description of Each Compiler Option .....	5-7
(1) Processor type specification (-C) .....	5-8
(2) Object module file creation specification (-O/-NO) .....	5-12
(3) Symbol name length specification (-S/-NS) .....	5-14
(4) Symbol name upper-/lower-case specification (-CA/-NCA) .....	5-17
(5) ROMable object file creation specification (-R/-NR) .....	5-20
(6) Optimization process specification (-Q/-NQ) .....	5-24
(7) Debug information output specification (-G/-NG) .....	5-27
(8) Execution-time error check specification (-L/-NL) .....	5-29
(9) Preprocess list file creation specification (-P/-NP, -K/-NK) .....	5-32
(10) Preprocessing specification (-D/-ND, -U/-NU, -I) .....	5-38

(11) Assembler source module file creation specification (-A/-NA, -SA/-NSA) .....	5-44
(12) Error list file creation specification (-E/-NE, -SE/-NSE) .....	5-50
(13) Cross-reference list file creation specification (-X/-NX) .....	5-55
(14) List file format specification (-LW, -LL, -LT, and -LF) .....	5-57
(15) Warning output specification (-W) .....	5-67
(16) Execution status display specification (-V/-NV) .....	5-69
(17) Parameter file specification (-F) .....	5-71
(18) Temporary file creation directory specification (-T) .....	5-73
(19) HELP message output specification (--) .....	5-75
 CHAPTER 6. OUTPUT FILES OF C COMPILER .....	6-1
6.1 Object Module File .....	6-1
6.2 Assembler Source Module File .....	6-2
6.3 Error List File .....	6-5
6.3.1 Error list file with C source .....	6-5
6.3.2 Error list file containing error messages only .....	6-8
6.4 Preprocess List File .....	6-10
6.5 Cross-reference List File .....	6-13
 CHAPTER 7. EFFECTIVE UTILIZATION OF C COMPILER .....	7-1
7.1 EXIT Status Function for Efficient Compilation .....	7-1
7.2 Environment Variable for Development Environment Setting .....	7-2
7.3 Interruption of Compiler Operation .....	7-3



	<u>Page</u>
CHAPTER 8. START-UP ROUTINES AND ERROR HANDLING ROUTINES .	8-1
8.1 General .....	8-1
8.2 File Organization .....	8-2
8.3 Description of Each Batch File .....	8-4
8.3.1 Batch file for creating a start-up routine .....	8-4
8.3.2 Batch file for updating error handling routine libraries .....	8-6
8.4 Start-up Routines .....	8-8
8.4.1 Outline of start-up routine .....	8-8
8.4.2 Description of sample program .....	8-11
8.4.3 Points for improvement .....	8-25
8.5 Error Handling Routines .....	8-28
8.5.1 Outline of error handling routine .....	8-28
8.5.2 Description of sample program .....	8-30
8.5.3 Point for improvement .....	8-33
 CHAPTER 9. ERROR MESSAGES .....	9-1
9.1 Types of Error Messages .....	9-1
9.2 List of Error Messages .....	9-1
 APPENDIX A. SAMPLE PROGRAMS .....	A-1
APPENDIX B. LIST OF HINTS ON USE .....	B-1
APPENDIX C. LIST OF COMPILER OPTIONS .....	C-1

## ILLUSTRATIONS AND TABLES

	<u>Page</u>
Fig. 1-1. Flow of Translation .....	1-2
Fig. 1-2. Development Process of Microcomputer-applied Product .....	1-3
Fig. 1-3. Software Development Process .....	1-4
Fig. 1-4. Program Development Procedure by This C Compiler .....	1-5
Fig. 1-5. Creation of Source Module File .....	1-6
Fig. 1-6. Functions of C Compiler .....	1-7
Fig. 1-7. Functions of Assembler .....	1-8
Fig. 1-8. Functions of Linker .....	1-9
Fig. 1-9. Functions of Object Converter .....	1-10
Fig. 1-10. Functions of Librarian .....	1-11
Fig. 1-11. Functions of Screen Debugger .....	1-12
Fig. 4-1. I/O Files of This Compiler .....	4-3
Fig. 8-1. ROMable Processing .....	8-22
Fig. 8-2. Configuration of Error Handling Routine .....	8-29
 Table 1-1. Maximum Performance Characteristics of This C Compiler .....	 1-13
Table 2-1. System Files in Floppy Disk 1 .....	2-1
Table 2-2. Library Files in Each LIBNCA Directory .....	2-4
Table 2-3. System Configuration .....	2-6
Table 4-1. I/O Files of This C Compiler .....	4-1
Table 4-2. Optimization Techniques .....	4-9
Table 4-3. Error Handling Routines .....	4-16
Table 5-1. Types of Compiler Options .....	5-1
Table 5-2. Priority of Compiler Options .....	5-5
Table 5-3. Processor Types (78K/0 Series) .....	5-8
Table 5-4. Processor Types (78K/II Series) .....	5-9
Table 5-5. Processor Types (78K/III Series) .....	5-9
Table 5-6. Optimization Types .....	5-25
Table 5-7. Error Check Types .....	5-30
Table 5-8. Process Types with -K Option .....	5-35
Table 5-9. Warning Message Levels .....	5-67

	<u>Page</u>
Table 8-1. Contents of Directory "BAT" .....	8-2
Table 8-2. Contents of Directory "SRC" .....	8-3
Table 8-3. Contents of Directory "INC" .....	8-3
Table 8-4. Start-up Routine to Be Used .....	8-9
Table 8-5. Correspondence of Source Files to Object Files .....	8-10
Table 8-6. Selective Use of Start-up Routines .....	8-11
Table 8-7. Comparison of Contents between Start-up Routines .....	8-12
Table 8-8. Initial Values of Variables in ROM Area .....	8-22
Table 8-9. Initial Values of Variables in RAM Area (Copy Destination) .....	8-23
Table 8-10. Symbols to Be Used in Library Functions .....	8-25
Table 8-11. Error Handling Routines .....	8-30

## CHAPTER 1. GENERAL

The CC78K Series C Compiler is a program for translating source programs written in the C language for the 78K series into machine language.

## 1.1 What Is a C Compiler ?

### 1.1.1 C language and Assembly language

To have a microprocessor do its job, programs and data are necessary. These programs and data must be written by a human being (namely, a programmer in this case) and stored in the memory section of the microcomputer. Programs and data that can be handled by the microcomputer are nothing but a set or combination of binary numbers which is called machine language (that is, the language that can be understood or interpreted by the computer).

An assembly language is a symbolic language characterized by one-to-one correspondence of its symbolic (mnemonic) statements with machine language instructions. Because of this one-to-one correspondence, the assembly language can provide the computer with detailed instructions (for example, to improve I/O processing speed). However, this means that the programmer must instruct each and every operation of the computer. For this reason, it is difficult for him or her to understand the logic structure of the program at glance and the programmer is likely to make errors in coding.

High-level languages were developed as substitutes for such assembly languages. The high-level languages include a language called C which allows the programmer to write a program without regard to the architecture of the computer. As compared with assembly language programs, it can be said that programs written in C have easy-to-understand logic structure.

C has a rich set of parts called functions for use to create programs. In other words, the programmer can write a program by combining these functions.

C is characterized by its ease of understanding by human beings. However, understanding of languages by the microcomputer cannot be extended up to a program written in C. Therefore, to have the computer understand the C language program, another program is required to translate C language statements to the corresponding machine language instructions. A program which translates the C language into machine language is called a C compiler.

This C compiler accepts C source modules as inputs and generates object modules or assembler source modules as outputs. Therefore, the programmer can write a program in C and if he or she wishes to instruct the computer up to details of program execution, the C source program can be modified in assembly language. The flow of translation by this C compiler is illustrated in Fig. 1-1.

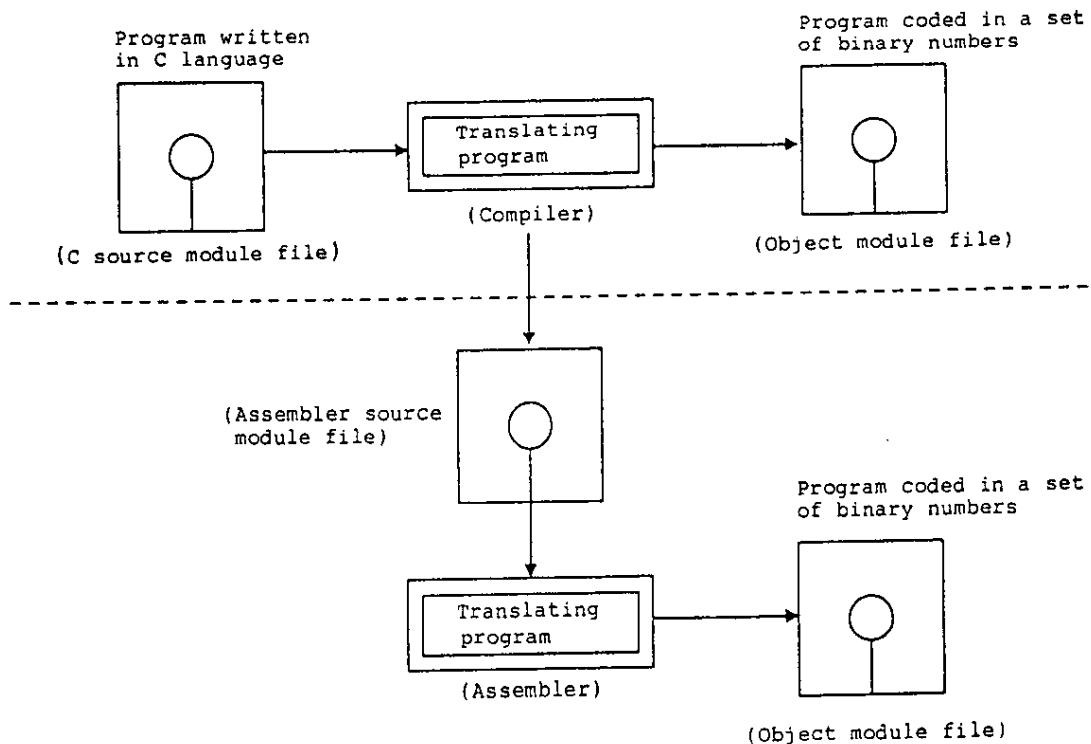


Fig. 1-1. Flow of Translation

### 1.1.2 Development of microcomputer-applied products and role of this product

Fig. 1-2 illustrates the standing of the programming in the C language in the development process of microcomputer-applied products.

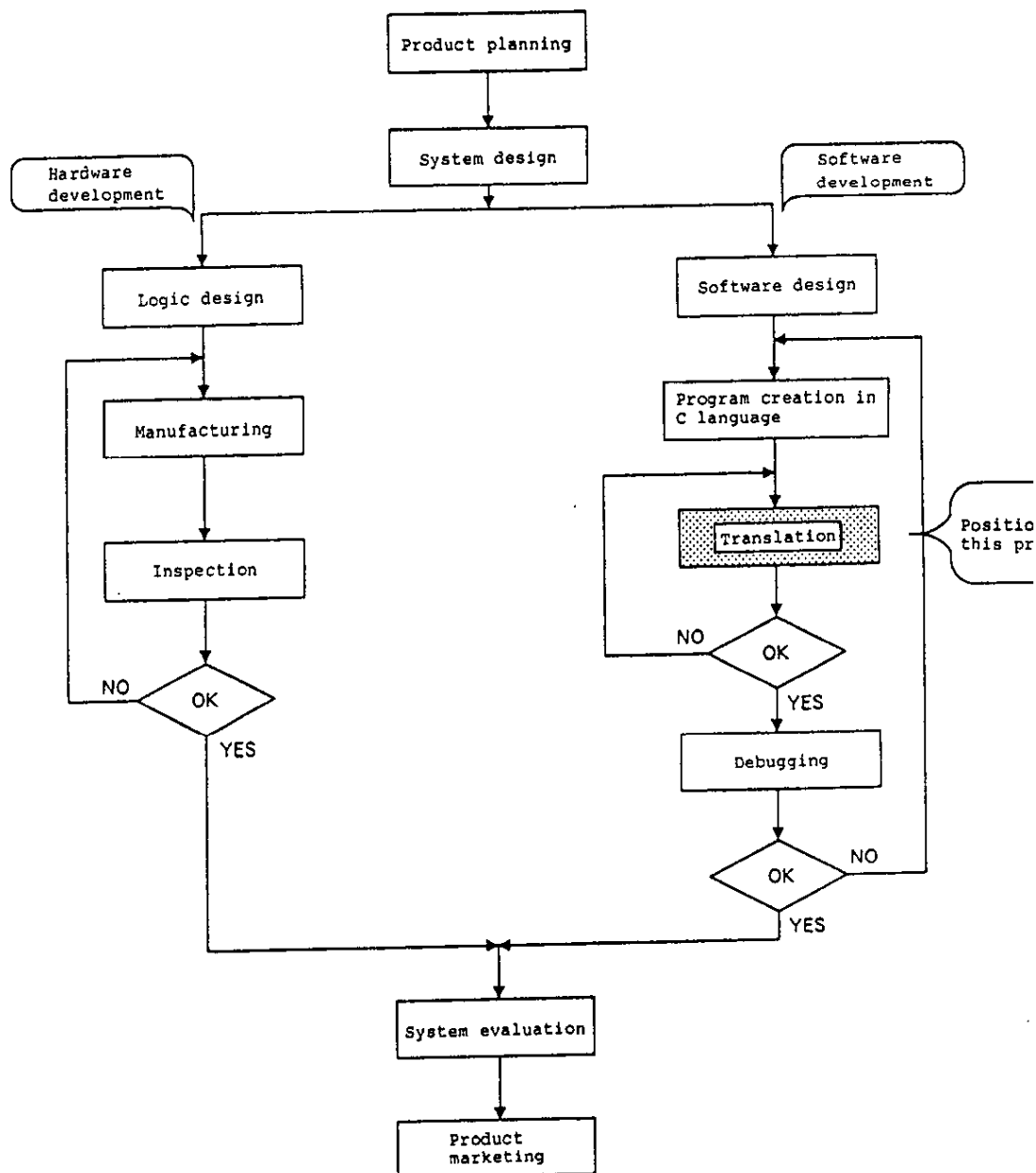


Fig. 1-2. Development Process of Microcomputer-applied Products

The software development process will be further detailed in Fig. 1-3 below.

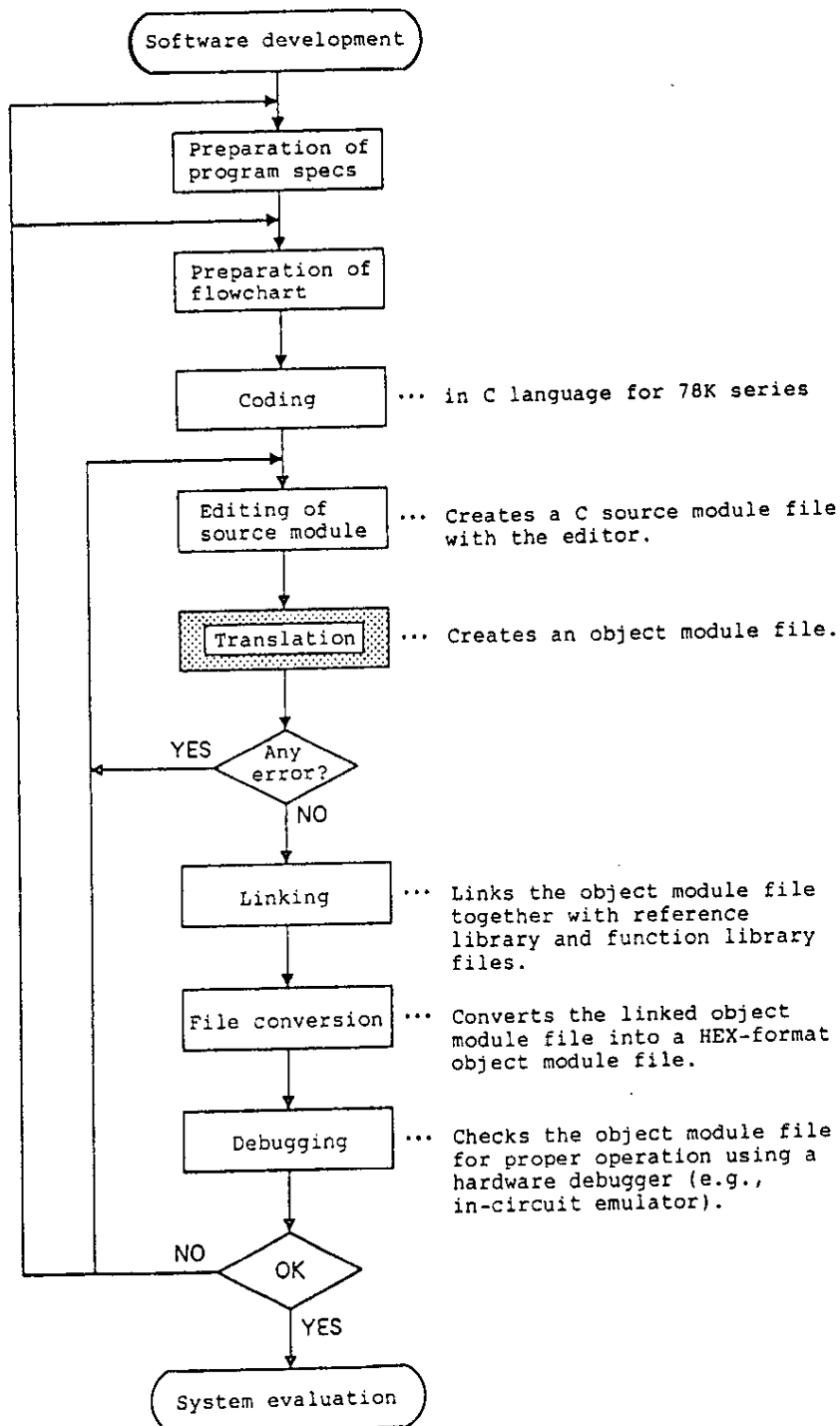


Fig. 1-3. Software Development Process

## 1.2 Program Development Procedure by C Compiler

Fig. 1-4 illustrates the program development procedure by this C compiler.

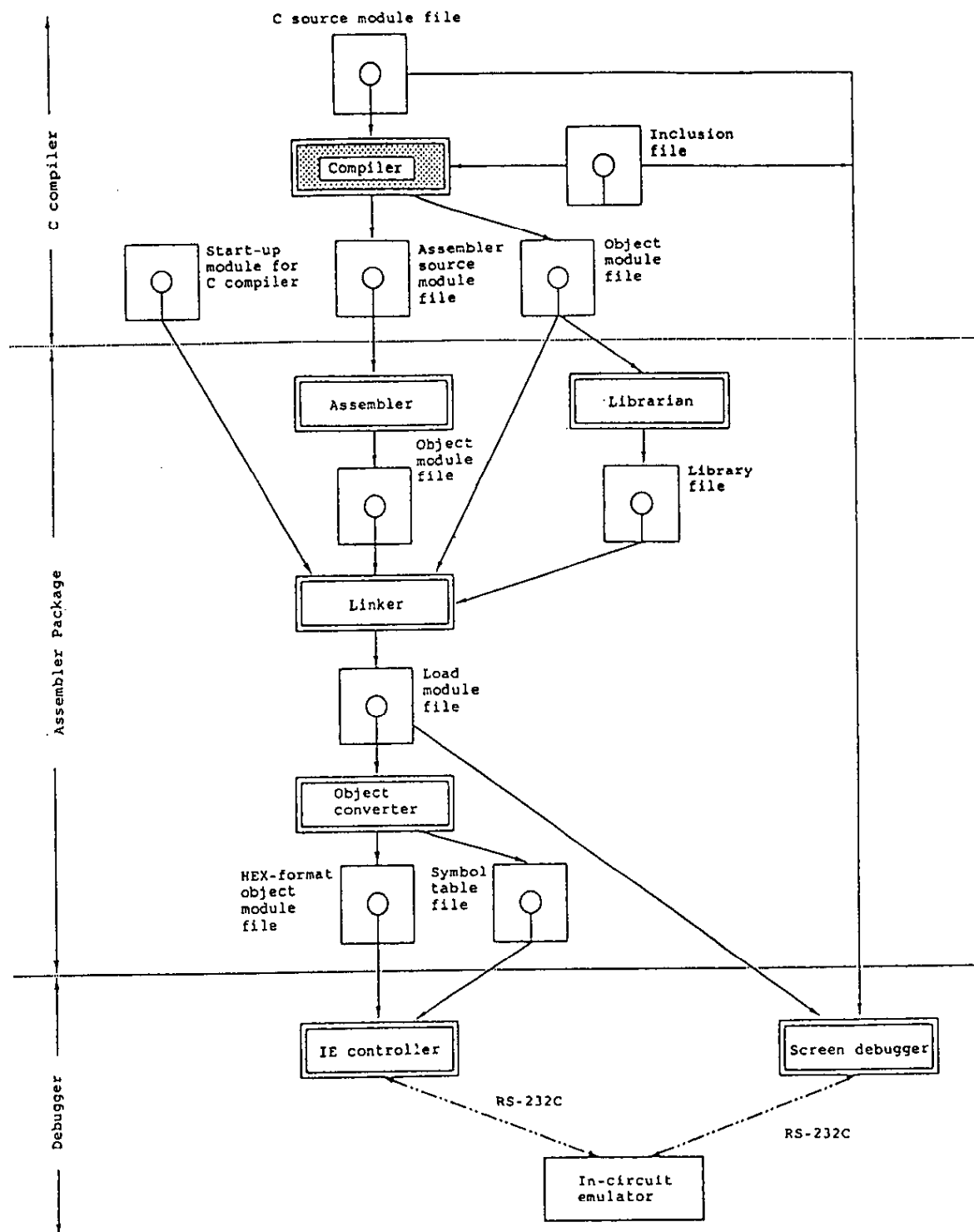


Fig. 1-4. Program Development Procedure by This C Compiler



### 1.2.1 Creating a source module file with the editor

Divide a program functionally into several modules. Each module becomes the unit of coding as well as the unit of input to the compiler. A module serving as the unit of input to the C compiler is called a C source module.

After coding each C source module, the source module is written into a file with the editor. The file thus created is called a C source module file.

The C source module file becomes an input file to this C compiler.

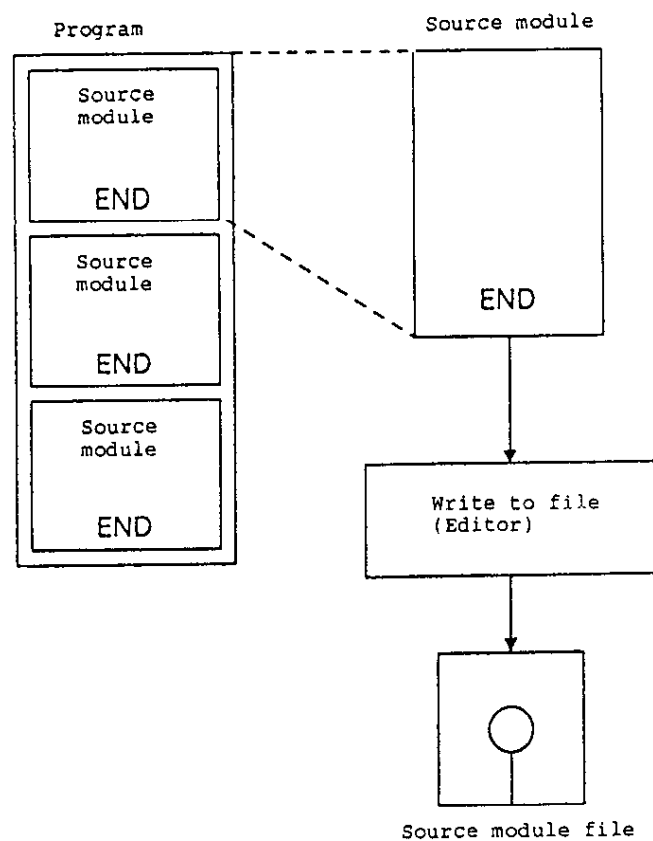


Fig. 1-5. Creation of Source Module File

### 1.2.2 C compiler

This C compiler accepts C source module files as input files and translates the C language of each input source module into machine language (a set or combination of binary numbers).

If any coding error is found in the input source module, the C compiler outputs a compile (translation) error. If no compile error is found, the C compiler outputs an object module file. The compiler may also output an assembler source module file to allow program modifications and verifications at the assembly language level. If an assembly source module file is to be output, the "-A" option must be specified at compile time to instruct the C compiler to create an assembly source module file.

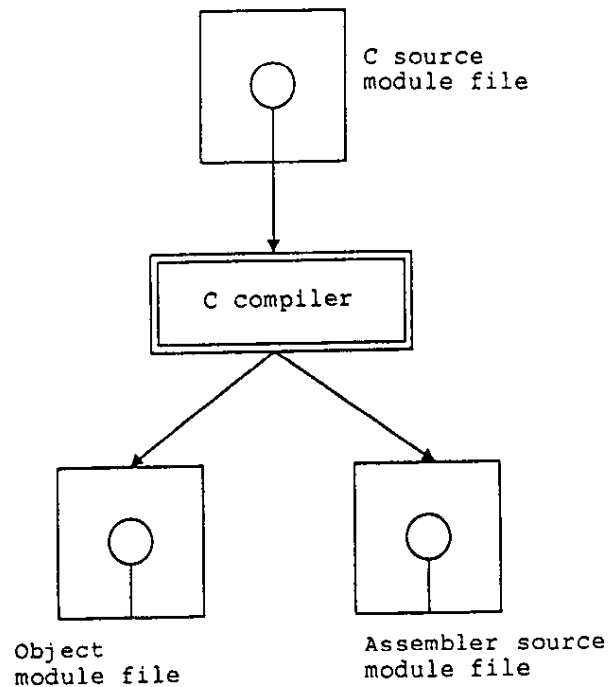


Fig. 1-6. Functions of C Compiler

### 1.2.3 Assembler

The assembler is a translating program which accepts assembler source module files as input files and translates the assembly language of each input source module into machine language. If any coding error is found in the input source module, the assembler outputs an assembly error. If no assembly error is found, the assembler outputs an object module file which contains machine language information and relocation information relating to the allocation address of each machine language instruction.

The assembler also outputs information at assembly time as an assembly list file.

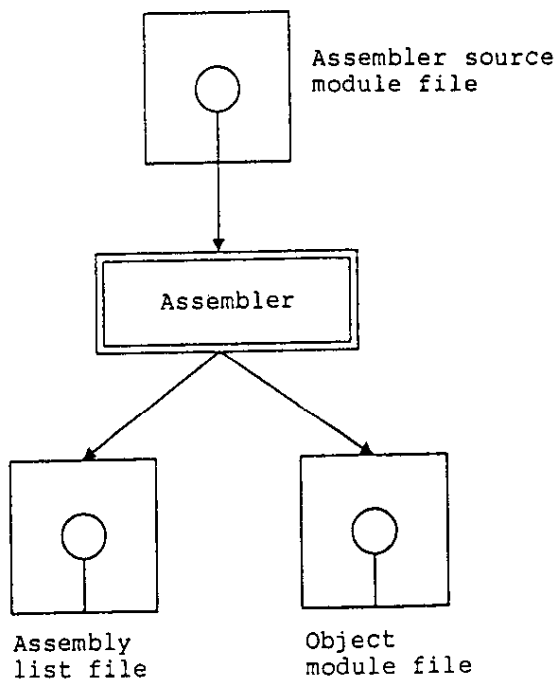


Fig. 1-7. Functions of Assembler

#### 1.2.4 Linker

The linker accepts two or more object module files output by the C compiler or assembler and gathers them with a library file for output as a single load module file. (This linking process is necessary even when only one object module is output.)

In this process, the linker determines the allocation addresses of relocatable segments in the input module, by which the correct value of each relocatable symbol or external reference symbol is determined and embedded in the load module file.

The linker also outputs information at link time as a link map file.

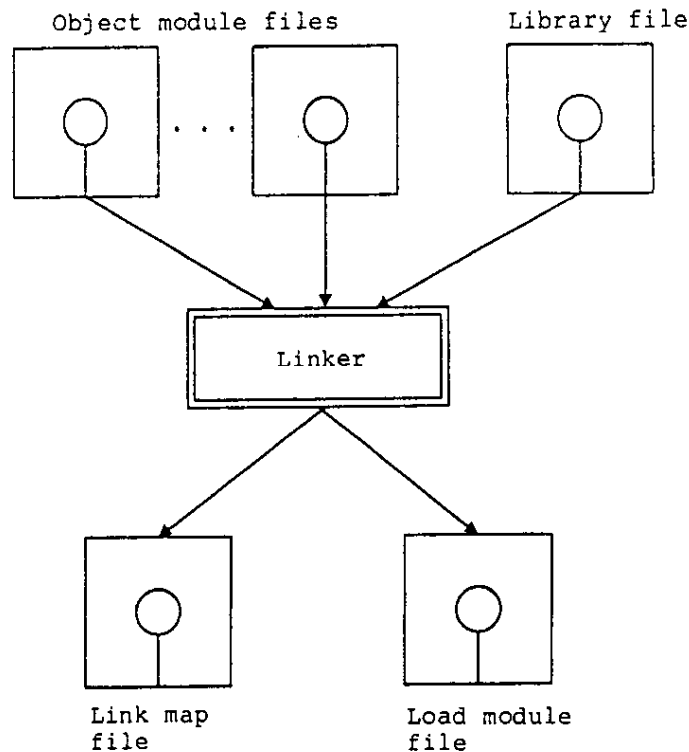


Fig. 1-8. Functions of Linker

### 1.2.5 Object converter

The object converter accepts the load module file output by the linker as an input file, converts it into a file format, and outputs the result of the conversion as a HEX-format object module file.

The object converter also outputs the symbol information required in symbolic debugging as a symbol table file.

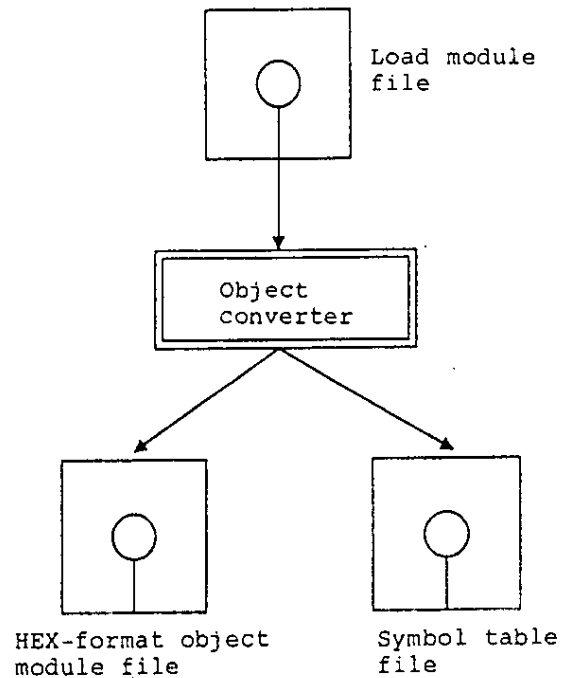


Fig. 1-9. Functions of Object Converter

### 1.2.6 Librarian

It is convenient to have a collection of modules which are to be used often and have a distinct interface maintained in a library. By this library, a number of modules can be handled easily as a single file.

The linker has a function to retrieve only the required modules from the library file. Therefore, by registering (storing) multiple modules in a single library file, each of the required module names need not be specified when linking object modules.

The librarian is used to create and update a library file.

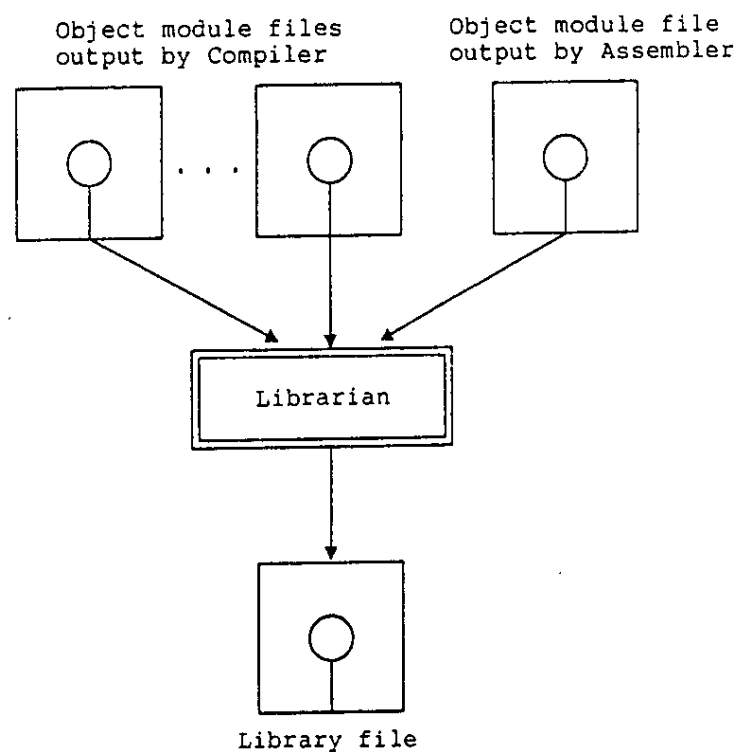


Fig. 1-10. Functions of Librarian

### 1.2.7 Screen debugger

By downloading the HEX-format object module file output by the object converter into the in-circuit emulator (IE) or evaluation board (EB board) of the target system and by reading the symbol table file, debugging of object programs can be performed at the symbolic level.

Another way to do this is to specify the option to output debugging information for a source program to be compiled. By this option specification at compile time, the symbol and line number information required for debugging will be added to the load object module to allow debugging of the program at the source level.

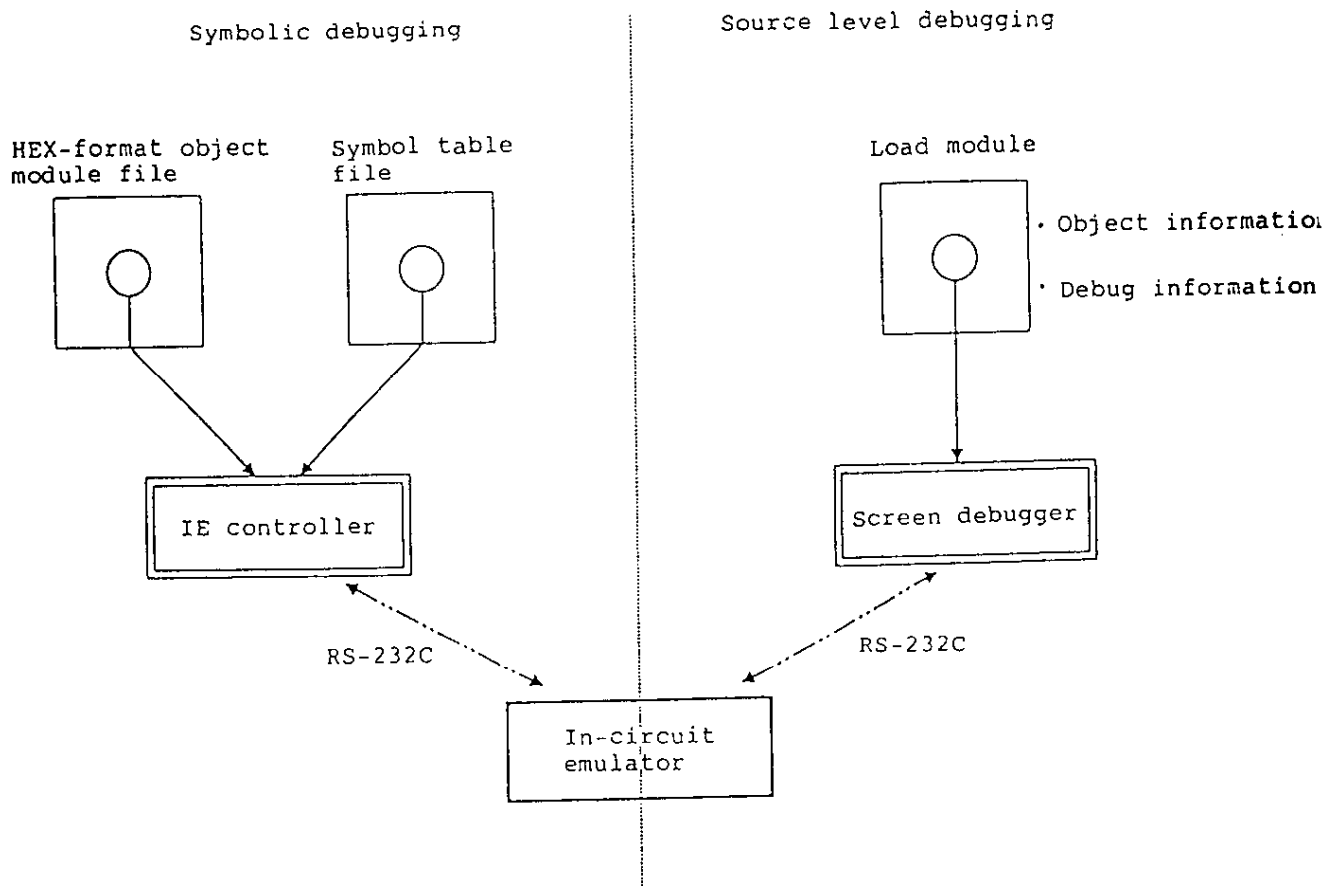


Fig. 1-11. Functions of Screen Debugger

## 1.3 Reminders Before Program Development

Before you set your hand to the development of a program, keep in mind the points (limit values or minimum guaranteed values) summarized in Table 1-1 below.

Table 1-1. Maximum Performance Characteristics of  
This C Compiler

No.	Item	Limit value/Min. guaranteed value
1	Nesting level of compound statements, looping statements, or conditional control statements	45 levels
2	Nesting of conditional translations	255 levels
3	Nesting of qualifiers/declarators	12 levels
4	Nesting of parentheses per expression	32 levels
5	Number of characters which have a meaning as a macro name	31 characters
6	Number of characters which have a meaning as an internal or external symbol name	7 characters (see Note 1.)
7	Number of symbols per source module file	1,024 symbols (see Note 2.)
8	Number of symbols which has block scope within a block	255 symbols (see Note 2.)
9	Number of macros per source module file	1,024 macros (see Note 3.)
10	Number of parameters per function definition or function call	39 parameters
11	Number of parameters per macro definition or macro call	31 parameters
12	Number of characters per logical source line	509 characters
13	Number of characters within a string literal after linkage	509 characters
14	Size of one data object	65,535 bytes
15	Nesting of <code>#include</code> directives	8 levels
16	Number of case labels per <code>switch</code> statement	257 labels
17	Number of source lines per translation unit	Approx. 3,000 lines
18	Number of source lines that can be translated without temporary file creation	Approx. 300 lines
19	Nesting of function calls	40 levels
20	Number of labels that can be declared per statement	33 labels



No.	Item	Limit value/Min. guaranteed value
21	Total size of code, data, and stack segments per object module	65,535 bytes
22	Number of members per structure or union	127 members
23	Number of <b>enum</b> constants per enumeration	127 constants
24	Nesting of structures or unions inside a structure or union	15 levels
25	Nesting of initializer elements	15 levels

The values of items 5, 6, 12, 13, 14, 15, and 21 are limit values and the values of all the other items are minimum guaranteed values.

For details of each item in this table, see the CC78K Series C Compiler Use's Manual for Language.

- NOTE:
1. This limit value may be expanded to 30 characters with a C compiler option (-S).
  2. This value applies when symbols can be processed with the available memory space alone without using any temporary file. When a temporary file is used because of insufficient memory space, this value must be changed according to the file size.
  3. This value includes the reserved macro definitions of the C compiler.

## 1.4 Features of This C Compiler

This C compiler has extended functions for CPU code generation that are not supported by the ANSI (American National Standards Institute) Standard C. The extended functions of the C compiler allow the special function registers for the 78K series to be described at the C language level and thus help shorten object code and improve program execution speed. For details of these extended functions, see Chapter 11, Extended Functions in the CC78K Series C Compiler User's Manual for Language.

Outlined here are the following extended functions to help shorten object code and improve execution speed:

- o **callt** functions ..... Functions can be called using the callt table area.
- o **Register variables** ... Variables can be allocated to registers.
- o **saddr** area ..... Variables can be allocated to the saddr area.
- o **sfr** area ..... sfr names can be used.
- o **noauto** functions ..... Functions which do not output  
  **norec** functions           code for stack frame formation  
                              can be created.
- o **ASM** statements ..... An assembly language program can be described in a C source program.
- o **bit** type variables ... Accessing the saddr or sfr area can be made on an bit-by-bit basis.
- o **callf** functions ..... A function body can be stored in the callf area.

① **callt functions**

Functions can be called by using the **callt** table area. The address of each function to be called (this function is called a **callt** function) is stored in the **callt** table from which it can be called later. This makes code shorter than the ordinary call instruction and helps shorten object code.

② **Register variables**

Variables declared with the **register** storage class specifier are allocated to the register or **saddr** area. Instructions to the variables allocated to the register or **saddr** are shorter in code length than those to memory. This helps shorten object and improves program execution speed as well.

③ **saddr area**

Variables declared with the keyword **sreg** can be allocated to the **saddr** area. Instructions to these **sreg** variables are shorter in code length than those to memory. This helps shorten object code and also improves program execution speed.

④ **sfr area**

By declaring use of **sfr** names, manipulations on the **sfr** area can be described at the C source level.

⑤ **noauto functions**

Functions declared as **noauto** do not output code for preprocessing and post-processing (stack frame formation). By calling a **noauto** function, arguments are passed via registers as much as possible. This helps shorten object code and improve program execution speed as well.

⑥ **norec functions**

Functions declared as **norec** do not output code for preprocessing and post-processing (stack frame formation). By calling a **norec** function, arguments are passed via registers as much as possible. Automatic variables to be used inside a **norec** function are allocated to the **saddr** area. This helps shorten object code and also improve program execution speed.

⑦ **bit type variables**

A **bit** type variable outputs a bit manipulation instruction to an external variable which has no initial value (or has a unknown value). This allows programming in C at the assembler source level as well as accessing to the **saddr** and **sfr** areas on a bit-by-bit basis.

⑧ **ASM statements**

The assembler source program described by the user can be embedded in an assembler source file to be output by this C compiler.

⑨ **Interrupt functions**

- o The preprocessor directive **#pragma vect** outputs a vector table and outputs an object code corresponding to the interrupt. This directive allows programming of interrupt functions in the C source level.
- o The preprocessor directive **#pragma DI** or **#pragma EI** creates a function to disable or enable interrupts.

⑩ **callf function**

The **callf** instruction stores the body of a function in the **callf** area and allows the calling of the function with a code shorter than that with the **call** instruction.

- ⑪ Use of 1M-byte extended space (supported by the 78K/II only)

The preprocessor directive **#pragma extend** outputs a code to access the 1M-byte extended space to an object through direct in-line expansion without resort to a function call and creates an object file.

- ⑫ Table conversion function (supported by the 78K/III only)

The preprocessor directive **#pragma table** allows the addresses of the vector table and callt table output by the C compiler to be changed.

## CHAPTER 2. PRODUCT OVERVIEW

## 2.1 Contents of Floppy Disks

The floppy disks supplied as this product contain the following two types of files:

System files (Floppy disk 1)	{	BIN (Files for execution)
		INCLUDE (Header file)
		SAMPLE (Sample programs)
		LIB (Standard directive file for linking)
		BAT (Start-up routine, Error handling routine)
		SRC (Related files)
		INC (Device information Include file)

Library files .... Library file for each device  
(2nd and subsequent floppy disks)

## 2.1.1 System files

System files are offered in the supplied floppy disk 1. The contents of each directory stored in the floppy disk 1 are as listed in Table 2-1 below.

Table 2-1. System Files in Floppy Disk 1

Directory name	Filename	Role of file
BIN	CC78Kn.EXE	Executable files
	SA78Kn.EXE	Controller
	XO78Kn.EXE	Syntax analyzer section
		Cross-reference output section
	CG78Kn.EXE	Code generator section
	LO78Kn.EXE	Object list output section
	OP78Kn.EXE (V2.00 or later)	Optimizer section
	CC78Kn.OM1 }	Overlay files (contain device information)
	CC78Kn.OMx	See Note 2 below.
	CC78Kn.HLP	Help file
	CC78Kn.MSG	Message file
INCLUDE	ooo.H	See Section 10.2, "Headers" in the CC78K Series User's Manual for Language.
INCLUDE ¥78xxx	SFRBIT.H	Header file defining sfr bit function names.
SAMPLE	PRIME.C	Sample C source file
	SAMPLE.BAT	Batch file
	README.DOC	Document file

Table 2-1. System Files in Floppy Disk 1 (contd)

Directory name	Filename	Role of file
LIB	LINKxxx.DIR	Standard directive file for linking
	LINKxxx.ROM	.DIR ... Not for ROMable processing .ROM ... For ROMable processing xxx: Device type (see Tables 5-3 to 5-5 for device types)
LIB¥BAT	MKSTUP.BAT	Batch file for start-up routine creation
	MKERRLIB.BAT	Batch file for updating error handling routine
	ooo.ERR	File used for MKERRLIB.BAT (Subcommand file used when starting up the Librarian)
LIB¥SRC	ooo.ASM	Source file for start-up routine and error handling routine
	ooo.INC	Inclusion file for ooo.ASM
LIB¥INC	ooo.INC	Device information Inclusion file

Note: 1. n = 0, 2, 3, where n indicates each 78K series number (78K/0, 78K/II, or 78K/III).

2. The number of overlay files differs depending on the 78K series.

Remarks: 1. A command file or executable file (with the file type .EXE) is the first file to be read into memory when the program is started up.

2. Overlay files as many as required will be read into memory during the program execution.



### 2.1.2 Library files

Library files are offered in the second and subsequent floppy disks supplied as this product.

- o Library files consist of standard library and start-up routine files for each device.
- o Each floppy disk contains the following two directories:

LIBNCA ..... This directory offers standard library and start-up routine files required when linking object files created by specifying the compiler option -NCA.

LIBCA ..... This directory offers standard library and start-up routine files required when linking object files created by specifying the compiler option -CA.

The following table shows the contents of each LIBNCA directory.

Table 2-2. Library Files in Each LIBNCA Directory

Directory name	Filename (see Note 1)	Role of file
LIBNCA	CLKnCOM.LIB	Library file common to 78K/n (see Note 2)
LIBNCA¥ LIBxxx	CLxxx.LIB	Library file for sfr check (see Note 3)
	CSxxx.REL	Start-up routine files
	CSxxxR.REL	
	ESxxxR.REL	
	ROMxxx.REL	ROMable module file

Notes: 1. xxx: Processor type (see Tables 5-3 to 5-5 in Chapter 5)

n : n = 0, 2, or 3, where n indicates each 78K series name (78K/0, 78K/II, or 78K/III).

2. For the following processor types (target devices), use the library file indicated below for each processor type in lieu of the common library file.

310 → CL310.LIB      312 → CL312.LIB

310A → CL310A.LIB      312A → CL312A.LIB

3. CLxxx.LIB is a file used to check an illegal read- or write-access to sfr (special function registers). This library file for error check will be incorporated in the object file if you specify the C compiler option "-L" (option for execution-time error check specification) at the time of starting up the C compiler. Therefore, when you have specified this -L option, also specify CLxxx.LIB as a library file together with the common library file at linkage time.

The contents of the directory LIBCA are the same as those listed for the directory LIBNCA except that "U" is suffixed to each filename.

## 2.2 Forms of File Media Supplied

This product is offered in either of the following two forms of file media:

- o 5-inch double-sided high-density (2HD) floppy disk
- o 3.5-inch double-sided high-density (2HD) floppy disk

## 2.3 System Configuration

This C compiler operates in the environment indicated in Table 2-3 below.

Table 2-3. System Configuration

Host computer	OS		Memory size
	CPU	CONFIG.SYS	
PC-9800 series	V30 <sup>TM</sup> 80386 <sup>TM</sup> 80286 <sup>TM</sup> 8086 <sup>TM</sup>	MS-DOS (V2.11, V3.10) V3.30A)	360 KB may be used.
IBM PC IBM PC/XT <sup>TM</sup> IBM PC/AT <sup>TM</sup>		PC-DOS (V3.10)	

- Note: 1. The CC78K series C compiler operates on the MS-DOS that NEC offers for the PC-9800 series personal computers. NEC Corporation is not responsible for improper operation of this program on any other commercially available MS-DOS.
2. Memory size denotes the maximum value of memory required for this C compiler to operate. This memory size does not include any system area.
3. With PC-DOS, a backslash (\) is used in place of the ¥ sign.

## CHAPTER 3. EXECUTION OF C COMPILER

This chapter describes how to execute this C compiler using the CC78K3 as an example. By actually executing a sample program according to the procedure described in this chapter, you may accustom yourself to the operations of the C compiler.

### 3.1 Before Executing the C Compiler

#### 3.1.1 Confirming the contents of the supplied disk

Set the supplied floppy disk of this C compiler (or its backup disk) in the available drive and confirm that the disk contains all the program files listed in Section 2.1, Contents of Floppy Disks.

## 3.1.2 Sample program

For the purpose of explaining how to execute this C compiler, the following sample program named "PRIME.C" is used.

```

#define TRUE    1
#define FALSE   0
#define SIZE    200

char    mark[SIZE+1];

main()
{
    int i, prime, k, count;

    count = 0;

    for ( i = 0 ; i <= SIZE ; i++)
        mark[i] = TRUE;
    for ( i = 0 ; i <= SIZE ; i++) {
        if (mark[i]) {
            prime = i + i + 3;
            printf("%6d", prime);
            count++;
            if((count%8) == 0) putchar('\n');
            for ( k = i + prime ; k <= SIZE ; k += prime)
                mark[k] = FALSE;
        }
    }
    printf("\n%d primes found. ", count);
}

printf(s, i)
char *s;
int i;
{
    int j;
    char *ss;

    j = i;
    ss = s;
}

putchar(c)
char c;
{
    char d;
    d = c;
}

```

Note: With PC-DOS, use a backslash (\) in place of the ¥ sign used in the above sample program.

### 3.2 Procedure for C Compiler Execution

- (1) Compile the sample program "PRIME.C".
  - o Enter the compiler start-up command line as follows:  
(The command line may be described in either uppercase or lowercase letters for input.)

A>cc78k3 -c310 sample\$prime.c

└────────── Specifies the type (model number)  
                  of the target device.

- o The following message will be output to the console:

```
78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx
```

- (3) Check the contents of drive A.

The C compiler has created PRIME.REL (object module file).

By specifying compiler options such as -A, -P, -X, and -E in the start-up command line of the C compiler, this C compiler can output four list files; an assembler source module file, a preprocess list file, a cross-reference list file, and an error list file in addition to the object module file.



## CHAPTER 4. C COMPILER

This C compiler accepts source module files coded in the C assembly language for the 78K series and Inclusion files as input files, translates the C language coding of each source module into machine language coding, and outputs them as an object module file. After the translation, the C compiler can also output an assembly source module file to allow program modifications and verifications at the assembly language level. In addition, the C compiler outputs list files such as a preprocess list file, a cross-reference list file, and an error list file.

If any compile error is found during the translation of the input source module file, the C compiler outputs the error message to the console or error list, but will generate none of the above-mentioned output files except the error list file.

#### 4.1 Input/Output Files of C Compiler

The files listed in Table 4-1 below are input and output to and from this C compiler.

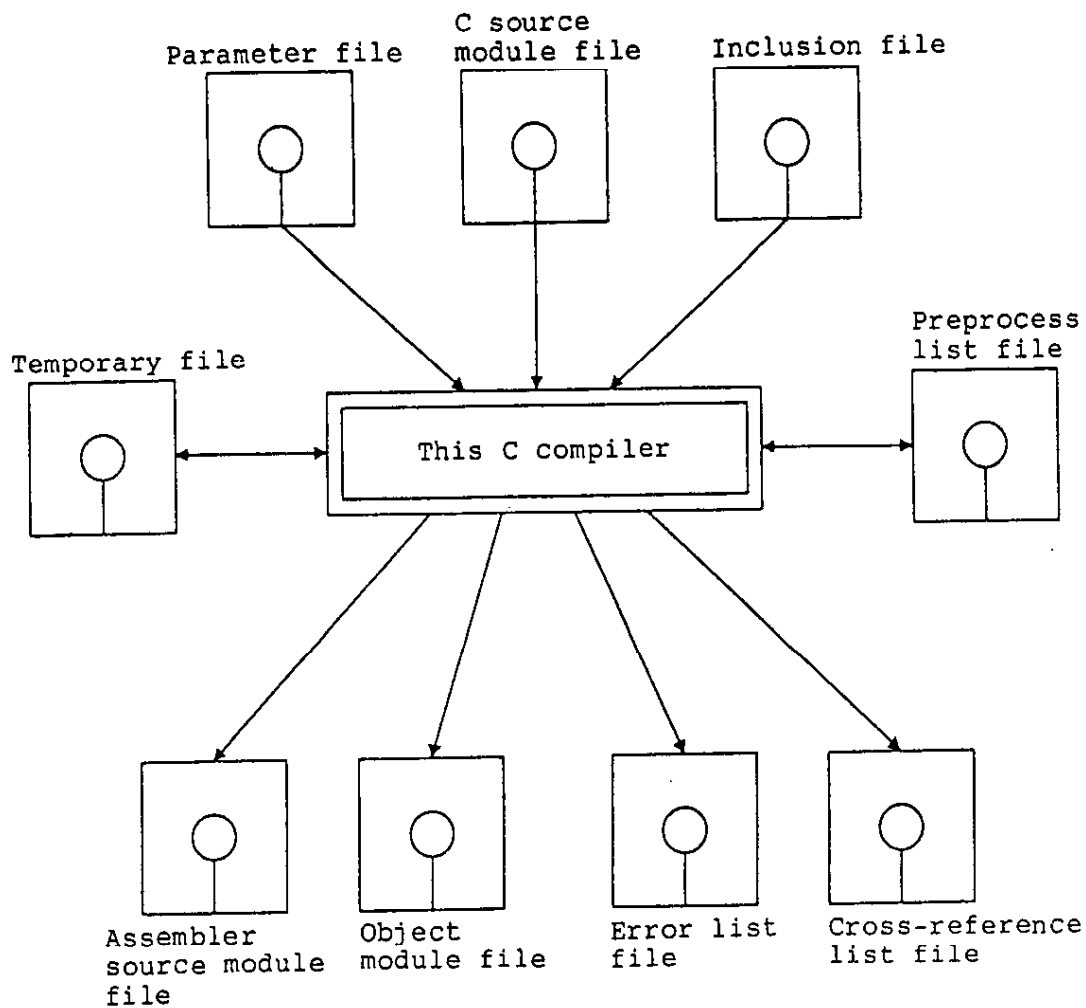
Table 4-1. I/O Files of This C Compiler

Type	Description of file	Default file type
Input files	<u>C source module file</u> ... A program file coded in the C language for the 78K series) that must be translated into machine language before use. (This file must be created by the user.)	.C
	<u>Inclusion file</u> .... A file that can be referenced in a C source module file or a program file coded in the C language for the 78K series. (This file must be created by the user.)	.H
	<u>Parameter file</u> .... A file created by the user to specify a number of commands which are excessive to specify in the start-up command line of the C compiler. (This file must be created by the user using the editor.)	.PCC



Table 4-1. I/O Files of This C Compiler (contd)

Type	Description of file	Default file type
Output files	<u>Object module file</u> .... A binary image file contains machine language information, relocation information on the address of each machine-coded instruction, and symbol information.	.REL
	<u>Assembler source module file</u> ..... An ASCII image object code file containing the result of the translation by the compiler.	.ASM
	<u>Preprocess list file</u> .... An ASCII image list file containing the results of preprocessor directives such as <code>#include</code> .	.PPL
	<u>Cross-reference list file</u> .... A list file containing information on the function names and variable names used in the C source module file compiled.	.XRF
	<u>Error list file</u> ..... A list file containing information on source files and compile error messages.	.ECC .CER .HER .ER
I/O file	<u>Temporary file</u> ... An intermediate file for compilation. On normal termination of the compiler, this file will be renamed by a formal name. On abnormal termination of the compiler, this file will be deleted.)	.\$nn (Fixed to this name)



Remarks: If any error is found during the translation of an input source module file, none of the above listed output files except the error list file will be created. The temporary file will be renamed by a formal name on normal termination of the compiler and deleted on abnormal termination of the compiler.

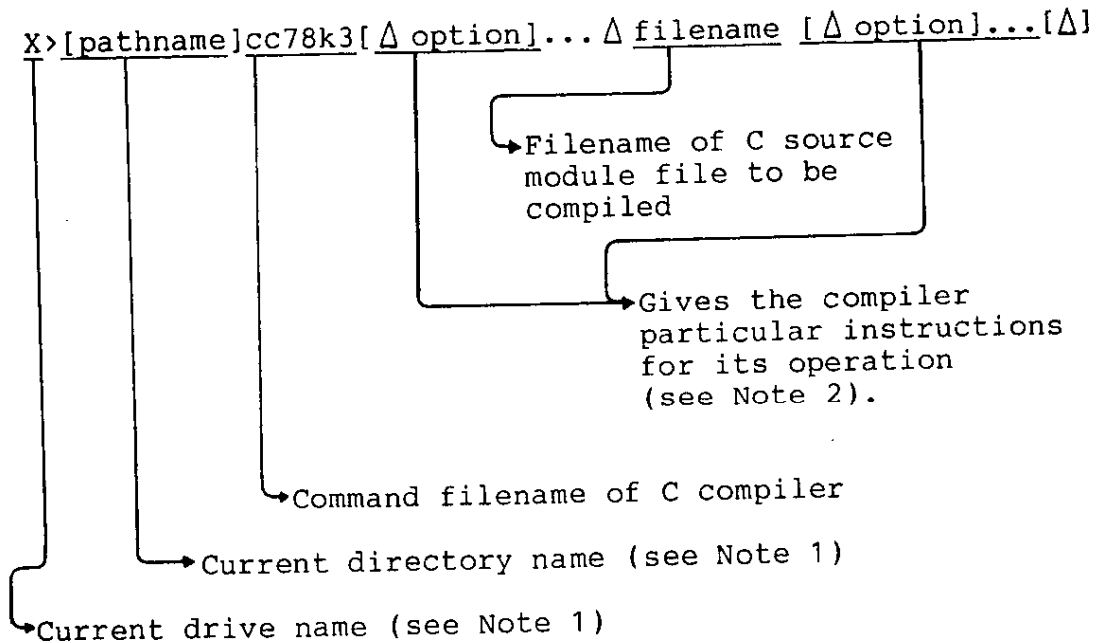
Fig. 4-1. I/O Files of This Compiler

## 4.2 How to Start Up the C Compiler

### 4.2.1 Starting up the C compiler

The C compiler can be started up in either of the following two ways:

(1) Start-up with command line:



Example: A>cc78k3 -c310 prime.c -a -p -x -e

- NOTE: 1. With MS-DOS version 2.11, the command files of the C compiler must have been stored in the current drive or current directory.
2. If two or more compiler options are to be specified, each compiler option must be delimited with a space. These options may be described in either uppercase or lowercase. See Chapter 5, Compiler Options for details of each option.

## (2) Start-up with parameter file

When specifying two or more options at compile time, you may be occasionally compelled to repeat the same specifications over and over again because the required information for starting up the compiler with a command line is excessive. In such a case, a parameter file can be used to start up the compiler.

When using a parameter file, specify the parameter file specification option in the command line as follows:

```
X>cc78k3[△ C source module file]△ -f parameter filename
```

File containing  
information  
required for  
starting up the  
compiler

Parameter file  
specification  
option

- o The parameter file must be created with the editor.
- o Conventions of option description within a parameter file are as follows:

```
[ [ [△]option[ △ option] ...[△]△]] ...
```

- o Only one C source module filename can be specified for the parameter file.
- o The C source filename may be described after compiler options.
- o All the compiler options and the output filename to be specified in the command line must be described in the parameter file.

Example: Create a parameter file named "PRIME.PCC" with the editor.

- o Contents of parameter file "PRIME.PCC"

```
sample\prime.c -c310 -aprime.asm
-e -x
```

- o To start up the C compiler using parameter file "PRIME.PCC", enter as follows:

```
A>cc78k3 -fprime.pcc
```

#### 4.2.2 Execution start and end messages

##### (1) Execution start message

When the C compiler is started up, the following message is output to the console, indicating the start of the compiler execution.

```
78K/III Series C Compiler Vx.xx [xx xxx xx]
Copyright (C) NEC Corporation xxxx
```

##### (2) Execution end messages

- o If no compile error is found as a result of a compile operation, the C compiler will output the following message to the console and return control to the OS.

```
Compilation complete,      0 error(s) and      0 warning(s) found.
```

- o If any compile error is found as a result of a compile operation, the C compiler will output the following message to the console and return control to the OS.

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype
SAMPLEYPRIME.C(20) : W745 Expected function prototype
SAMPLEYPRIME.C(26) : W622 No return value
SAMPLEYPRIME.C(37) : W622 No return value
SAMPLEYPRIME.C(44) : W622 No return value
Compilation complete,      0 error(s) and      5 warning(s) found.
```

- o If any fatal error (which makes the C compiler impossible to continue its processing) is found during a compile operation, the C compiler will output the following message to the console, stop its processing, and return control to the OS.

Example 1:

```
A><cc78k3 -c310 -e sample.c
```

```
uCOM-78K/III Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx
```

```
A006 File not found 'SAMPLE.C'  
Program aborted
```

In this example, the compile operation was discontinued by a fatal error resulting from the specification of a C source module file which does not exist in the current drive A.

Example 2:

```
A><cc78k3 -c310 -e sampleYprime.c -m
```

```
uCOM-78K/III Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx
```

```
A018 Option in not recognized '-m'  
Program aborted
```

In this example, the compile operation was discontinued by a fatal error resulting from the input of a compiler option "-m" which is not recognized by the C compiler.

If the compiler is aborted following the output of an error message, check the cause of the error message by referring to Chapter 9 Error Messages and take corrective action(s) as required.

#### 4.3 C Compiler Options

When you start up the C compiler, you may specify a compiler option. This compiler option gives the C compiler particular instructions for its operation.

Because two or more compiler options can be specified at the same time, you may select compiler options according to the specific purpose so that compilation of C source module files can be performed efficiently.

For the types of compiler options, how to specify compiler options, and priority of these options, see Chapter 5, Compiler Options.

#### 4.4 Optimization

With this C compiler, optimization is carried out for creation of efficient object module files. The optimization techniques supported by this C compiler are shown in Table 4-2.

Table 4-2. Optimization Techniques

Optimization phase		Description	Example
Syntax analyzer section	①	Constant calculation at compile time	$a=3*5; \rightarrow a=15;$
	②	True/False check by partial evaluation of logical expression	$0\&\&(a  b) \rightarrow 0$ $1  (a\&\&b) \rightarrow 1$
	③	Offset computation for pointer, array, etc.	Compiler computes offset values at compile time.
Code generator section	④	Register management	Compiler puts unused registers to effective use.
	⑤	Utilization of special instructions of target CPU	$a=a+1; \rightarrow$ Compiler uses "inc" instruction for this. It also uses move instructions for assigning array elements.
	⑥	Utilization of shorter instructions	mov a,#0 or xor a,a (differs depending on the device) If there are two instructions which do the same thing, Compiler uses the one shorter in byte length.
	⑦	Conversion of long jump to short jump instruction	Compiler does this conversion by re-manipulating the output intermediate code.

Remarks: The compiler executes the above optimization phases ① to ⑦ even if the optimize option is omitted from specification.



Table 4-2. Optimization Techniques (contd)

Optimization phase		Description	Example
Optimizer section	⑧	Deletion of common partial expression	$a=b+c;$ $\rightarrow$ $a=b+c;$ $d=b+c+e;$ $d=a+e;$
	⑨	Movement to outside instruction loop	<pre> for(i=0;i&lt;10;i++) { ... a=b+c; ... } </pre> <p style="text-align: center;">↓</p> <pre> a=b+c; for(i=0;i&lt;10;i++) { ... ... } </pre>
	⑩	Deletion of unwanted instructions	$a=a;$ $\rightarrow$ Delete $a=b;$ $\rightarrow$ Delete Hereafter, "a" will not be referred.
	⑪	Deletion of copies	$a=b;$ $c=a+d;$ $\rightarrow$ $c=b+d$ Hereafter, "a" will not be referred. (a is an automatic variable.)
	⑫	Change of order of operations on expressions	Compiler executes first an operation which becomes valid by leaving the operation result in a register.
	⑬	Memory allocation (Temporary variables)	Compiler allocates locally used variables to registers.
	⑭	Peep hole optimization	Substitution of special pattern Example: $a*1 \rightarrow a$ , $a+0 \rightarrow a$
	⑮	Alleviation of operation strength	Example: $a*2 \rightarrow a+a$ , $a<<1$
	⑯	Memory allocation (Register variables)	Compiler allocates data to quickly accessible memory. Example: Register area, saddle area

- Remarks:
- o The compiler executes the above optimization phases ⑧ to ⑬ only when the optimize option is specified.
  - o The compiler executes the above optimization phases ⑭ and ⑮ even if the optimize option is omitted from specification.
  - o The compiler executes the above optimization phase ⑯ only when a register declaration is contained in the C source program or when the optimize option is specified.
  - o The above optimization phases ⑧ to ⑬ and ⑯ are intended for future support.
  - o With the CC78K2 C compiler, more sophisticated optimization such as ⑭ and ⑮ can be executed by specifying the optimize option.

## 4.5 ROMable Processing of Programs

This section explains how to make object programs ROMable. The ROMable processing function refers to storing initial values such as external variables with initial values into a ROM and copying the contents of the ROM into a RAM at the time of the system execution.

### 4.5.1 At compile time

By specifying the -R option at compile time, the object program output by the compiler becomes ROMable. Because the function to make programs ROMable is the default value, the -R option may be omitted.

Specify the -NR option if the program need not to be subjected to ROMable processing.

### 4.5.2 At linkage time

When linking object module files, these object module files must be linked with an appropriate start-up module file. The start-up module file initializes the object program. If the ROMable (-R) specification option is specified, the start-up module file must be changed to the one which allows the ROMable processing of programs. (The start-up module files shown in the examples below are applicable when the target device is the uPD78310.)

#### (1) When the program is subject to ROMable processing

```
cs310r.rel ..... Start-up module file for device
                    model 310 (which corresponds to
                    ROMable module file). This file
                    contains an initialize data
                    copy routine and indicates the
                    start of initialize data.
rom310.rel ..... Indicates the end of initialize
                    data.
```

Module files are linked in the order beginning with cs310r.rel and ending with rom.rel.

- (2) When the program is not subject to ROMable processing  
cs310.rel ..... Start-up module file for device  
model 310.

Module files may be linked in any order.

When executing the user program, execution starts with the start-up module file. The symbol (label name) to be used as the start address is `_@cstart`.

## 4.6 Error Check at Execution Time

When the source debugger is activated, the compiler may perform operations which are not intended by the user because of various errors. For this reason, object code for error checking should be output in addition to ordinary object code by specifying the -L option when starting up the C compiler.

### 4.6.1 Error handling routine

If an error is found, an error handling routine is called. This routine is a permanent loop which has preprocessing and post-processing. By checking the address where a break has occurred with this routine, the compiler can find which type of error has occurred. Table 4-3 shows the types of error handling routines.

Table 4-3. Error Handling Routines

Check item	Routine name	Function
Initialize data	errini.asm	This routine is called if the size of the initialize data allocated to the ROM area does not match with the segment size of the RAM area.
Stack check	errstk.asm	This routine is called if a stack overflow occurs.
Divide by 0	errdiv.asm	This routine is called if division with divisor 0 is attempted.
Pointer access	errptr.asm	This routine is called if illegal area accessing with a pointer is attempted.
Overflow	errovf.asm	This routine is called if an overflow occurs in any of various operations.
sfr access	errsfr.asm	This routine is called if illegal read- or write-access to the sfr area is attempted.

#### 4.6.2 Error check library names

When the error check specification (-L or -NL) option is specified, library names become as follows:

##### (1) Run time libraries

To identify a run time library name without error check, characters "@" are prefixed to the symbol (function name). For a run time library with error check, the prefix characters "@" must be changed as shown below depending on the type of error check:

Without error check	@@function name
Divide by zero	?@function name
Divide by zero + Overflow	?_function name
Overflow	@_function name
Stack check	__function name
sfr access	??function name

(In sfr access, there is no library without error check.)

Assignment to sfr.bit	??asto8
Assignment to sfr	??asto8
Assignment to sfrp	??asto16

An error check will not be made on other than the above operations to sfr.

The sfr address and the data to be written are stored in registers or in the saddr area and each error check routine is called.

## (2) Standard libraries

To identify a standard library name without error check, character "\_" is prefixed to the symbol (function name). For a standard library with error check, the prefix character "-" must be changed as shown below depending on the type of error check and the character length of the symbol name must be reduced to 8 characters. (The same will apply when the symbol name length that can be recognized by the compiler is extended to 30 characters by the symbol name length specification option.)

Without error check	_function name
Stack check	_ @function name
Stack check + Pointer access	_ ?function name
Divide by zero	_ @function name
Divide by zero + Overflow	?_ function name
Pointer access	@? function name
Overflow	@_ function name

## (3) User-created functions

Character "\_" must be prefixed to the symbol (function name) with or without error check.

## CHAPTER 5. COMPILER OPTIONS

## 5.1 Types of Compiler Options

A compiler option gives the C compiler particular instructions for its operation. Compiler options may be broadly divided into the following 19 types:

Table 5-1. Types of Compiler Options

No.	Classification	Option name	Function
1	Option for processor type specification	-C	Specifies the processor type of the target device.
2	Option for object module file creation specification	-O	Specifies that an object module file is to be output.
3	Option for symbol name length specification	-S	Specifies that symbol name length is to be extended.
4	Option for symbol name upper-/lower-case specification	-CA	Specifies that symbol names in uppercase letters are not to be distinguished from those in lowercase letters.
5	Option for ROMable object file creation specification	-R	Specifies that a ROMable object module file is to be created.
6	Option for optimization specification	-Q	Specifies that optimization is to be performed for efficient object generation.
7	Option for debug information output specification	-G	Specifies that symbol information for debugging is to be output to the object module.
8	Option for execution-time error check specification	-L	Specifies that an error check library is to be added to the object module.



Table 5-1. Compiler Options (contd)

No.	Classification	Option name	Function
9	Options for preprocess list file creation specification	-P	Specifies that a preprocess list file is to be output.
		-K	Specifies the type of process required for the preprocess list is to be output.
10	Options for preprocessing specification	-D	Specifies that macro-definitions are to be executed in a similar manner to <b>#define</b> statements.
		-U	Specifies that macro-definitions are to be undefined in a similar manner to <b>#undef</b> statements.
		-I	Specifies that inclusion file(s) are to be input from a specified directory.
11	Options for assembler module file creation specification	-A	Specifies that an assembler source module file is to be output.
		-SA	Specifies that a C source is to be output to the added assembler source module file.
12	Options for error list file creation specification	-E	Specifies that an error list file is to be output.
		-SE	Specifies that a C source is to be output to the added error list file.
13	Option for cross-reference list creation specification	-X	Specifies that a cross-reference list is to be output.

Table 5-1. Compiler Options (contd)

No.	Classification	Option name	Function
14	Options for list format specification	-LW	Changes the number of characters per line of a list file.
		-LL	Changes the number of lines to be printed per page of a list.
		-LT	Changes the number of characters for tabulation.
		-LF	Specifies addition of a formfeed code to the end of a list.
15	Option for warning output specification	-W	Specifies the output of warning messages to the console.
16	Option for execution status output specification	-V	Specifies that the execution status is to be output to the console.
17	Option for parameter file input specification	-F	Specifies the input of a parameter file (input filename and options).
18	Option for temporary file creation path specification	-T	Specifies creation of a temporary file on a specified path.
19	Option for HELP message output specification	--	Specifies the output of HELP messages to the console.

The above table outlines the types of compiler options. The description format, function, and usage of each of these options are detailed in Section 5.4 below. APPENDIX C also contains a summary of the compiler options detailed in Section 5.4. When using any of these options, use of APPENDIX C is recommended for quick reference.

## 5.2 How to Specify Compiler Options

Compiler options can be specified in either of the following two ways:

- (1) Specification in the start-up command line of the C compiler.
- (2) Specification within a parameter file

The above two methods of specifying compiler option(s) have been explained in Section 4.3, "How to Start Up the C Compiler".

### 5.3 Priority of Compiler Options

Table 5-2 shows the priority of compiler options when two or more options are specified at the same time.

Table 5-2. Priority of Compiler Options

	-NO	-G	-P	-NP	-D	-U	-A	-E	-X	--
-R	x									x
-Q	x									x
-QS	x	x								x
-QZ	x	x								x
-QE	x	x								x
-QJ	x	x								x
-G	x									x
-L	x									x
-K			△	x						x
-D						○				x
-U					○					x
-SA							x			x
-LW			△				△	△	△	x
-LL			△				△	△	△	x
-LT			△				△	△	△	x
-LF			△				△	△	△	x

"X" in the table indicates that the option in the left column becomes invalid if the option in the top column is specified at the same time.

Example: A>cc78k3 -c310 -e sample.c -no -r -g

In the above example, the -R and -G options become invalid.

"△" (triangle) in the table indicates that the option in the left column becomes invalid if the option in the top column is not specified at the same time.

Example: A>cc78k3 -c310 -e sample.c -p -k

In the above example, the -K option is valid because the -P option is specified at the same time.

"O" (circle) in the table indicates that if the option in the left column and the option in the top column are specified at the same time, the last specified option takes precedence over the preceding option.

Example: A>cc78k3 -c310 -e sample.c -utest -dtest=1

In the above example, the -D option takes precedence over the -U option which has been described before -D. In this case, the -U option becomes invalid.

With two options contradicting each other such as -O and -NO and -G and -NG, whichever you specified later will take precedence over the other preceding option.

Example: A>cc78k3 -c310 -e sample.c -o -no

In the above example, the -NO option takes precedence over the -O option which has been described before -NO. In this case, the -O option becomes invalid.

Compiler options not listed in Table 5-2 are not affected by any other options. However, when the "--" option (for HELP message output specification) is specified, all the other compiler option specifications become invalid.

#### 5.4 Description of Each Compiler Option

A detailed description of each compiler option is provided in this section. Examples shown in the EXAMPLES column are those prepared by using the CC78K3 C compiler package. Compiler options may be described in either uppercase or lowercase letters. In the following examples of the respective options, compiler options are all shown in lowercase letters.

---

**-C****Processor type specification**

---

**(1) Processor type specification (-C)**

Description format: -C processor-type

Default assumption: This option cannot be omitted.

**FUNCTION**

The -C option specifies the processor type of the target device, C source programs for which are subject to compilation.

**USE**

This option must always be specified. The C compiler performs translation of source programs for the specified target device and generates object code corresponding to the target device.

**EXPLANATION**

The target devices that can be specified with the -C option and their corresponding processor types are listed below in the table for each 78K series.

&lt;With 78K/0&gt;

Table 5-3. Processor Types (78K/0 Series)

Target device name	Processor type
uPD78001	-C001
uPD78002	-C002
uPD78011	-C011
uPD78012	-C012
uPD78013	-C013
uPD78014, uPD78P014	-C014
uPD78022	-C022
uPD78023	-C023
uPD78024, uPD78P024	-C024
uPD78042	-C042
uPD78043	-C043
uPD78044, uPD78P044	-C044

---

-C

---

Processor type specification

---

&lt;With 78K/II&gt;

Table 5-4. Processor Types (78K/II Series)

Target device name	Processor type
uPD78210	-C210
uPD78212	-C212
uPD78213	-C213
uPD78214, uPD78P214	-C214
uPD78217A	-C217A
uPD78218A, uPD78P218A	-C218A
uPD78220	-C220
uPD78224, uPD78P224	-C224
uPD78233	-C233
uPD78234	-C234
uPD78237	-C237
uPD78238, uPD78P238	-C238
uPD78243	-C243
uPD78244	-C244

&lt;With 78K/III&gt;

Table 5-5. Processor Types (78K/III Series)

Target device name	Processor type
uPD78310	-C310
uPD78312, uPD78P312	-C312
uPD78310A	-C310A
uPD78312A, uPD78P312A	-C312A
uPD78320	-C320
uPD78322, uPD78P322	-C322
uPD78323	-C323
uPD78324, uPD78P324	-C324
uPD78327	-C327
uPD78328, uPD78P328	-C328
uPD78330	-C330
uPD78334, uPD78P334	-C334
uPD78350, uPD78P352	-C350

**NOTE**

- o The -C option cannot be omitted from specification in the command line except when the following directive description is contained in the C source.

```
#pragma pc (processor-type)
```



-C

## Processor type specification

- o If the target device specified in the command line is different from that specified in the C source, the target device specification in the command line will take precedence over the specification in the C source.

EXAMPLES

- o To specify target device uPD78310 in the command line with this option

```
A>cc78k3 -c310 sampleYprime.c
```

```
78K/111 Series C Compiler Vx.xx [xx xxx xx]
Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype
SAMPLEYPRIME.C(20) : W745 Expected function prototype
SAMPLEYPRIME.C(26) : W622 No return value
SAMPLEYPRIME.C(37) : W622 No return value
SAMPLEYPRIME.C(44) : W622 No return value
Compilation complete, 0 error(s) and 5 warning(s) found.
```

- o To specify target device in C source and start up C compiler

```
#pragma      pc(310)
#define TRUE  1
#define FALSE 0
#define SIZE  200

char  mark[SIZE+1];

main()
{
    int i, prime, k, count;
    ;
}
```

```
A>cc78k3 sampleYprime.c
```

```
78K/111 Series C Compiler Vx.xx [xx xxx xx]
Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype
SAMPLEYPRIME.C(20) : W745 Expected function prototype
SAMPLEYPRIME.C(26) : W622 No return value
SAMPLEYPRIME.C(37) : W622 No return value
SAMPLEYPRIME.C(44) : W622 No return value
Compilation complete, 0 error(s) and 5 warning(s) found.
```

In this case, the processor type specification in the command line may be omitted.

---

-C

---

Processor type specification

---

- o To specify target device different from that specified in C source in command line and start up C compiler

```
#pragma      pc(320)
#define TRUE  1
#define FALSE 0
#define SIZE  200

char  mark[SIZE+1];

main()
{
    int i, prime, k, count;
    ;
}
```

A>cc78k3 -c310 sampleYprime.c

78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx

```
SAMPLEYPRIME.C(1) : W832 Duplicated chip specifier
SAMPLEYPRIME.C(18) : W745 Expected function prototype
SAMPLEYPRIME.C(20) : W745 Expected function prototype
SAMPLEYPRIME.C(26) : W622 No return value
SAMPLEYPRIME.C(37) : W622 No return value
SAMPLEYPRIME.C(44) : W622 No return value
Compilation complete,      0 error(s) and      6 warning(s) found.
```

In this case, the target device specified in the command line takes precedence over that specified in the C source.

---

**-O/-NO**

**Object module file  
creation specification**

---

(2) Object module file creation specification (-O/-NO)

Description format:	-O [output-filename]
	or
	-NO
Default assumption:	-O (input-filename.REL)

#### FUNCTION

- o The -O option specifies the output (creation) of an object module file. It also specifies the output destination or output filename of the object module file.
- o The -NO option specifies the non-generation of an output module file.

#### USE

- o Use the -O option to change the output destination or output filename of an object module file.
- o If the compilation of a source module is to be performed only to output an assembler source module file, use the -NO option. (This will reduce the translation time.)

#### EXPLANATION

- o If a compile error is found during a compile operation with the -O option specified, no object module file will be output by the C compiler.
- o If a drive name is omitted from the -O option specification, the object module file will be output to the current drive.
- o If both the -O and -NO options are specified at the same time, whichever you specified later will take precedence over the other preceding option.

---

**-O/-NO**

Object module file  
creation specification

---

EXAMPLE

- o When both -NO and -O options are specified at the same time

A>cc78k3 -c310 sampleYprime.c -no -o

78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx

SAMPLEYPRIME.C(18) : W745 Expected function prototype  
SAMPLEYPRIME.C(20) : W745 Expected function prototype  
SAMPLEYPRIME.C(26) : W622 No return value  
SAMPLEYPRIME.C(37) : W622 No return value  
SAMPLEYPRIME.C(44) : W622 No return value  
Compilation complete, 0 error(s) and 5 warning(s) found.

In this case, the compiler will ignore the -NO option and accept the -O option as valid.

---

**-S/-NS****Symbol name length  
specification**

---

**(3) Symbol name length specification (-S/-NS)**

Description format: -S or -NS Default assumption: -NS
--

**FUNCTION**

- o The -S option tells the C compiler to extend the length of each symbol name to a maximum of 30 characters.
- o The -NS option tells the C compiler not to extend the symbol name length.

**USE**

If the length of each symbol name is to be extended to eight or more characters, the number of characters that the compiler will recognize as a symbol can be extended by the -S option.

**EXPLANATION**

- o If the -S option is specified, the C compiler will recognize up to 30 characters as a symbol name and output symbol information to the object module file to be created by the C compiler.  
The symbol information to be output is up to 31 characters including "\_" prefixed to each symbol name.
- o If the -S option is not specified, the C compiler will recognize up to seven characters as a symbol name and output symbol information to the object module file.  
The symbol information to be output is up to eight characters including "\_" prefixed to each symbol name.

---

**-S/-NS****Symbol name length  
specification**

---

- o If the -NO option is specified at the same time with the -S option, the C compiler will recognize up to 30 characters as a symbol name but the symbol information to be output is up to eight characters including "\_" prefixed to each symbol name just the same as when the -S option is omitted.
- o If both the -S and -NS options are specified at the same time, whichever you specified later will take precedence over the other preceding option.

**EXAMPLES**

- o To compile source program with -S option specified

```
78K/III Series C Compiler Vx.xx [xx xxx xx]
  Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype
SAMPLEYPRIME.C(20) : W745 Expected function prototype
SAMPLEYPRIME.C(26) : W622 No return value
SAMPLEYPRIME.C(37) : W622 No return value
SAMPLEYPRIME.C(44) : W622 No return value
Compilation complete,    0 error(s) and    5 warning(s) found.
```

---

-S/-NS

---

Symbol name length  
specification

---

o When PRIME.ASM file is referenced

```

: 78K/III Series C Compiler Vx.xx Assembler Source
:                                     Date:xx xxx xxxx Time:xx:xx:xx

```

```

: Command      : -c310 sampleYprime.c -a -s
: In-file      : SAMPLEYPRIME.C
: Asm-file     : PRIME.ASM
: Para-file    :

```

```

$PROCESSOR(310)
$NODEBUG

```

```

NAME      PRIME
EXTRN     @@isrem
PUBLIC    _mark
PUBLIC    _main
PUBLIC    _printf
PUBLIC    _putchar

```

```

@@CODE    CSEG
: line    5
: line    8
_main:
    push    hl
    movw    ax, sp
    subw    ax, #08H
    movw    hl, ax
    movw    sp, ax
: line    11
    movw    ax, #00H ; 0
    mov     [hl+1], a      ; count
    xch     a, x

```

```

    ;

```

---

**-CA/-NCA****Symbol name upper-/lower-  
case specification**

---

(4) Symbol name upper-/lower-case specification (-CA/-NCA)

Description format:	-CA
	or
	-NCA
Default assumption:	-NCA

### FUNCTION

- o The -CA option tells the C compiler not to distinguish between symbol names written in uppercase letters and those in lowercase letters.
- o The -NCA option tells the C compiler that a distinction need not be made between symbol names written in uppercase letters and those in lowercase letters.

### USE

Use the -CA option if no distinction need to be made between symbol names written in uppercase letters and those in lowercase letters.

### EXPLANATION

- o If the -CA option is specified, the C compiler will convert lowercase letters in symbol names to uppercase equivalents and output symbol information to the object module file to be created by the C compiler.
- o If the -CA option is not specified, the C compiler will not convert lowercase letters in symbol names to uppercase equivalents and output symbol information to the object module file.
- o When specifying the -CA or -NCA option, libraries must be changed according to the option, for which see Subsection 2.1.2, Library files in Chapter 2.



---

**-CA/-NCA****Symbol name upper-/lower-  
case specification**

---

- o If both the -CA and -NCA options are specified at the same time, whichever you specified later will take precedence over the other preceding option.
- o If the -NO option is specified at the same time with the -CA option, the -CA option will be ignored.

#### EXAMPLES

- o To compile source program with -CA option specified

```
A>cc78k3 -c310 sampleYprime.c -a -ca
```

```
78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype  
SAMPLEYPRIME.C(20) : W745 Expected function prototype  
SAMPLEYPRIME.C(26) : W622 No return value  
SAMPLEYPRIME.C(37) : W622 No return value  
SAMPLEYPRIME.C(44) : W622 No return value  
Compilation complete, 0 error(s) and 5 warning(s) found.
```

-CA/-NCA

Symbol name upper-/lower-  
case specification

o When PRIME.ASM file is referenced

```
;78K/III Series C Compiler Vx.xx Assembler Source
;                                     Date:xx xxx xxxx Time:xx:xx
;
```

```
; Command   : -c310 sampleYprime.c -a -ca
; In-file    : SAMPLEYPRIME.C
; Asm-file   : PRIME.ASM
; Para-file  :
```

```
$PROCESSOR(310)
$NODEBUG
```

```
NAME      PRIME
EXTRN     @@isrem
PUBLIC    _MARK
PUBLIC    _MAIN
PUBLIC    _PRINTF
PUBLIC    _PUTCHAR
```

```
@@CODE    CSEG
; line     5
; line     8
_MAIN:
    push    hl
    movw    ax, sp
    subw    ax, #08H
    movw    hl, ax
    movw    sp, ax
; line     11
    movw    ax, #00H ; 0
    mov     [hl+1], a ; COUNT
    xch     a, x
    ;
```

---

**-R/-NR****ROMable object file  
creation specification**

---

(5) ROMable object file creation specification (-R/-NR)

Description format:	-R
	or
	-NR
Default assumption:	-R

#### FUNCTION

- o The -R option tells the C compiler to create a ROMable object module file.
- o The -NR option tells the C compiler not to create a ROMable object module file.

#### USE

Use the -R option if you wish to make the output object module file ROMable (capable of being encoded into a ROM chip).

#### EXPLANATION

- o Because the default assumption of this ROMable object file creation specification is -R, the -R option may be omitted. If you wish to invalidate the -R option, specify the -NR option.
- o If neither an object module file nor an assembler source source module file is to be output, the -R option specification will become invalid.
- o If both the -R and -NR options are specified at the same time, whichever you specified later will take precedence over the other preceding option.

---

**-R/-NR****ROMable object file  
creation specification**

---

**EXAMPLES**

- o To compile source program with -R option specified

```
A>cc78k3 -c310 sampleYprime.c -a -r
```

```
78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype  
SAMPLEYPRIME.C(20) : W745 Expected function prototype  
SAMPLEYPRIME.C(26) : W622 No return value  
SAMPLEYPRIME.C(37) : W622 No return value  
SAMPLEYPRIME.C(44) : W622 No return value  
Compilation complete,      0 error(s) and      5 warning(s) found.
```

---

-R/-NRROMable object file  
creation specification

---

o When PRIME.ASM file is referenced

```

; 78K/III Series C Compiler Vx.xx Assembler Source
;                                     Date:xx xxx xxxx Time:xx:xx:
;
; Command   : -c310 sampleYprime.c -a -r
; In-file   : SAMPLEYPRIME.C
; Asm-file  : PRIME.ASM
; Para-file :
$PROCESSOR(310)
$NODEBUG

        NAME      PRIME
        EXTRN     @@isrem
        ;

        movw      ax,hl
        addw      ax,#02H
        movw      sp,ax
        pop       hl
        ret

@@CNST  CSEG
L0012:  DB        '%6d'
        DB        00H
L0018:  DB        0AH
        DB        '%d primes found.'
        DB        00H

@@R_DATA CSEG          /*segment for initialize data*/
        DB        (201)

@@DATA  DSEG
_mark:  DS          (201)  /*segment for temporary data area*/
        END

```

---

-R/-NRROMable object file  
creation specification

---

- o To compile the source program with -NR option specified

```
A>cc78k3 -c310 sampleYprime.c -a -nr
```

```
78K/III Series C Compiler Vx.xx [xx xxx xx]
  Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype
SAMPLEYPRIME.C(20) : W745 Expected function prototype
SAMPLEYPRIME.C(26) : W622 No return value
SAMPLEYPRIME.C(37) : W622 No return value
SAMPLEYPRIME.C(44) : W622 No return value
Compilation complete,      0 error(s) and      5 warning(s) found.
```

- o When PRIME.ASM file is referenced

```
; 78K/III Series C Compiler Vx.xx Assembler Source
;                                                    Date:xx xxx xxxx Time:xx:xx:xx
```

```
; Command      : -c310 sampleYprime.c -a -nr
; In-file       : SAMPLEYPRIME.C
; Asm-file      : PRIME.ASM
; Para-file     :
```

```
$PROCESSOR(310)
$NODEBUG
```

```
NAME      PRIME
EXTRN     @@isrem
```

```
;
```

```
movw      ax,hl
addw      ax,#02H
movw      sp,ax
pop        hl
ret
```

```
@@CNST    CSEG
L0012:    DB      '%6d'
          DB      00H
L0018:    DB      0AH
          DB      '%d primes found.'
          DB      00H
```

```
@@DATA    DSEG      /*segment for initialize data*/
_mark:    DB      (201)
          END
```

```

Description format:  -Q[optimization-type]
                    or
                    -NQ
Default assumption:  -NQ

```

5-24

-Q/-NQ

Optimization process  
specification

Table 5-6. Optimization Types

Type of optimization	Description of process	Series name		
		0	II	III
Omitted	Only R is assumed to have been specified. (With 78K/II, Z and R are assumed.)	o	o	o
S	Optimize with emphasis on the object execution speed.	o	o	o
Z	Optimize with emphasis on the object size.	o	o	o
U	Interpret char without qualifier as <b>unsigned</b> .	o	o	o
C	Execute operations on char data without sign extension.	o	o	o
R	Allocate register variables to the saddr area in addition to the register area.	o	o	o
L	Set the preprocessing of a function for library call.	o	o	o
E	Executes processes such as deletion of common partial expressions and copies.			o
J	Optimize jump instructions.			o
X	Optimize with maximum processes. Same as when -Q, Z, C, R, E, and J are specified.			o

o: Applicable; Blank: Not applicable

- o If the -Q option is omitted, the C compiler will assume -NQ and will not perform optimization.
- o If neither an object module file nor an assembler source module file is to be output, the -Q option specification will become invalid.
- o If both the -Q and -NQ options are specified at the same time, whichever you specified later will take precedence over the other preceding option.
- o If two or more -Q options are specified at the same time, the last specified -Q option will take precedence over the other -Q options.



---

**-Q/-NQ****Optimization process  
specification**

---

**EXAMPLES**

- o To compile the source program with -Q option specified with optimization type "u" (interpret char data without qualifier as unsigned)

```
A>cc78k3 -c310 sampleYprime.c -qu
```

```
78K/III Series C Compiler Vx.xx [xx xxx xx]
Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype
SAMPLEYPRIME.C(20) : W745 Expected function prototype
SAMPLEYPRIME.C(26) : W622 No return value
SAMPLEYPRIME.C(37) : W622 No return value
SAMPLEYPRIME.C(44) : W622 No return value
Compilation complete, 0 error(s) and 5 warning(s) found.
```

- o When both -QS and -QR options are specified at the same time

```
A>cc78k3 -c310 sampleYprime.c -qc -qr
```

```
78K/III Series C Compiler Vx.xx [xx xxx xx]
Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype
SAMPLEYPRIME.C(20) : W745 Expected function prototype
SAMPLEYPRIME.C(26) : W622 No return value
SAMPLEYPRIME.C(37) : W622 No return value
SAMPLEYPRIME.C(44) : W622 No return value
Compilation complete, 0 error(s) and 5 warning(s) found.
```

In this case, the compiler ignores the -QS option and accepts the -QR as valid.

To make both -QS and -QR valid, enter the start-up command line as follows:

```
A>cc78k3 -c310 sampleYprime.c -qcr
```

---

**-G/-NG**

---

Debug information  
output specification

---

**(7) Debug output information specification (-G/-NG)**

Description format: -G
or
-NG
Default assumption: -NG

**FUNCTION**

- o The -G option specifies the addition of symbol information for debugging to the object module file to be created by the C compiler.
- o The -NG option specifies the suppression of debug information output to the object module file.

**USE**

If the -G option is not specified, the line number and symbol information required for a symbol table file which becomes an input file to the screen debugger will not be output. Therefore, when performing symbolic debugging, compile all the modules to be linked by specifying the -G option.

**EXPLANATION**

- o If both the -G option and -NG options are specified at the same time, whichever you specified later will take precedence over the other preceding option.
- o If neither an object module file nor an assembler source module file is to be output, the -G option specification will become invalid.

---

**-G/-NG**

Debug information  
output specification

---

#### EXAMPLES

- o To compile the source program with -G option specified

```
A>cc78k3 -c310 sampleYprime.c -g
```

```
78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype  
SAMPLEYPRIME.C(20) : W745 Expected function prototype  
SAMPLEYPRIME.C(26) : W622 No return value  
SAMPLEYPRIME.C(37) : W622 No return value  
SAMPLEYPRIME.C(44) : W622 No return value  
Compilation complete,      0 error(s) and      5 warning(s) found.
```

---

**-L/-NL**

**Execution-time error  
check specification**

---

**(8) Execution-time error check specification (-L/-NL)**

Description format: -L[error-check-type]

or

-NL

Default assumption: -NL

**FUNCTION**

- o The -L option specifies the addition of an execution-time error check library to the object module file to be created by the C compiler.
- o The -NL option specifies the non-addition of an execution-time error check library to the object object module file.

**USE**

Use the -L option to specify check items according to the purpose of the error check.

**EXPLANATION**

- o Error check types that can be specified with the -L options are as shown in Table 5-7.
- o Two or more error check types may be specified with the -L option.

-L/-NL	Execution-time error check specification
--------	---

Table 5-7. Error Check Types

Error check type	Description
Omitted	All types (2, P, S, Z, and D) are assumed to have been specified.
1	Execute an overflow check on multiplication and division.
2	Execute an overflow check on all operations including 1.
P	Check for an illegal address access by using a pointer.
S	Check if the stack area to be used has been reserved.
Z	Check for division by zero.
D	Check for an illegal read- or write-access to the sfr area.

- o When the source debugger is activated, the C compiler may perform operations not intended by the user because of various errors. For this reason, when starting up the C compiler, specify the -L option to generate error-checking object code in addition to ordinary object code.
- o If neither an object module file nor an assembler source module file is to be output, the -L option specification will become invalid.
- o If both the -L and -NL options are specified at the same time, whichever you specified later will take precedence over the other preceding option.

---

-L/-NL

Execution-time error  
check specification

---

EXAMPLE

- o To specify stack overflow check and divide-by-zero check

```
A>cc78k3 -c310 sampleYprime.c -lsz
```

```
78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype  
SAMPLEYPRIME.C(20) : W745 Expected function prototype  
SAMPLEYPRIME.C(26) : W622 No return value  
SAMPLEYPRIME.C(37) : W622 No return value  
SAMPLEYPRIME.C(44) : W622 No return value  
Compilation complete, 0 error(s) and 5 warning(s) found.
```

---

**-P/-NP**

Preprocess list file  
creation specification

---

(9) Preprocess list file creation specification

(-P/-NP, -K/-NK)

Description format: -P [output-filename]

or

-NP

Default assumption: -NP

FUNCTION

- o The -P option specifies the output destination or output filename of the preprocess list file to be output by the C compiler.
- o The -NP option specifies the non-generation of a preprocess list file.

USE

- o Use the -P option to change the output destination or output filename of a preprocess list file.
- o If the compilation of a source module is to be performed only to output an assembler source module file, use the -NP option. (This will reduce the translation time.)

EXPLANATION

- o If an output filename is omitted from the -P option specification, "input filename.PPL" will be assumed as the filename of the preprocess list file.
- o If a drive name is omitted from the -P option specification, the preprocess list file will be output to the current drive.
- o If both the -P and -NP options are specified at the same time, whichever you specified later will take precedence over the other preceding option.

---

-P/-NP

Preprocess list file  
creation specification

---

EXAMPLE

- o To output preprocess list file "SAMPLE.PPL"

A>cc78k3 -c310 sampleYprime.c -psample.ppl

78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx

SAMPLEYPRIME.C(18) : W745 Expected function prototype  
SAMPLEYPRIME.C(20) : W745 Expected function prototype  
SAMPLEYPRIME.C(26) : W622 No return value  
SAMPLEYPRIME.C(37) : W622 No return value  
SAMPLEYPRIME.C(44) : W622 No return value  
Compilation complete, \_ 0 error(s) and 5 warning(s) found.



---

-K/-NK

Preprocess list file  
creation specification

---

Description format: -K [process-type]

or

-NK

Default assumption: -KFLN

#### FUNCTION

- o The -K option specifies special process(es) required for the output preprocess list file specified by the -P option.
- o The -NK option tells the C compiler that no special processes other than by default assumption are required for the preprocess list file.

#### USE

When a preprocess list is to be output, use the -K option to specify special processes such as comment deletion, reference of macroexpansion, etc.

#### EXPLANATION

- o Process types that can be specified with the -K option are as shown in Table 5-8 below.
- o Two or more process types may be specified with the -K option.

---

**-K/-NK****Preprocess list file  
creation specification**

---

Table 5-8. Process Types with -K Option

Process type	Description
Omitted	F, L, and N are assumed to have been specified.
C	Delete comments.
D	Expand <b>#define</b> directives.
F	Execute conditional compilation of <b>#if</b> , <b>#ifdef</b> , and <b>#ifndef</b> directives.
I	Expand <b>#include</b> directives.
L	Process <b>#line</b> directives.
N	Execute line number and paging process.

- o If the -K option is not specified together with the -P option, the -K option specification will become invalid.
- o If both the -K and -NK options are specified at the same time, whichever you specified later will take precedence over the other preceding option.
- o If two or more -K options are specified at the same time, the last specified -K option will take precedence over the other K-options.

---

-K/-NK

Preprocess list file  
creation specification

---

### EXAMPLES

- o To output preprocess list file "SAMPLE.PPL" with comment deletion and line number/paging process specified

```
A>cc78k3 -c310 sampleYprime.c -p -kcn
```

```
78K/III Series C Compiler Vx.xx [xx xxx xx]
Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype
SAMPLEYPRIME.C(20) : W745 Expected function prototype
SAMPLEYPRIME.C(26) : W622 No return value
SAMPLEYPRIME.C(37) : W622 No return value
SAMPLEYPRIME.C(44) : W622 No return value
Compilation complete, 0 error(s) and 5 warning(s) found.
```

- o When PRIME.PPL file is referenced

```
/*
78K/III Series C Compiler Vx.xx Preprocess List
Date:xx xxx xxxx Page: 1

Command : -c310 sampleYprime.c -p -kcn
In-file : SAMPLEYPRIME.C
PPL-file : PRIME.PPL
Para-file :
*/
```

```
1 : #define TRUE 1
2 : #define FALSE 0
3 : #define SIZE 200
4 :
5 : char mark[SIZE+1];
6 :
7 : main()
8 : {
9 :     int i, prime, k, count;
10 :
11 :     count = 0;
12 :     :
```

---

-K/-NK

---

Preprocess list file  
creation specification

---

- o When -both -KN and -KC options are specified

```
A>cc78k3 -c310 sampleYprime.c -p -kn -kc
```

```
78K/III Series C Compiler Vx.xx [xx xxx xx]
Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype
SAMPLEYPRIME.C(20) : W745 Expected function prototype
SAMPLEYPRIME.C(26) : W622 No return value
SAMPLEYPRIME.C(37) : W622 No return value
SAMPLEYPRIME.C(44) : W622 No return value
Compilation complete,      0 error(s) and      5 warning(s) found.
```

- o When PRIME.PPL file is referenced

```
/*
78K/III Series C Compiler Vx.xx Preprocess List      Date:xx xxx xxxx Page:
```

```
Command   : -c310 sampleYprime.c -p -kn -kc
In-file   : SAMPLEYPRIME.C
PPL-file  : PRIME.PPL
Para-file :
*/
```

```
#define TRUE      1
#define FALSE     0
#define SIZE      200

char  mark[SIZE+1];

main()
{
    int i, prime, k, count;

    count = 0;

    for ( i = 0 ; i <= SIZE ; i++)
        mark[i] = TRUE;
    for ( i = 0 ; i <= SIZE ; i++) {
        if (mark[i]) {
            prime = i + i + 3;
            printf("%6d", prime);
            count++;
            if((count%8) == 0) putchar('\n');
            for ( k = i + prime ; k <= SIZE ; k += prime)
                mark[k] = FALSE;
        }
    }
}
```

In this case, the Compiler ignores -KN option and accepts -KC option as valid.

(10) Preprocessing specification (-D/-ND, -U/-NU, -I)

Description format: -D macro-name=[definition-name] .  
[ ,macro-name[=definition-name] ...  
or  
-ND

Default assumption: Accepts only macro definitions  
within C source module file as  
valid.

## FUNCTION

- o The `-D` option tells the C compiler to execute macro-definitions just the same as `#define` directive statements in the C source.
- o The `-ND` option invalidates the `-D` option.

USE

Use the -D option if you wish to substitute all specific constants with macro names.

## EXPLANATION

- o Two or more macrodefinitions may be specified by delimiting each definition with a "," (comma).  
No space (blank) is allowed before or after "=" and ",".
- o If a definition name is omitted from the -D option specification, "1" will be assumed as the definition name.
- o If the same macro name is specified by both the -D and -U options, whichever you specified later will take precedence over the other preceding option.
- o If both the -D and -ND options are specified at the same time, whichever you specified later will take precedence over the other preceding option.

---

-D/-ND

Preprocessing specification

---

EXAMPLE

o To compile source program with -D option specified

A>cc78k3 -c310 sampleYprime.c -dtest.time=10

78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx

SAMPLEYPRIME.C(18) : W745 Expected function prototype  
SAMPLEYPRIME.C(20) : W745 Expected function prototype  
SAMPLEYPRIME.C(26) : W622 No return value  
SAMPLEYPRIME.C(37) : W622 No return value  
SAMPLEYPRIME.C(44) : W622 No return value  
Compilation complete. 0 error(s) and 5 warning(s) found.

Description format: -U macro-name[,macro-name] ...

or

-NU

Default assumption: Accepts macro definitions specified  
by -D option as valid.

### FUNCTION

- o The -U option tells the C compiler to invalidate (undefine) macro definitions just the same as #undef directive statements in the C source.
- o The -NU option tells the C compiler not to undefine the macrodefinitions defined by the -D option.

### USE

Use the -U option if you wish to undefine the macro names defined by the -D option.

### EXPLANATION

- o Two or more macrodefinitions may be undefined by delimiting each definition with a "," (comma).  
No space (blank) is allowed before or after ",".
- o Only the macrodefinitions defined by the -D option can be undefined by the -U option. The macro names defined by #define directive statements in the C source file and the system macro names that the C compiler has cannot be invalidated with the -U option.
- o If the same macro name is specified by both the -D and -U options, whichever you specified later will take precedence over the other preceding option.
- o If both the -U and -NU options are specified at the same time, whichever you specified later will take precedence over the other preceding option.

---

**-U/-NU**

---

**Preprocessing specification**

---

**EXAMPLE**

- o When same macro name is specified by -D and -U options

```
A>cc78k3 -c310 sampleYprime.c -dtest -utest -
```

```
78K/111 Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype  
SAMPLEYPRIME.C(20) : W745 Expected function prototype  
SAMPLEYPRIME.C(26) : W622 No return value  
SAMPLEYPRIME.C(37) : W622 No return value  
SAMPLEYPRIME.C(44) : W622 No return value  
Compilation complete, 0 error(s) and 5 warning(s) found.
```



---

**-I****Preprocessing specification**

---

Description format: <code>-I directory [,directory] ...</code>
Default assumption: Directory containing directory source file specified by environ- ment variable <code>INC78Kn</code> ( $n=0,2,3$ )

#### FUNCTION

- o The `-I` option tells the C compiler to input the Inclusion files specified by the `#include` directive statement in the C source from a specified directory.

#### USE

Use the `-I` option to search a directory or directories for Inclusion files.

#### EXPLANATION

- o Two or more directories may be specified by delimiting each directory with a `","` (comma). No space (blank) is allowed before or after `","`.
- o If two or more directories are specified following the `-I` option or if two or more `-I` options are specified at the same time, the C compiler will search directories for the file(s) specified by `#include` in the order of their specification in the command line. Then, the compiler will search directories in the same order as the default assumption.

---

-IPreprocessing specification

---

EXAMPLE

- o To compile source program with -I option specified

A>cc78k3 -c310 sampleYprime.c -ib:.,b:Ysample

78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx

SAMPLEYPRIME.C(18) : W745 Expected function prototype  
SAMPLEYPRIME.C(20) : W745 Expected function prototype  
SAMPLEYPRIME.C(26) : W622 No return value  
SAMPLEYPRIME.C(37) : W622 No return value  
SAMPLEYPRIME.C(44) : W622 No return value  
Compilation complete, 0 error(s) and 5 warning(s) found.

---

**-A/-NA****Assembler source module  
file creation specification**

---

(11) Assembler source module file creation specification  
(-A/-NA, -SA/-NSA)

Description format: -A [output-filename]

or

-NA

Default assumption: -NA

### FUNCTION

- o The -A option specifies the output of an assembler source module file. It also specifies the output destination or output filename of the assembler source module file.
- o The -NA option specifies the non-generation of an assembler source module file.

### USE

Use the -A option to change the output destination or output filename of an assembler source module file.

### EXPLANATION

- o As the output filename of an assembler source module file, either a disk type or device type filename can be specified.
- o If an output filename is omitted from the -A option specification, the C compiler will assume that "input filename.ASM" has been specified as the output filename.
- o If a drive name is omitted from the -A option specification, the assembler source module file will be output to the current drive.
- o If both the -A and -SA options are specified at the same time, the -SA option will be ignored.

---

**-A/-NA**

**Assembler source module  
file creation specification**

---

- o If both the -A and -NA options are specified at the same time, whichever you specified later will take precedence over the other preceding option.

#### EXAMPLES

- o To create assembler source module file named "SAMPLE.ASM"

```
A>cc78k3 -c310 sampleYprime.c -asample.asm
```

```
78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype  
SAMPLEYPRIME.C(20) : W745 Expected function prototype  
SAMPLEYPRIME.C(26) : W622 No return value  
SAMPLEYPRIME.C(37) : W622 No return value  
SAMPLEYPRIME.C(44) : W622 No return value  
Compilation complete,    0 error(s) and    5 warning(s) found.
```

---

-A/-NAAssembler source module  
file creation specification

---

- o To output assembler source module file onto printer

A>cc78k3 -c310 sampleYprime.c -aprn

78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx

SAMPLEYPRIME.C(18) : W745 Expected function prototype  
SAMPLEYPRIME.C(20) : W745 Expected function prototype  
SAMPLEYPRIME.C(26) : W622 No return value  
SAMPLEYPRIME.C(37) : W622 No return value  
SAMPLEYPRIME.C(44) : W622 No return value  
Compilation complete, 0 error(s) and 5 warning(s) found.

---

**-SA/-NSA****Assembler source module  
file creation specification**

---

Description format: -SA [output-filename]

or

-NSA

Default assumption: -NSA

### FUNCTION

- o The -SA option specifies the output of an assembler source module file with the C source source program added as comments to it. It also specifies the output destination or output filename of the assembler source module file.
- o The -NSA option specifies the non-addition of the C source source module file to the assembler source module file to be output.

### USE

Use the -SA option if you wish to output an assembler source module file together with the C source module file.

### EXPLANATION

- o As the output filename of an assembler source module file, either a disk type or device type filename can be specified.
- o If an output filename is omitted from the -SA option specification, the C compiler will assume that "input filename.ASM" has been specified as the output filename.
- o If a drive name is omitted from the -SA option specification, the assembler source module file will be output to the current drive.

---

**-SA/-NSA****Assembler source module  
file creation specification**

---

- o If both the -A and -SA options are specified at the same time, the -SA option will be ignored.
- o If both the -SA and -NSA options are specified at the same time, whichever you specified later will take precedence over the other preceding option.

**EXAMPLES**

- o To create assembler source module file with -SA option

```
A>cc78k3 -c310 sampleYprime.c -sa
```

```
78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype  
SAMPLEYPRIME.C(20) : W745 Expected function prototype  
SAMPLEYPRIME.C(26) : W622 No return value  
SAMPLEYPRIME.C(37) : W622 No return value  
SAMPLEYPRIME.C(44) : W622 No return value  
Compilation complete,      0 error(s) and      5 warning(s) found.
```

---

-SA/-NSAAssembler source module  
file creation specification

---

o When PRIME.ASM file is referenced

```

; 78K/111 Series C Compiler Vx.xx Assembler Source
;                                     Date:xx xxx xxxx Time:xx:xx:xx
;
; Command      : -c310 sampleYprime.c -sa
; In-file      : SAMPLEYPRIME.C
; Asm-file     : PRIME.ASM
; Para-file    :

```

```

$PROCESSOR(310)
$NODEBUG

```

```

NAME      PRIME
EXTRN     @@isrem
PUBLIC    _mark
PUBLIC    _main
PUBLIC    _printf
PUBLIC    _putchar

```

```

@@CODE    CSEG
; line     1 : #define TRUE      1
; line     2 : #define FALSE    0
; line     3 : #define SIZE     200
; line     4 :
; line     5 : char      mark[SIZE+1];
; line     6 :
; line     7 : main()
; line     8 : {
_main:
    push    hl
    movw    ax, sp
    subw    ax, #08H
    ;

```

In this example, C source has been added as comments.



---

**-E/-NE**

---

**Error list file  
creation specification**

---

(12) Error list file creation specification (-E/-NE,  
-SE/-NSE)

Description format: -E [output-filename]

or

-NE

Default assumption: -NE

### FUNCTION

- o The -E option specifies the output of an error list file. It also specifies the output destination or output filename of the error list file.
- o The -NE option specifies the non-generation of an error list file.

### USE

Use the -E option to change the output destination or output filename of an error list file.

### EXPLANATION

- o As the output filename of an error list file, either a disk type or device type filename can be specified.
- o If an output filename is omitted from the -E option specification, the C compiler will assume that "input filename.ECC" has been specified as the output filename.
- o If a drive name is omitted from the -E option specification, the error list file will be output to the current drive.
- o If the -W0 option is specified at the same time, no warning message will be output.

---

-E/-NE

Error list file  
creation specification

---

- o If both the -E and -NE options are specified at the same time, whichever you specified later will take precedence over the other preceding option.

#### EXAMPLES

- o To compile source program with -E option specified

```
A>cc78k3 -c310 sampleYprime.c -e
```

```
78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype  
SAMPLEYPRIME.C(20) : W745 Expected function prototype  
SAMPLEYPRIME.C(26) : W622 No return value  
SAMPLEYPRIME.C(37) : W622 No return value  
SAMPLEYPRIME.C(44) : W622 No return value  
Compilation complete,      0 error(s) and      5 warning(s) found.
```

- o When the error list file is referenced

```
SAMPLEYPRIME.C( 18) : W745 Expected function prototype  
SAMPLEYPRIME.C( 20) : W745 Expected function prototype  
SAMPLEYPRIME.C( 26) : W622 No return value  
SAMPLEYPRIME.C( 37) : W622 No return value  
SAMPLEYPRIME.C( 44) : W622 No return value
```

```
Compilation complete,      0 error(s) and      5 warning(s) found.
```

---

**-SE/-NSE****Error list file  
creation specification**

---

Description format: -SE [output-filename]

or

-NSE

Default assumption: -NSE

### FUNCTION

- o The -SE option specifies the output of an error list file with the C source source module file added to it. It also specifies the output destination or output filename of the error list file.
- o The -NSE option specifies the non-addition of the C source source module file to the error list file to be output.

### USE

Use the -SE option if you wish to output an error list file together with the C source module file.

### EXPLANATION

- o As the output filename of an error list file, either a disk type or device type filename can be specified.
- o If an output filename is omitted from the -SE option specification, the C compiler will assume that "input filename.CER" has been specified as the output filename.
- o If a drive name is omitted from the -SE option specification, the error list file will be output to the current drive.
- o No filename can be specified for any Inclusion file. If the file type of the inclusion file is H or C, an error list with file type "HER" or "CER", respectively, will be output. Otherwise, an error list file with file type "ER" will be output.

---

**-SE/-NSE****Error list file****creation specification**

---

- o If no error is found, no C source module file will be added. No error list file will be created for an Inclusion file.
- o If the -W0 option is specified at the same time, no warning message will be output.
- o If both the -SE and -NSE options are specified at the same time, whichever you specified later will take precedence over the other preceding option.

### EXAMPLES

- o To compile source program with -SE option specified

```
A>cc78k3 -c310 sampleYprime.c -se
```

```
78K/III Series C Compiler Vx.xx [xx xxx xx]
  Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype
SAMPLEYPRIME.C(20) : W745 Expected function prototype
SAMPLEYPRIME.C(26) : W622 No return value
SAMPLEYPRIME.C(37) : W622 No return value
SAMPLEYPRIME.C(44) : W622 No return value
Compilation complete,    0 error(s) and    5 warning(s) found.
```

---

-SE/-NSEError list file  
creation specification

---

o When PRIME.CER file is referenced

```

/*
78K/III Series C Compiler Vx.xx Error List           Date:xx xxx xxxx Time:xx:

Command      : -c310 sampleYprime.c -se
In-file      : SAMPLEYPRIME.C
Err-file     : PRIME.CER
Para-file    :
*/

#define TRUE      1
#define FALSE     0
#define SIZE      200

char    mark[SIZE+1];

main()
{
    int i, prime, k, count;

    count = 0;

    for ( i = 0 ; i <= SIZE ; i++)
        mark[i] = TRUE;
    for ( i = 0 ; i <= SIZE ; i++) {
        if (mark[i]) {
            prime = i + i + 3;
            printf("%6d", prime);
            count++;
        }
    }

    *** WARNING W745 Expected function prototype

/*
Compilation complete,      0 error(s) and      5 warning(s) found.
*/

```

In this example, C source has been added.

---

**-X/-NX**

Cross-reference list file  
creation specification

---

(13) Cross-reference list file creation specification  
(-X/-NX)

Description format: -X [output-filename]

or

-NX

Default assumption: -NX

#### FUNCTION

- o The -X option specifies the output of a cross-reference list. It also specifies the output destination or output filename of the cross-reference list file.
- o The -NX option specifies the non-generation of a cross-reference list.

#### USE

Use the -X option to change the output destination or output filename of a cross-reference list file.

#### EXPLANATION

- o As the output filename of a cross-reference list file, either a disk type or device type filename can be specified.
- o If an output filename is omitted from the -X option specification, the C compiler will assume that "input filename.XRF" has been specified as the output filename.
- o If both the -X and -NX options are specified at the same time, whichever you specified later will take precedence over the other preceding option.

---

**-X/-NX**

Cross-reference list file  
creation specification

---

EXAMPLE

- o To compile source program with -X option specified

A>cc78k3 -c310 sampleYprime.c -x

78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx

SAMPLEYPRIME.C(18) : W745 Expected function prototype  
SAMPLEYPRIME.C(20) : W745 Expected function prototype  
SAMPLEYPRIME.C(26) : W622 No return value  
SAMPLEYPRIME.C(37) : W622 No return value  
SAMPLEYPRIME.C(44) : W622 No return value  
Compilation complete, 0 error(s) and 5 warning(s) found.

---

-LW	List file format specification
-----	-----------------------------------

---

(14) List file format specification (-LW, -LL, -LT, and -LF)

Description format: -LW number-of-columns
Default assumption: -LW132 (or -LW80 for output to console)

#### FUNCTION

The -LW option specifies the number of print columns per line (PAGEWIDTH) of a list file.

#### USE

Use the -LW option to change the number of print columns per line of any of the various list files.

#### EXPLANATION

- o The number of print columns per line to be specified with the -LW option must be within the following value range without including a terminator (CR or LF):  
$$72 \leq \text{No. of print columns per line} \leq 132$$
However, when the list file is to be output to the console, up to 80 characters will be output.
- o If the number of columns is omitted, 132 columns per line will be assumed.
- o If no list file is specified, the -LW option specification will become invalid.



---

-LWList file format  
specification

---

EXAMPLES

- o To output cross-reference list file onto printer with -LW option omitted

```
A>cc78k3 -c310 sampleYprime.c -xprn
```

```
78K/III Series C Compiler Vx.xx [xx xxx xx]
Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype
SAMPLEYPRIME.C(20) : W745 Expected function prototype
SAMPLEYPRIME.C(26) : W622 No return value
SAMPLEYPRIME.C(37) : W622 No return value
SAMPLEYPRIME.C(44) : W622 No return value
Compilation complete,      0 error(s) and      5 warning(s) found.
```

- o When PRIME.XRF file is referenced

78K/III Series C Compiler V1.10 Cross reference List

Date: 5 Sep 1990 Page: 1

```
Command : -c310 sampleYprime.c -xprn
In-file : YSAMPLEYPRIME.C
Xref-file : PRIME.XRF
Para-file :
```

ATTRIB	MODIFY	TYPE	SYMBOL	DEFINE	REFERENCE
EXTERN	array	mark	5	14	16 22
EXTERN	func	main	7		
AUTO1	int	i	9	13	13 14 15 15 15 16 17 17
AUTO1	int	prime	9	17	18 21 21
AUTO1	int	k	9	21	21 21 22
AUTO1	int	count	9	11	19 20 25
EXTERN	func	printf	28	18	25
EXTERN	func	putchar	39	20	
PARAM	pointer	s	29	36	
PARAM	int	i	30	35	
AUTO1	int	j	32	35	
AUTO1	pointer	ss	33	36	
PARAM	char	c	40	43	
AUTO1	char	d	42	43	
	#define	TRUE	1	14	
	#define	FALSE	2	22	
	#define	SIZE	3	5	13 15 21

---

-LW

---

List file format  
specification

---

- o To output cross-reference list file with columns per line specified as 80

```
A>cc78k3 -c310 sampleYprime.c -x -lw80
```

```
78K/III Series C Compiler Vx.xx [xx xxx xx]
Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype
SAMPLEYPRIME.C(20) : W745 Expected function prototype
SAMPLEYPRIME.C(26) : W622 No return value
SAMPLEYPRIME.C(37) : W622 No return value
SAMPLEYPRIME.C(44) : W622 No return value
Compilation complete, 0 error(s) and 5 warning(s) found.
```

- o When PRIME.XRF is referenced

```
78K/III Series C Compiler Vx.xx Cross reference List Date:xx xxx xxxx Page:
```

```
Command : -c310 sampleYprime.c -x -lw80
In-file : SAMPLEYPRIME.C
Xref-file : PRIME.XRF
Para-file :
```

ATTRIB	MODIFY	TYPE	SYMBOL	DEFINE	REFERENCE			
EXTERN		array	mark	5	14	16	22	
EXTERN		func	main	7				
AUTO1		int	i	9	13	13	13	14
					15	15	15	16
					17	17	21	
AUTO1		int	prime	9	17	18	21	21
AUTO1		int	k	9	21	21	21	22
AUTO1		int	count	9	11	19	20	25
EXTERN		func	printf	28	18	25		
EXTERN		func	putchar	39	20			
PARAM		pointer	s	29	36			
PARAM		int	i	30	35			
AUTO1		int	j	32	35			
AUTO1		pointer	ss	33	36			
PARAM		char	c	40	43			
AUTO1		char	d	42	43			
		#define	TRUE	1	14			
		#define	FALSE	2	22			
		#define	SIZE	3	5	13	15	21

---

**-LL****List file format  
specification**

---

Description format: -LL number-of-lines

Default assumption: -LL66 (or no page ejection for  
output to console)

#### FUNCTION

The -LL option specifies the number of print lines per page (PAGELENGTH) of a list file.

#### USE

Use the -LL option to change the number of print lines per page of any of the various list files.

#### EXPLANATION

- o The number of print lines per page to be specified with the -LL option must be within the following value range:  
 $20 \leq \text{No. of print lines per page} \leq 65,535$
- o If the number of lines is specified as 0 (i.e., -LL0), no page ejection will be performed for the list file to be output to the console.
- o If the number of lines is omitted, 66 lines per page will be assumed.
- o If no list file is specified, the -LL option specification will become invalid.

---

-LLList file format  
specification

---

EXAMPLES

- o To output cross-reference list file with no. of lines per page specified as 20

A>cc78k3 -c310 sampleYprime.c -x -l20

78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx

SAMPLEYPRIME.C(18) : W745 Expected function prototype  
SAMPLEYPRIME.C(20) : W745 Expected function prototype  
SAMPLEYPRIME.C(26) : W622 No return value  
SAMPLEYPRIME.C(37) : W622 No return value  
SAMPLEYPRIME.C(44) : W622 No return value  
Compilation complete, 0 error(s) and 5 warning(s) found.

-LL

List file format  
specification

o When PRIME.XRF file is referenced

78K/III Series C Compiler Vx.xx Cross reference List

Date:xx xxx xxxx Page: 1

Command : -c310 sampleYprime.c -x -1120  
 In-file : SAMPLEYPRIME.C  
 Xref-file : PRIME.XRF  
 Para-file :

ATTRIB	MODIFY	TYPE	SYMBOL	DEFINE	REFERENCE			
EXTERN		array	mark	5	14	16	22	
EXTERN		func	main	7				
AUTO1		int	i	9	13	13	13	14
15	15	15	16	17	17			
					21			
AUTO1		int	prime	9	17	18	21	21

78K/III Series C Compiler Vx.xx Cross reference List

Date:xx xxx xxxx Page: 2

AUTO1	int	k	9	21	21	21	22
AUTO1	int	count	9	11	19	20	25
EXTERN	func	printf	28	18	25		
EXTERN	func	putchar	39	20			
PARAM	pointer	s	29	36			
PARAM	int	i	30	35			
AUTO1	int	j	32	35			
AUTO1	pointer	ss	33	36			
PARAM	char	c	40	43			
AUTO1	char	d	42	43			
	#define	TRUE	1	14			
	#define	FALSE	2	22			

78K/III Series C Compiler Vx.xx Cross reference List

Date:xx xxx xxxx Page: 3

#define	SIZE	3	5	13	15	21
---------	------	---	---	----	----	----

---

**-LT****List file format  
specification**

---

Description format: -LT number-of-columns Default assumption: -LT8
---

### FUNCTION

The -LT option specifies the number of columns which becomes the basis of tabulation processing to output a list file by replacing a HT (Horizontal Tab) code in the source module with several blank characters on the list.

### USE

If the number of columns per line of a list file is lessened by specifying the -LW option, use the -LT option to lessen the number of blanks by HT code, thereby saving the number of columns.

### EXPLANATION

- o The number of columns for tabulation to be specified with the -LT option must be within the following value range:  
$$0 \leq \text{No. of columns for tabulation} \leq 8$$
- o If the number of columns is specified as 0 (i.e., -LT0), no tabulation will be performed and instead HT code will be output.
- o If no list file is specified, the -LT option specification will become invalid.

---

-LTList file format  
specification

---

EXAMPLES

- o To output preprocess list with -LT option omitted

```
A>cc78K3 -c310 sampleYprime.c -p
```

```
78K/III Series C Compiler Vx.xx [xx xxx xx]
Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype
SAMPLEYPRIME.C(20) : W745 Expected function prototype
SAMPLEYPRIME.C(26) : W622 No return value
SAMPLEYPRIME.C(37) : W622 No return value
SAMPLEYPRIME.C(44) : W622 No return value
Compilation complete.      0 error(s) and      5 warning(s) found.
```

- o When the preprocess list is referenced

```
/*
78K/III Series C Compiler Vx.xx Preprocess List
Date:xx xxx xxxx Page: 1
```

```
Command : -C310 SAMPLEYPRIME.C -P
In-file : SAMPLEYPRIME.C
PPL-file : PRIME.PPL
Para-file :
*/
```

```
1 : #define TRUE      1
2 : #define FALSE    0
3 : #define SIZE     200
4 :
5 : char    mark[SIZE+1];
6 :
7 : main()
8 : {
9 :     int i, prime, k, count;
10 :
11 :     count = 0;
12 :
13 :     for ( i = 0 ; i <= SIZE ; i++)
14 :         mark[i] = TRUE;
15 :     for ( i = 0 ; i <= SIZE ; i++) {
16 :         if (mark[i]) {
17 :             prime = i + i + 3;
18 :             printf("%6d",prime);
19 :             count++;
20 :             if((count%8) == 0) putchar('\n');
21 :             for ( k = i + prime ; k <= SIZE ; k += prime)
22 :                 mark[k] = FALSE;
23 :         }
24 :     }
25 :     printf("\n%d primes found.",count);
26 : }
27 :
```

---

-LTList file format  
specification

---

- o To specify the number of blanks by HT code as "1"

```
A>cc78K3 -c310 sampleYprime.c -p -lt1
```

```
78K/III Series C Compiler Vx.xx [xx xxx xx]
Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype
SAMPLEYPRIME.C(20) : W745 Expected function prototype
SAMPLEYPRIME.C(26) : W622 No return value
SAMPLEYPRIME.C(37) : W622 No return value
SAMPLEYPRIME.C(44) : W622 No return value
Compilation complete,      0 error(s) and      5 warning(s) found.
```

- o When the preprocess list is referenced

```
/*
78K/III Series C Compiler Vx.xx Preprocess List
                        Date:xx xxx xxxx Page:  1
```

```
Command   : -C310 SAMPLEYPRIME.C -P -LT1
In-file   : SAMPLEYPRIME.C
PPL-file  : PRIME.PPL
Para-file :
*/
```

```
1 : #define TRUE 1
2 : #define FALSE 0
3 : #define SIZE 200
4 :
5 : char mark[SIZE+1];
6 :
7 : main()
8 : {
9 :     int i, prime, k, count;
10 :
11 :     count = 0;
12 :
13 :     for ( i = 0 ; i <= SIZE ; i++)
14 :         mark[i] = TRUE;
15 :     for ( i = 0 ; i <= SIZE ; i++) {
16 :         if (mark[i]) {
17 :             prime = i + i + 3;
18 :             printf("%6d", prime);
19 :             count++;
20 :             if((count%8) == 0) putchar('\n');
21 :             for ( k = i + prime ; k <= SIZE ; k += prime)
22 :                 mark[k] = FALSE;
23 :         }
24 :     }
25 :     printf("\n%d primes found.", count);
26 : }
27 :
```

The number of blanks by HT code is 1.



---

**-LF****List file format  
specification**

---

Description format: -LF Default assumption: None
---

### FUNCTION

The -LF option specifies the addition of a formfeed code to the end of a list file.

### EXPLANATION

If no list file is specified, the -LF option specification will become invalid.

### EXAMPLE

- o To output assembler source module file with -LF option specified

```
A>cc78K3 -c310 sampleYprime.c -a -lf
```

```
78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype  
SAMPLEYPRIME.C(20) : W745 Expected function prototype  
SAMPLEYPRIME.C(26) : W622 No return value  
SAMPLEYPRIME.C(37) : W622 No return value  
SAMPLEYPRIME.C(44) : W622 No return value  
Compilation complete, 0 error(s) and 5 warning(s) found.
```

---

**-W****Warning output  
specification**

---

(15) Warning output specification (-W)

Description format: -W[level]

Default assumption: -W1

**FUNCTION**

The -W option tells the compiler to output or not to output warning messages to the console according to the specified warning message level.

**USE**

Use the -W option to output ordinary or detailed warning messages or to suppress warning message output.

**EXPLANATION**

- o Warning message levels that can be used with the -W option are as follows:

Table 5-9. Warning Message Levels

Level	Description
0	Suppress warning message output.
1	Output ordinary warning messages.
2	Output detailed warning messages.

- o If the -E or -SE option is specified together with the the -W option, warning messages will also be output to the error list file.
- o If warning message level 0 (-W0) is specified, no warning message will be output to the console or error list file (when the -E or -SE option is specified).

---

-W

Warning output  
specification

---

### EXAMPLES

- o When warning messages are referenced with -W option omitted

A>cc78K3 -c310 sampleYprime.c

78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx

SAMPLEYPRIME.C(18) : W745 Expected function prototype  
SAMPLEYPRIME.C(20) : W745 Expected function prototype  
SAMPLEYPRIME.C(26) : W622 No return value  
SAMPLEYPRIME.C(37) : W622 No return value  
SAMPLEYPRIME.C(44) : W622 No return value  
Compilation complete, 0 error(s) and 5 warning(s) found.

- o To suppress warning message output

A>cc78K3 -c310 sampleYprime.c -w0

78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx

Compilation complete, 0 error(s) and 5 warning(s) found.

```
Description format: -V
                    or
                    -NV
Default assumption: -NV
```

- o The -V option specifies the output of the current execution status of the compiler to the console.
- o The -NV option specifies the suppression of current execution status output.

Use the -V option if you wish to display the current execution status of the compiler on the console during the compiler execution.

- o The -V option displays the translation phase names and function names under execution.
- o If both the -V and -NV options are specified at the same item, whichever you specified later will take precedence over the other preceding option.

---

**-V/-NV**

**Execution status**

**display specification**

---

**EXAMPLE**

- o To compile source program with -V option specified

A>cc78K3 -c310 sampleYprime.c -v

78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx

Phase:parser  
main():  
SAMPLEYPRIME.C(18) : W745 Expected function prototype  
SAMPLEYPRIME.C(20) : W745 Expected function prototype  
SAMPLEYPRIME.C(26) : W622 No return value  
printf():  
SAMPLEYPRIME.C(37) : W622 No return value  
putchar():  
SAMPLEYPRIME.C(44) : W622 No return value  
Phase:code generator  
main():  
printf():  
putchar():  
Phase:list and object output  
main():  
printf():  
putchar():  
Compilation complete.      0 error(s) and      5 warning(s) found.

---

**-F**

---

**Parameter file specification**

---

**(17) Parameter file specification (-F)**

Description format: -F filename

Default assumption: Allows input of options or input  
filenames from command line only.

**FUNCTION**

The -F option specifies the input of compiler options or input filename(s) from a specified parameter file.

**USE**

- o If the required information for starting up the C compiler is too excessive to specify in the command line, use the -F option to input two or more compiler options from a specified parameter file at compile time.
- o If the same option specifications are to be used repeatedly for each compilation, describe such options in a parameter file and input them from the parameter file by using the -F option.

**EXPLANATION**

- o Nesting of parameter files is not allowed.
- o The number of characters that can be used in a parameter file is not limited.
- o A blank, tab, or "Ⓔ" is interpreted as a delimiter for each compiler option or input name.
- o The options or input filenames described in a parameter file will be expanded to the location on the command file where the -F option has been specified.
- o As regards the precedence of expanded options, the last specified option in the parameter file takes precedence over the preceding options.

---

-F

---

Parameter file specification

---

- o Characters between ";" or "#" and "Ⓕ" are interpreted as a comment statement.

EXAMPLE

- o Contents of parameter file "PRIME.PCC"

```
;parameter file
sampleYprime.c -c310 -aprime.asm
-e -x
```

- o To compile source program using PRIME.PCC file

```
A>cc78K3 -fprime.pcc
```

```
78K/III Series C Compiler Vx.xx [xx xxx xx]
  Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype
SAMPLEYPRIME.C(20) : W745 Expected function prototype
SAMPLEYPRIME.C(26) : W622 No return value
SAMPLEYPRIME.C(37) : W622 No return value
SAMPLEYPRIME.C(44) : W622 No return value
Compilation complete,      0 error(s) and      5 warning(s) found.
```

---

-T

Temporary file creation  
directory specification

---

(18) Temporary file creation directory specification (-T)

Description format: -T directory

Default assumption: Drive/directory specified by environ-  
ment variable TMP or current drive/  
current directory if TMP is not  
specified

#### FUNCTION

The -T option specifies the drive or directory in which a temporary file is to be automatically generated by the C compiler.

#### USE

Use the -T option to specify a drive or directory for temporary file creation.

#### EXPLANATION

- o Even if a previously created temporary file still exists, the temporary file will be overwritten at the next creation specification unless the file is write-protected.
- o If the memory capacity required for temporary file creation is available, a temporary file will be created in memory. If the required memory size is not available, a temporary file will be created below a specified directory, the memory contents will be written out to the file, and subsequent temporary file access will be made to the file.
- o The created temporary file will be deleted on completion of the compile operation. The temporary file will also be deleted when the compiler processing is interrupted by CTRL-C key input.



---

-T

Temporary file creation  
directory specification

---

EXAMPLES

- o To specify creation of temporary file in directory "TMP"

A>cc78K3 -c310 sampleYprime.c -tmp

78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx

SAMPLEYPRIME.C(18) : W745 Expected function prototype  
SAMPLEYPRIME.C(20) : W745 Expected function prototype  
SAMPLEYPRIME.C(26) : W622 No return value  
SAMPLEYPRIME.C(37) : W622 No return value  
SAMPLEYPRIME.C(44) : W622 No return value  
Compilation complete, 0 error(s) and 5 warning(s) found.

---

--

---

HELP message output  
specification

---

(19) HELP message output specification (--)

Description format: --

Default assumption: No HELP message is to be output.

#### FUNCTION

The -- option tells the C compiler to display a HELP message on the console.

#### USE

The HELP message is a list of compiler options. Use this list for reference when executing the C compiler.

#### EXPLANATION

- o If the -- option is specified, all the other compiler options will become invalid.
- o Press the RETURN key to see the rest of the options in the list which are not shown on the screen. To discontinue the HELP message display, first press any key other than RETURN and then press the RETURN key.

--

---

HELP message output  
specification

---

EXAMPLE

o To specify HELP message display with -- option.

A>cc78K3 --

78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx

Usage : CC78K3 [option[...]] input-file [option[...]]

The option is as follows ([ ] means omissible, ... means repeat).

-cx : Select target chip. ( x = chip name ) \*Must be specified.  
-o[file]/no : Create object file / Not.  
-s/ns : Expand symbol length up to 30 / or symbol length is 7.  
-ca/nca : Convert alphabet to capital for symbol / Not.  
-r/nr : Generate ROMable object module / Not.  
-q[y]/nq : Optimize output code / Not. (default x = r)  
y = s : optimize object speed  
y = z : optimize object size  
y = c : assign char variable without sign expand  
y = l : call function prepare/dispose library  
y = r : use SADDR area for register variable  
y = u : change plain char to unsigned char  
y = e : common subexpression elimination and copy propagation  
y = j : jump optimization  
y = x : max. optimization(y = zcrej)  
-g/ng : Output debug information for object file / Not.

Usage : CC78K3 [option[...]] input-file [option[...]]

-l[x] : Add error check library / Not. (default : x = 2dpsz)  
x = 1 : overflow check (level 1)  
x = 2 : overflow check (level 2)  
x = d : sfr access check  
x = p : illegal pointer access check  
x = s : stack overflow check  
x = z : 0 divide check  
-p[file]/np : Create the preprocess list file / Not.  
-k[x]/nk : Specified preprocess list mode / None. (default x = fln)  
x = c : delete comment  
x = d : execute #define  
x = f : execute #if, #ifdef, #ifndef  
x = i : execute #include  
x = l : execute #line  
x = n : add line number and paging  
-dname[=data][,name[=data]]...  
: Define name with data.  
-uname[,name]...  
: Undefine name.  
-idirectory[,directory]...  
: Set include search path.

--

---

HELP message output  
specification

---

Usage : CC78k3 [option[...]] input-file [option[...]]  
-a[file]/na : Create the assembler source file / Not.  
-sa[file]/nsa : Create the assembler source file with the C source / Not.  
-e[file]/ne : Create the error list file / Not.  
-se[file]/nse : Create the error list file with the C source / Not.  
-x[file]/nx : Create the cross reference list file / Not.  
-lw[width] : Specify list file columns per line. (width=72 to 132)  
-ll[length] : Specify list file lines per page. (length=0, 20 to 65535)  
          0 means no paging.  
-lt[n] : Expand TAB character for list file. (n=1 to 8)  
          Not expand. (n=0)  
-lf/nlf : Add Form Feed at end of list file / Not.  
-w[n] : Change warning level. (n=0 to 2)  
-v/nv : Verbose compile messages / Not.  
-ffile : Input option or source file name from specified file.  
-tdirectory : Set temporary directory  
-- : show this message

## DEFAULT ASSIGNMENT

: -o -r -lw32 -ll66 -nlf -wl

;;



The C compiler outputs the following five files:

- o Object module file
- o Assembler source module file
- o Preprocess list file
- o Cross-reference list file
- o Error list file

### 6.1 Object Module File

An object module file is a binary image file of the result of translating a C source program.

This file also contains debug information when so specified by the debug information output option (-G).

## 6.2 Assembler Source Module File

An assembler source module file is an ASCII image list file of the result of translating a C source program and is a source module file in the assembly language corresponding to the C source program.

This file also contains a C source program as comments when so specified by the assembler source module file creation option (-SA) is specified.

### [Output Format]

```

; 78K/III Series C Compiler V①x.xx Assembler Source
;                                     Date:②xxxxx Time:③xxxxx

; Command      : ④-c310 sampleYprime.c -sa
; C-file       : ⑤SAMPLEYPRIME.C
; Asm-file     : ⑥PRIME.ASM
; Para-file    : ⑦

SPROCESSOR(⑧310)
⑨$NODEBUG

⑫          NAME      PRIME
⑫          EXTRN     @@isrem
          ...

; line ⑩1 :⑪#define TRUE      1
; line ⑩2 :⑪#define FALSE    0
; line ⑩3 :⑪#define SIZE     200
          ...

⑫_main:
⑫      push    hl
⑫      movw    ax, sp
          ...

```

## [Description of Output Items]

Item No.	Item	No. of columns	Format
①	Version number	4	The version number of C compiler is expressed in the form of "x.yz".
②	Date	11 (fixed)	System date is expressed in the form "DD Mmm YYYY".
③	Time	8 (fixed)	System time is expressed in the form "HH:MM:SS".
④	Command line		The contents of the command line after "CC78K3" are output to this column. Characters exceeding the limit value per line are output from the 15th column of the next line. One or more tabs are replaced with one blank.
⑤	C source module filename	78 max. (variable)	Specified C source filename is output here. If file type is omitted, ".c" will be added to the primary name. Characters exceeding the limit value per line are output from the 15th column of the next line. One or more tabs are replaced with one blank.
⑥	Assembler source module filename	78 max. (variable)	Specified assembler source module filename is output here. If file type is omitted, ".asm" will be added to the primary name. Characters exceeding the limit value per line are output from the 15th column of the next line. One or more tabs are replaced with one blank.
⑦	Parameter file		The contents of parameter file are output to this column. Characters exceeding the limit value per line are output from the 15th column of the next line, provided ";" is output to the 1st column. One or more tabs are replaced with one blank.



Item No.	Item	No. of columns	Format
⑧	Processor type	4 (variable)	Character string (processor type) specified by -C option is output. See the -C option in Chapter 5, Compiler Options.
⑨	Debug information	8 (variable)	Either \$DEBUG or \$NODEBUG is output as DEBUG control.
⑩	Line number	5 (fixed)	Line number of C source module file. Each line number is expressed by a decimal number not exceeding five digits and is right-justified with zero suppression for output.
⑪	C source		ASCII image of the input C source program. Characters exceeding the limit value per line are output from the 1st column of the next line.
⑫	Assembler source body		Assembler source as the result of compilation is output. Characters at the 80th and subsequent columns are output without moving them to the next line.

### 6.3 Error List File

An error list file contains information on all warning messages and error messages which have been generated during a compile operation.

By specifying a compiler option, a C source program can be added to an error list file. The error list file containing the C source program can be used as a C source module file by correcting errors in the C source program and deleting comments such as list header from the error list file.

#### 6.3.1 Error list file with C source

[Output Format]

```

/*
78K/III Series C Compiler V①x.xx Error List           Date:②xxxxx Time:③xxx

Command   : ④-c310 sampleYprime.c -se
C-file    : ⑤SAMPLEYPRIME.C
Err-file  : ⑥PRIME.CER
Para-file : ⑦
*/

⑧#define TRUE      1
⑧#define FALSE     0
⑧#define SIZE      200

⑧char    mark[SIZE+1];

⑧main()
⑧{
⑧    int i, prime, k, count;

⑧    cont = 0;
⑧    *** ERROR ⑨F711 ⑩Undeclared 'cont' ; function 'main'

⑧    for ( i = 0 ; i <= SIZE ; i++)
⑧        mark[i] = TRUE;
⑧    for ( i = 0 ; i <= SIZE ; i++) {
⑧        if (mark[i]) {
⑧            *** ERROR ⑨F306 ⑩illegal index , indirection not allowed

/*
Compilation complete, ⑪1 error(s) and ⑫5 warning(s) found.
*/

```

## [Description of Output Items]

Item No.	Item	No. of columns	Format
①	Version number	4	The version number of C compiler is expressed in the form of "x.yz".
②	Date	11 (fixed)	System date is expressed in the form "DD Mmm YYYY".
③	Time	8 (fixed)	System time is expressed in the form "HH:MM:SS".
④	Command line		The contents of the command line after "CC78K3" are output to this column. Characters exceeding the limit value per line are output from the 13th column of the next line. One or more tabs are replaced with one blank.
⑤	C source module filename	78 max. (variable)	Specified C source module filename is output here. If file type is omitted, ".c" will be added to the primary name. Characters exceeding the limit value per line are output from the 13th column of the next line.
⑥	Error list filename	78 max. (variable)	Specified error list filename is output here. If file type is omitted, ".cer" will be added to the primary name. Characters exceeding the limit value per line are output from the 13th column of the next line.
⑦	Parameter file		The contents of parameter file are output to this column. Characters exceeding the limit value per line are output from the 13th column of the next line. One or more tabs are replaced with one blank.

Item No.	Item	No. of columns	Format
⑧	C source		Image of input C source module file. Characters at the 80th and subsequent columns are output without moving them to the next line.
⑨	Error message number	4 (fixed)	Each error or warning message number is output in the form "#nnn", where "#" becomes "A" for an Abort error, "F" for a Fatal error and "W" for a warning and "nnn" is a 3-digit decimal number indicating the type of error.
⑩	Error message		See Chapter 9, Error Messages. Characters at the 80th and subsequent columns are output without moving them to the next line.
⑪	No. of errors	4 (fixed)	Number of errors is expressed by a decimal number not exceeding four digits and is right-justified with zero suppression for output.
⑫	No. of warnings	4 (fixed)	Number of warnings is expressed by a decimal number not exceeding four digits and is right-justified with zero suppression for output.

## 6.3.2 Error list file containing error messages only

## [Output Format]

① SAMPLEYPRIME.C(② 18) : ③ W745 ④ Expected function prototype  
 ① SAMPLEYPRIME.C(② 20) : ③ W745 ④ Expected function prototype  
 ① SAMPLEYPRIME.C(② 26) : ③ W622 ④ No return value  
 ① SAMPLEYPRIME.C(② 37) : ③ W622 ④ No return value  
 ① SAMPLEYPRIME.C(② 44) : ③ W622 ④ No return value

Compilation complete. ⑤ 0 error(s) and ⑥ 5 warning(s) found.

## [Description of Output Items]

Item No.	Item	No. of columns	Format
①	C source module filename	78 max. (variable)	Specified C source module filename is output here. If file type is omitted, ".c" will be added to the primary name.
②	Line number	5 (fixed)	Line number of C source module file. Each line number is expressed by a decimal number not exceed-five digits and is right-justified with zero suppression for output.
③	Error message number	4 (fixed)	Each error or warning message number is output in the form "#nnn", where "#" becomes "A" for an Abort error, "F" for a fatal error and "W" for a warning and "nnn" is a 3-digit decimal number indicating the type of error.
④	Error message		See Chapter 9, Error Messages.

Item No.	Item	No. of columns	Format
⑤	No. of errors	4 (fixed)	Number of errors is expressed by a decimal number not exceeding four digits and is right-justified with zero suppression for output.
⑥	No. of warnings	4 (fixed)	Number of warnings is expressed by a decimal number not exceeding four digits and is right-justified with zero suppression for output.

#### 6.4 Preprocess List File

A preprocess list file is an ASCII image file containing information on only the result of preprocessing executed for a C source program.

If the process type "N" is not specified in the -K option, the preprocess list file can be used as a C source module file.

##### [Output Format]

With PAGEWIDTH = 80

```

/*
78K/III Series C Compiler V①x.xx Preprocess List      Date:②xxxxx Page:③xxx

Command   : ④-c310 sampleYprime.c -p -lw80
C-file    : ⑤SAMPLEYPRIME.C
PPL-file  : ⑥PRIME.XRF
Para-file : ⑦
*/

⑧1 : ⑨#define TRUE      1
⑧2 : ⑨#define FALSE    0
⑧3 : ⑨#define SIZE     200
⑧4 : ⑨
⑧5 : ⑨char      mark[SIZE+1]:
⑧6 : ⑨

```

## [Description of Output Items]

Item No.	Item	No. of columns	Format
①	Version number	4	The version number of C compiler is expressed in the form of "x.yz".
②	Date	11 (fixed)	System date is expressed in the form "DD Mmm YYYY".
③	No. of pages	4 (fixed)	Number of pages is expressed by a decimal number not exceeding four digits and is right-justified with zero suppression for output.
④	Command line		The contents of the command line after "CC78K3" are output to this column. Characters exceeding the limit value per line are output from the 13th column of the next line. One or more tabs are replaced with one blank.
⑤	C source module filename	78 max. (variable)	Specified C source module filename is output here. If file type is omitted, ".c" will be added to the primary name. Characters exceeding the limit value per line are output from the 13th column of the next line. One or more tabs are replaced with one blank.
⑥	Preprocess list filename	78 max. (variable)	Specified preprocess list filename is output here. If file type is omitted, ".ppl" will be added to the primary name. Characters exceeding the limit value per line are output from the 13th column of the next line. One or more tabs are replaced with one blank.



Item No.	Item	No. of columns	Format
⑦	Parameter file		The contents of parameter file are output to this column. Characters exceeding the limit value per line are output from the 13th column of the next line, provided ";" is output to the 1st column. One or more tabs are replaced with one blank.
⑧	Line number	5 (fixed)	Line number of C source module file. Each line number is expressed by a decimal number not exceeding five digits and is right-justified with zero suppression for output.
⑨	C source		Input C source program. If all statements cannot be output on one line, the rest of the statements are output from the 9th column of the next line.

## 6.5 Cross-reference List File

A cross-reference list file contains a list of identifiers such as function names and variable names which have been declared, defined, and referenced in a C source program. The list also contains information such as the attribute and line number of each identifier.

These identifiers are output in the list file in the order of their appearance in the C source program.

## [Output Format]

With PAGEWIDTH=80

78K/III Series C Compiler V①x.xx Cross reference List Date:②xxxxx Page:③x

Command : ④-c310 sampleYprime.c -x -lw80  
 C-file : ⑤SAMPLEYPRIME.C  
 Xref-file : ⑥PRIME.PPL  
 Para-file : ⑦  
 Inc-file : [n] ⑧

ATTRIB	MODIFY	TYPE	SYMBOL	DEFINE	REFERENCE
⑨ EXTERN	⑩	⑪ array	⑫ mark	⑬ 5	⑭ 14 ⑭ 16 ⑭ 22
⑨ EXTERN	⑩	⑪ func	⑫ main	⑬ 7	
⑨ AUTO1	⑩	⑪ int	⑫ i	⑬ 9	⑭ 13 ⑭ 13 ⑭ 13 ⑭ 14
					⑭ 15 ⑭ 15 ⑭ 15 ⑭ 16
					⑭ 17 ⑭ 17 ⑭ 21
⑨ AUTO1	⑩	⑪ int	⑫ prime	⑬ 9	⑭ 17 ⑭ 18 ⑭ 21 ⑭ 21
⑨ AUTO1	⑩	⑪ int	⑫ k	⑬ 9	⑭ 21 ⑭ 21 ⑭ 21 ⑭ 22
⑨ AUTO1	⑩	⑪ int	⑫ count	⑬ 9	⑭ 11 ⑭ 19 ⑭ 20 ⑭ 25

## [Description of Output Items]

Item No.	Item	No. of columns	Format
①	Version number	4	The version number of C compiler is expressed in the form of "x.yz".
②	Date	11 (fixed)	System date is expressed in the form "DD Mmm YYYY".
③	No. of pages	4 (fixed)	Number of pages is expressed by a decimal number not exceeding four digits and is right-justified with zero suppression for output.
④	Command line		The contents of the command line after "CC78K3" are output to this column. Characters exceeding the limit value per line are output from the 13th column of the next line. One or more tabs are replaced with one blank.
⑤	C source module filename	78 max. (variable)	Specified C source module filename is output here. If file type is omitted, ".c" will be added to the primary name. Characters exceeding the limit value per line are output from the 13th column of the next line.
⑥	Cross-reference list filename	78 max. (variable)	Specified cross-reference list filename is output here. If file type is omitted, ".rxf" will be added to the primary name. Characters exceeding the limit value per line are output from the 13th column of the next line.
⑦	Parameter file		The contents of parameter file are output to this column. Characters exceeding the limit value per line are output from the 13th column of the next line. One or more tabs are replaced with one blank.

Item No.	Item	No. of columns	Format
⑧	Include file	78 max. (variable)	<p>Filenames specified in C source are output to this column. n indicates an Inclusion file number which begins with 1. Characters exceeding the limit value per line are output to the 13th line of the next line. If there is no Inclusion file, nothing will be output to this column.</p>
⑨	Symbol attribute	6 (fixed)	<p>One of the following attributes declared for each symbol is output (left-justified).</p> <p>EXTERN .. External variable</p> <p>EXSTC ... External static variable</p> <p>INSTC ... Internal static variable</p> <p>AUTO nn .. auto variable (nn indicates scope beginning with 1)</p> <p>REG nn ... Register variable (nn indicates scope beginning with 1)</p> <p>EXTYP ... External typedef declaration</p> <p>INTYP ... Internal typedef declaration</p> <p>LABEL ... Label</p> <p>TAG ..... Tag of structure or union</p> <p>MEMBER .. Member of structure or union</p> <p>PARAM ... Parameter of function</p>

Item No.	Item	No. of columns	Format
⑩	Type qualifier of symbol	4 (fixed)	One of the following type qualifiers declared for each symbol is output (left-justified). CONST ... const variable VLT ..... volatile variable CALLT ... callt function CALLF ... callf function NOAUTO .. noauto function NOREC ... norec function SREG ... sreg or bit function RWSFR ... Read/write sfr function ROSFR ... Read-only sfr function WOSFR ... Write-only sfr function VECT .... Interrupt function
⑪	Data type of symbol	7 (fixed)	One of the following data types of each symbol is output (left-justified): char, int, short, long, and field. "u" is prefixed to each of these unsigned types. void, func, array, pointer, struct, union, enum, bit and #define are also available as types.
⑫	Symbol name	16 (fixed)	If the symbol name length exceeds the 15th column, the symbol name will be output as is and items 13 and 14 will be output starting from the 39th column of the next line.
⑬	Symbol defined line No.	5 (fixed)	Line number at which each symbol has been defined and filename are output. Line number (5-digit) is expressed in the same format as Inclusion file number.

Item No.	Item	No. of columns	Format
14	Symbol referenced line No.	5 (fixed)	Line number at which each symbol has been referenced and filename are output. Line number (5-digit) is expressed in the same format as Inclusion file number. Characters exceeding the limit value per line are output from the 47th column of the next line.



## CHAPTER 7. EFFECTIVE UTILIZATION OF C COMPILER

## 7.1 EXIT Status Function for Efficient Compilation

At the termination of a compile (translation) operation, this C compiler returns to the OS the maximum error level which has occurred during the compile operation as an EXIT status code.

The following EXIT status codes are available:

- o Normal termination : 0
  - o WARNING is output : 0
  - o FATAL ERROR exists : 1
  - o Abnormal termination : 2
- (Execution is aborted)

By using these codes with a batch file, compile operations can be executed efficiently.

## [Example]

```
cc78k3 -c310 %1
IF ERRORLEVEL 1 GOTO ERR
cc78k3 -c310 %2
IF ERRORLEVEL 1 GOTO ERR
GOTO EXIT
:ERR
echo Some error found.
:EXIT
```

## [Explanation]

In the above example, assume that a fatal error occurs when the C source passed to %1 is compiled. Normally, the C compiler continues its processing after the output of an error message. By using the function which returns EXIT status "1" to the OS, the compiler execution can be stopped without processing the C source in the next %2.



## 7.2 Environment Variables for Development Environment

### Setting

When developing a program, if you create a directory for related files to put together these files, compile operations may be carried out smoothly by the C compiler. This can be implemented by specifying an environment variable.

The C compiler supports the following environment variables:

- o PATH : Path to search directories for executable files
- o INC78Kn: Path to search directories for Inclusion files (where n = 0, 2, 3 indicating each 78K series number)
- o TMP : Path to search directories for temporary files

#### [Example]

```
;AUTOEXEC.BAT
PATH A:%BIN;A:%BAT;A:%CC78K3;A:%TOOL;
verify on
break on
SET INC78K3=A:%CC78K3%INCLUDE
SET LIB78K3=A:%CC78K3%LIB
SET TMP=A:%
```

#### [Explanation]

- o By the PATH specification, the C compiler searches the directories A:%BIN, A:%BAT, A:%CC78K3, and A:%TOOL in this order for executable files.
- o The directory A:%CC78K3%INCLUDE will be searched for Inclusion files.
- o The directory A:%CC78K3%LIB will be searched for library files. (The linker will also be informed of the location of these library files.)
- o A temporary file will be created in the directory A:%.

### 7.3 Interruption of Compile Operation

A compile operation may be interrupted by control key input "CTRL-C". If "break on" is specified, control will be returned to the OS without regard to the key input timing. If "break off" is specified, control will be returned to the OS only during a screen display. On termination of the execution, all the temporary files and output files being open will be deleted.



## CHAPTER 8. START-UP ROUTINES AND ERROR HANDLING ROUTINES

## 8.1 General

## o Start-up routine

In the recent world of single-chip microcomputers as well, the need for program development in C is mounting up. To execute a program written in C, another program to execute ROMable processing for incorporation into a system or to invoke a user program (function `main`) is required. This program is called a start-up routine.

To execute a user-created program, a start-up routine must be created according to the program. However, to those who are less experienced in microcomputer program development in C, creating a start-up routine is not an easy job.

This chapter is intended for such people and explains the contents of each start-up routine and its usage by using a sample program and discusses important points for improving the sample program.

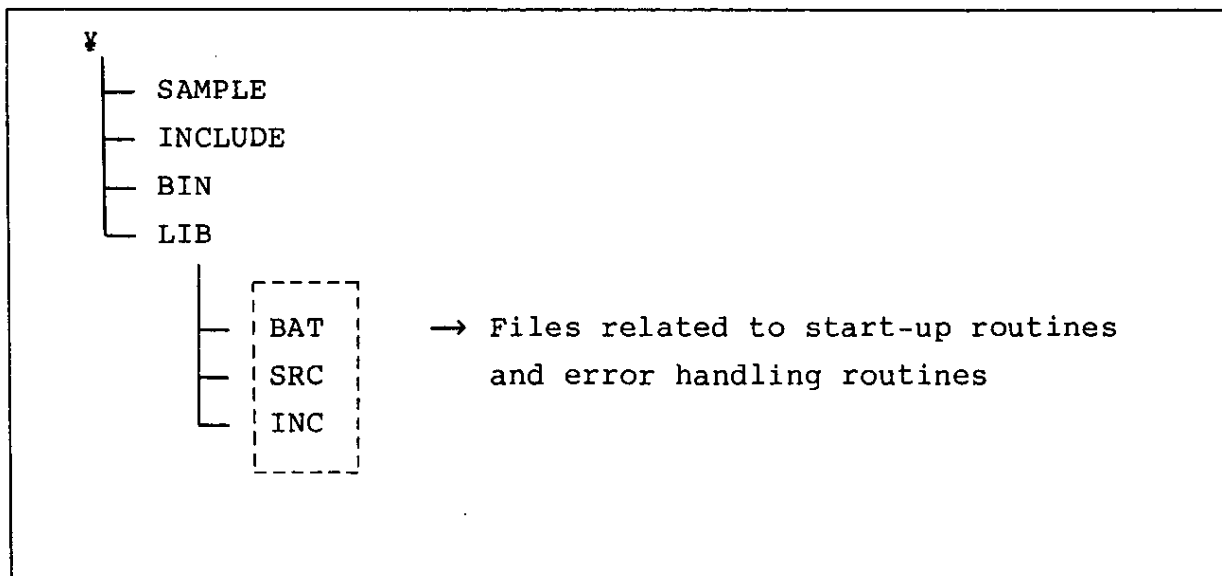
## o Error handling routine

All the C compilers in the CC78K series support a library which allows error checks to be made during program execution. The errors subject to check include an error due to an overflow in the result of an operation, an error due to division by zero, and a pointer access error. If any of these errors occurs, a program to handle the error will be called according to the type of error or the target device. This program is referred to as an error handling routine.

In this chapter, the contents of each error handling routine and its usage are also explained by using a sample program and important points for improving the sample program are discussed just the same as the start-up routine.

## 8.2 File Organization

Files related to the start-up routines and error handling routines are stored in three directories under the directory "LIB" in the supplied floppy disk 1.



The contents of the three directories BAT, SRC, and INC are as listed in Tables 8-1, 8-2, and 8-3, respectively.

Table 8-1. Contents of Directory "BAT"

Filename	Function
<Batch files>	
MKSTUP.BAT	o Batch file for creating a start-up routine
MKERRLIB.BAT	o Batch file for updating error handling routine libraries
CLXXX.ERR	o Subcommand file for creating library file CLXXX.LIB (used in MKERRLIB.BAT)
CLXXXU.ERR	o Subcommand file for creating library file CLXXXU.ERR (used in MKERRLIB.BAT)
	XXX: Processor type (see Tables 5-3 to 5-5 in Chapter 5)

Table 8-2. Contents of Directory "SRC"

Filename	Function
<Start-up routine source files>	
CSTARTR.ASM	o Start-up routine for ROMable processing
ESTARTR.ASM	o Start-up routine for ROMable processing (with ROMable processing error check)
CSTART.ASM	o Start-up routine without ROMable processing
ROM.ASM	o File for ROMable processing
<Error handling routine source files>	
ERRSTK.ASM	o Error handling on Stack overflow
ERRDIV.ASM	o Error handling on Divide by 0
ERRPTR.ASM	o Error handling on Pointer access
ERROVF.ASM	o Error handling on Overflow
ERRSFR.ASM	o Error handling on sfr access
ERRINI.ASM	o Error handling on ROMable area
<Inclusion files>	
DEFINE.INC	o Label definitions of saddr area
MACRO.INC	o Macrodefinitions with respect to Move instructions in each target device

Table 8-3. Contents of Directory "INC"

Filename	Function
<Inclusion file>	
CHIPXXX.INC	o Device information
	XXX: Processor type (see Tables 5-3 to 5-5 in Chapter 5)

### 8.3 Description of Each Batch File

#### 8.3.1 Batch file for creating a start-up routine

To create a start-up routine object file, the batch file "MKSTUP.BAT" stored under the directory "BAT" in the floppy disk 1 must be used.

Also, in the batch file MKSTUP.BAT, the Assembler in the RA78K series assembler package must be used. Therefore, set the Assembler in the directory in which the batch file exists or in a specified PATH. The required files of the Assembler are as shown below.

<with 78K/III>

RA78K3.EXE	(Executable file)
RA78K3.OM1 to RA78K3.OM6	(Overlay files)

The usage of this batch file is explained below.

#### Usage

Execute the following command in the directory BAT in which MKSTUP.BAT file exists:

A> <u>MKSTUP processor-type<sup>Cf1</sup> symbol-upper-/lowercase spec<sup>Cf2</sup> ®</u>
--

- Note: 1. See Tables 5-3 to 5-5 in Chapter 5 for the processor types.
2. See 5.4 (4), Symbol name upper-/lowercase specification in Chapter 5.

Example

To create a start-up routine for the target device uPD78320 at Compile time without specifying the symbol name upper-/lowercase specification option. (This option is defaulted to -NCA.)

```
A>MKSTUP 320 NCA Ⓢ
```

The batch file MKSTUP.BAT creates a directory named REL320 at the same level as the directory BAT as shown below. Then, start-up routine object files are stored under this newly created directory REL320.

```
├ BAT
└ REL320 - [ CS320R.REL
              ES320R.REL
              CS320.REL
              ROM320.REL ]
```



### 8.3.2 Batch file for updating error handling routine libraries

To replace an improved error handling routine with the old error handling routine in a library file, the batch file "MKERRLIB.BAT" stored under the directory BAT in the floppy disk 1 must be used. With the batch file MKERRLIB.BAT, the following two preparatory operations must be carried out:

#### (1) Setting the Assembler and Librarian

In the batch file MKERRLIB.BAT, the Assembler and Linker in the RA78K series assembler package must be used. Therefore, set the Assembler and Librarian in the directory in which the batch file exists or in a specified PATH. The required files of the Assembler and Librarian are as shown below.

<with 78K/III>

RA78K3.EXE	(Executable file)
RA78K3.OM1 to RA78K3.OM6	(Overlay files)
LB78K3.EXE	(Executable file)

#### (2) Setting a library file

Create a new directory LIB for storing an update library file at the same level as the directory BAT and store the library file in the directory LIB.

<pre> ├ BAT └ LIB - [ CLXXX.LIB ] -- Update library file </pre>
---

The usage of this batch file is explained below.

Usage

Execute the following command in the directory BAT in which MKERRLIB.BAT file exists:

```
A>MKERRLIB processor-type symbol-upper-/lowercase spec ®
```

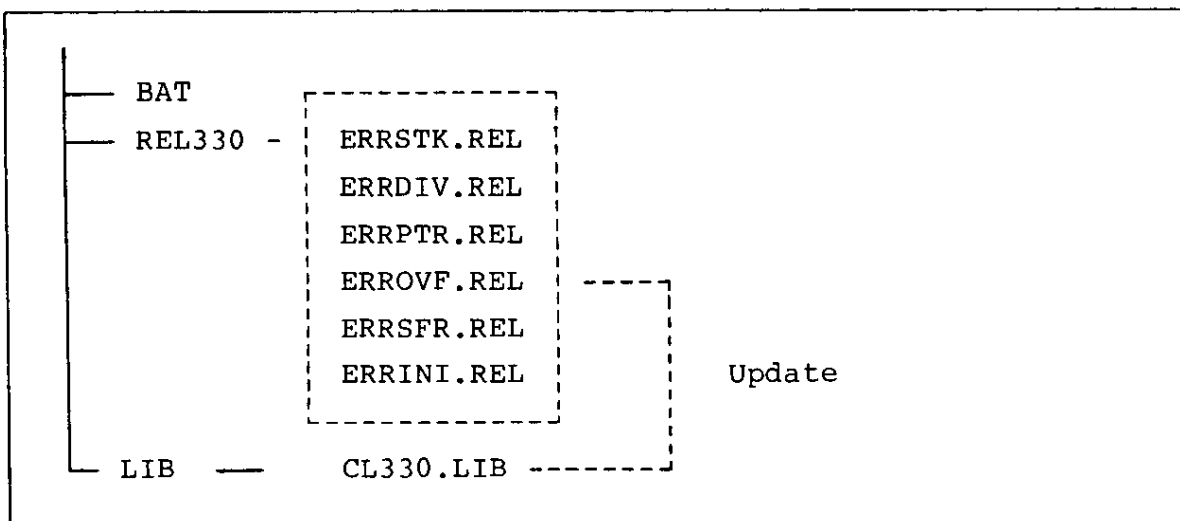
The processor type and symbol name upper-/lowercase specifications are the same as those explained in 8.3.1 above.

Example

To update library file CL330.LIB to be used for the target device uPD78330 at Compile time without specifying the symbol name upper-/lowercase specification option). (This option is defaulted to -NCA.)

```
A>MKERRLIB 330 NCA ®
```

The batch file MKERRLIB.BAT creates a directory named REL330 at the same level as the directory BAT as shown below. Then, error handling routine object files are created and stored under this newly created directory REL330. Finally, the Librarian updates the error handling routines in library file CL330.LIB with the error handling routines in directory REL330.



## 8.4 Start-up Routines

### 8.4.1 Outline of start-up routine

A start-up routine is a program to make necessary preparations for executing a user-created C source program. By linking the start-up routine with the user program, a load module file which serves for the intended purpose can be created.

#### (1) Functions

The start-up routine executes ROMable processing for incorporation into a system, invocation and termination of a C source program.

ROMable ... The initial values of all the external variables, processing static variables, and sreg variables defined in a C source program are allocated to the ROM area. However, if these variables remain allocated to the ROM area, their initial values cannot be rewritten. Thus, these initial values must be copied from the ROM area into a RAM. This processing is called a ROMable processing. When a program is encoded into a ROM, it is made operable on a microcomputer.

#### (2) Configuration

The sample program related to the start-up routine and its configuration are shown in Table 8-4 below.

Table 8-4. Start-up Routine to Be Used

Not for ROMable processing	For ROMable processing	
	For debugging	For incorporation into system
<div>cstart.asm</div> <div>Preprocess</div> <div>Initialize</div> <div>main function execution</div> <div>Postprocess</div>	<div>estarttr.asm</div> <div>Preprocess</div> <div>Initialize</div> <div>ROMable process</div> <div>main function execution</div> <div>Postprocess</div> <div>rom.asm</div> <div>Label definitions to be used for ROMable process</div>	<div>cstartr.asm</div> <div>Preprocess</div> <div>Initialize</div> <div>ROMable process</div> <div>main function execution</div> <div>Postprocess</div> <div>rom.asm</div> <div>Label definitions to be used for ROMable process</div>

The contents of start-up routine source files "cstartr.asm" and "estarttr.asm" are nearly the same as those of "cstart.asm". "estarttr.asm" also performs an error check in ROMable processing.

The start-up routine source file "rom.asm" contains labels definitions indicating the final addresses of data to be copied in ROMable processing and is required for the ROMable processing of a program. When using the cstartr.asm or estarttr.asm file, be sure to link "rom.asm" with the file. For how to link these files, see Section 4.5, "ROMable Processing of Programs" in Chapter 4.

## (3) Selective use of start-up routines

The user must create a start-up routine object file corresponding to each target device and link it with the program. For how to create the start-up routine object file, see 8.3.1, Batch file for creating start-up routine.

The name of an object file corresponding to each source file is shown in Table 8-5.

Table 8-5. Correspondence of Source Files to Object Files

Type of file	Source file	Object file
Start-up routine	cstart.asm	csxxx.rel (see Note 1) (csxxxu.rel)(see Note 2)
	estarttr.asm	esxxx.rel (esxxxu.rel)
	cstarttr.asm	csxxxr.rel (csxxxru.rel)
ROMable file	rom.asm	romxxx.rel (romxxxu.rel)

Note: 1. xxx: Processor type (see Tables 5-3 to 5-5 in Chapter 5)

2. ( ) indicates the start-up routine to be used when the -CA option is specified at Compile time.  
The compiler option -CA tells the C compiler not to distinguish between symbol names written in uppercase letters and those in lowercase letters.

Use these start-up routines selectively according to the development phase to be used. Table 8-6 shows the recommended phase for use with each routine.

Table 8-6. Selective Use of Start-up Routines

Type of start-up routine	Difference		Advantages when used	Phase recommended for use
	ROMable	ROMable w/check		
cstart.asm	x	x	Faster compile and linking process	Phases up to on-desk debugging
estarttr.asm	o	o	Error check can be made during ROMable processing	Incorporation into system (at time of error check in ROMable processing)
cstarttr.asm	o	x	-	Incorporation into system (at time of ROM)

#### 8.4.2 Description of sample program

Here, the contents of a start-up routine is explained by using examples of `estarttr.asm` and `rom.asm`.

The sample program used is applicable to the 78K/III.

##### o `estarttr.asm`

The start-up routine sample program "`estarttr.asm`" is explained in the order of preprocessing, initialization, ROMable processing, main function execution, and postprocessing shown in Table 8-4.

A difference between `estarttr.asm` and `cstart.asm` or `cstarttr.asm` is shown in Table 8-7 below. The contents of the sample program list and their descriptions are shown on the following pages.

Table 8-7. Comparison of Contents between Start-up Routines

estartr.asm	cstart.asm	cstartr.asm
Preprocessing	Same as estartr.asm	Same as estartr.asm
Initialization	Ditto	Ditto
ROMable processing	None	Partially different from estartr.asm
main function execution & post processing	Same as estartr.asm	Same as estartr.asm

Preprocessing

See page 8-17 for  
explanation.

NAME    @cstart

```

$INCLUDE 'CHIP.INC'
$INCLUDE 'DEFINE.INC'
$INCLUDE 'MACRO.INC'

```

① Inclusion of Include files

PUBLIC   \_@cstart

```

PUBLIC  _errno, _FNCTBL, _FNCENT, _BRKADR, _MEMTOP, _MEMBTM, _SEED
PUBLIC  _@DIVR, _LDIVR, _@TOKPTR

```

②

External definition declaration  
of symbols

```

PUBLIC  _@D_FUNC
PUBLIC  _@D_LINE
PUBLIC  _@KREG00

```

③ External definition declaration of  
labels for saddr area

```

PUBLIC  _@FARG1H
PUBLIC  _@FARG0L
PUBLIC  _@FARG0H

```

```

EXTRN  _main, _exit, _@STBEG — ④ External reference declaration of
                                stack-resolving symbol
EXTRN  _?R_INIT, _?R_DATA, _?R_INIS, _?R_DATS — ⑤ External reference declaration of
                                labels for ROMable processing

```

;error check

```

EXTRN  _@errini
EXTRN  _?INIT, _?DATA, _?INIS, _?DATS — ⑥ Same as ⑤

```



## Initialization

See page 8-18 for explanation.

## @@DATA DSEG

```

_errno:      DS      2
_@FNCTBL:    DS     2*32
_@FNCENT:    DS      2
_@BRKADR:    DS      2
_@SEED:      DS      4
_@DIVR:      DS      4
_@LDIVR:     DS      8
_@TOKPTR:    DS      2
_@MEMTOP:    DS     32
_@MEMBTM:

```

① Area reservation for symbols to be used in library

## @@CODE CSEG

## \_@cstart:

```

SEL      RB7      — ② Register bank setting
MOVW     AX, #_@STBEG
MOVW     SP, AX      ;SP <- stack begin address

```

③ Stack pointer setting

## \$\_IF(P312A OR P322 OR P328 OR P334)

```

MOVW     AX, #0
MOVW     !_errno, AX      ;errno <- 0
MOVW     !_@FNCENT, AX    ;FNCENT <- 0
MOVW     !_@SEED+2, AX
MOVW     AX, #1
MOVW     !_@SEED, AX      ;SEED <- 1
MOVW     AX, #_@MEMTOP
MOVW     !_@BRKADR, AX    ;BRKADR <- #MEMTOP

```

④ Initial value setting of symbols

Processor type:  
other than 310, 312

## \$\_ELSE

```

MOV      A, #0
MOV      !_errno, A
MOV      !_errno+1, A      ;errno <- 0
MOV      !_@FNCENT, A
MOV      !_@FNCENT+1, A    ;FNCENT <- 0
MOV      !_@SEED+1, A
MOV      !_@SEED+2, A
MOV      !_@SEED+3, A
MOV      A, #1
MOV      !_@SEED, A      ;SEED <- 1
MOVW     AX, #_@MEMTOP
MOV      !_@BRKADR+1, A
XCH      A, X
MOV      !_@BRKADR, A      ;BRKADR <- #MEMTOP

```

⑤ Same as ④

Processor type:  
310 or 312

## \$\_ENDIF

ROMable processing

See 8-20 for  
explanation.

## ;ROM DATA COPY

```

MOVW    DE, #_@INIT
MOVW    HL, #_@R_INIT
MOVW    UP, #_?R_INIT

```

;error check

```

MOVW    AX, #_?INIT
SUBW    AX, DE
ADDW    AX, HL
SUBW    AX, UP
BZ      $LINIT1
CALL    !_@errini

```

② Error check  
in ROMable  
processing

①

ROMable processing

## LINIT1:

```

CMPW    HL, UP
BE      $LINIT2
MOV     A, [HL+]
MOV     [DE+], A
BR      $LINIT1

```

## LINIT2:

```

.
.
.
.

```

## LINIS2:

```

MOVW    DE, #_@DATS
MOVW    HL, #_@R_DATS
MOVW    UP, #_?R_DATS

```

;error check

```

MOVW    AX, #_?DATS
SUBW    AX, DE
ADDW    AX, HL
SUBW    AX, UP
BZ      $LDATS1
CALL    !_@errini

```

②

①

## LDATS1:

```

CMPW    HL, UP
BE      $LDATS2
MOV     A, [HL+]
MOV     [DE+], A
BR      $LDATS1

```

## LDATS2:

main function execution  
& postprocessing

See page 8-23  
for explanation.

```

CALL    !_main          ;main();  — ① Execution of main function
MOVW    AX, #0
PUSH    AX
CALL    !_exit          ;exit(0);  — ② Execution of exit function
POP      AX
BR      SS

:
@@R_INIT      CSEG
_@R_INIT:
@@R_DATA      CSEG
_@R_DATA:
@@R_INIS      CSEG
_@R_INIS:
@@R_DATS      CSEG
_@R_DATS:
@@INIT DSEG
_@INIT:
@@DATA DSEG
_@DATA:
@@INIS DSEG      SADDRP
_@INIS:
@@DATS DSEG      SADDRP
_@DATS:
@@CALT CSEG      CALLTO
@@CNST CSEG
@@BITS BSEG

END

```

③ Segment/label definition to be used in ROMable processing

## (1) Preprocessing

Steps ① to ⑥ in the preprocessing phase of `estarttr.asm` are explained here.

## ① Inclusion (or insertion) of Include files

`CHIP.INC` → Device information (file to execute required settings according to each target device in the Initialization phase of the start-up routine)

`DEFINE.INC` → File for defining labels for `saddr` area

`MACRO.INC` → File for macrodefinitions with respect to Move instructions of each device

## ② External definition declaration of symbols

For details of each symbol, see (2) Initialization.

③ External definition declaration of labels for `saddr` area

For details of each label, see Appendix D - List of `saddr` Area Labels in the CC78K Series C Compiler Package User's Manual for Language.

## ④ External reference declaration of stack-solving symbol

- o Automatically generates a stack-resolving PUBLIC symbol (`__@STBEG`).

The symbol `__@STBEG` has the end address of the stack area + 1 as its value.

- o `__@STBEG` can be automatically generated by specifying the linker option `-S` (for stack-resolving symbol generation). Be sure to specify the `-S` option at Linking time.

## ⑤ ⑥ External reference declaration of labels for ROMable processing

- o Labels in ⑤ are defined in the postprocessing (see page 8-24).
- o Labels in ⑥ are defined in `rom.asm` (file for ROMable processing) (see page 8-25).

## (2) Initialization

Steps ① to ⑤ in the Initialization phase of `estart.asm` are explained here.

- ① Area reservation for symbols to be used in library  
Reserves areas for symbols to be used in the library.  
For details of each library function, see Section 10.3,  
Standard Library Functions in the CC78K Series C Compiler  
Package User's Manual for Language.

① - 1

<code>_errno:</code>	<code>DS</code>	<code>2</code>
----------------------	-----------------	----------------

- . An area for setting error code is reserved. This area is used by the following four library functions:

<code>strtol, strtoul</code>	(Character and string functions)
<code>brk, sbrk</code>	(Memory functions)

① - 2

<code>__@FNCTBL:</code>	<code>DS</code>	<code>2*32</code>
<code>__@FNCENT:</code>	<code>DS</code>	<code>2</code>

- . The following areas to be used by the `exit` function are reserved:
- `__@FNCTBL`: Indicates the start address of the area to be used by `atexit` function.  
`atexit` registers the address of each function in this area.
- `__@FNCENT`: Stores the total number of functions registered by `atexit` (up to 32 functions).

① - 3

<code>__@BRKADR:</code>	<code>DS</code>	<code>2</code>
-------------------------	-----------------	----------------

- . An area to set a break value is reserved.  
A break value indicates the start address of an area to be used by memory functions.

① - 4    

_@SEED:            DS    2
----------------------------

- . An area to store a value which becomes the base of pseudo-random number string.
- . This area is used by **rand** and **srand** (mathematical functions).

① - 5    

_@DIVR:            DS    4
----------------------------

- . In **div** (mathematical function), an area to store the result of a calculation is reserved.

① - 6    

_@LDIVR:           DS    8
----------------------------

- . In **ldiv** (mathematical function), an area to store the result of a calculation is reserved.

① - 7    

_@TOKPTR:          DS    2
----------------------------

- . In **strtok** (character/string function), an area to store a pointer is reserved.

① - 8    

_@MEMTOP:          DS    32
_@MEMBTM:

- . An area to be used by memory functions is reserved.  
With this sample program, a 32-byte area is reserved.
- . `__@MEMTOP` and `__@MEMBTM` are labels indicating the start and end addresses of this area, respectively.

## ② Register bank setting

Sets registers in register bank RB7 as work registers.

## ③ Stack pointer setting

- o Sets `@STBEG` in the stack pointer.
- o `__@STBEG` can be automatically generated by specifying the linker option `-S` (for stack-resolving symbol generation).

## ④ ⑤ Initial value setting of symbols

- o Sets initial value "0" in `__errno` and `__@FNCENT`. A positive integer value will be given during the execution of the library function corresponding to each symbol.
- o Sets initial value "1" in `__@SEED`. Pseudo-random number string is governed by the value of `__@SEED`. This value can be set by `srand` function. If `rand` is called before calling `srand`, it is the same as when `srand` is called with the value of `__@SEED` set to "1".
- o Sets the start address of an area reserved for memory functions (`__@MEMTOP`) as a break value.

## (3) ROMable processing

Steps ① and ② in the ROMable processing phase of `estarttr.asm` are explained here.

### ① ROMable processing

In this processing, the initial values of all the external variables and `sreg` variables allocated to ROM are copied to a RAM. There are four types of variables subject to processing, (a) to (d) as shown in the following example:

Example)

```

char    c = 1; ..... (a)  External variable with
                           initial value

int     i;      .....(b)  External variable without
                           initial value (see Note)

sreg int  si=0; .....(c)  sreg variable with
                           initial value

sreg char  sc;   .....(d)  sreg variable without
                           initial value (see Note)

main ()
{
    .
    .
    .
}

```

Note: An external variable without initial value or a sreg variable without initial value will be initialized with 0 and thus become the same as a variable which has initial value "0".

- o In Fig. 8-1, the ROMable processing of (a) external variable with initial value is illustrated. The initial value of variable (a) is allocated to the segment named "@@R\_INIT" on the ROM. The labels which indicate the start and end addresses of the area to which the initial value is allocated are @\_R\_INIT and \_?R\_INIT, respectively. In ROMable processing, these values are copied to the segment named "@@INIT" on the RAM and the labels indicating the start and end addresses of this area are defined as \_@INIT and \_?INIT, respectively.
- o Similar processing is performed on variables (b), (c), and (d).



Table 8-8 shows the segment names of the ROM to which variable (a) to (d) are allocated, and the start and end labels of the initial value of each segment and Table 8-9 shows the same information at the RAM (copy destination).

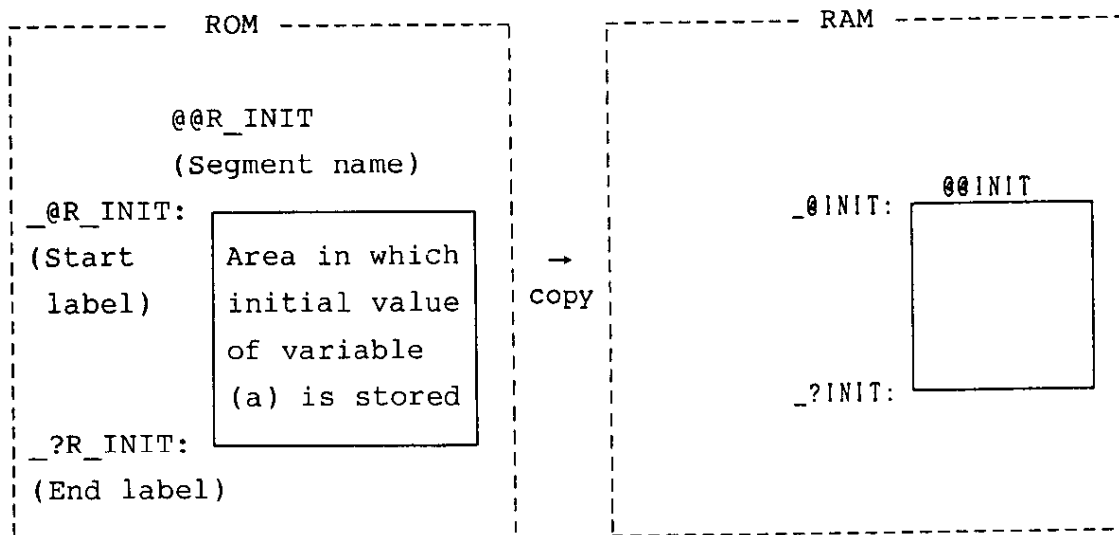


Fig. 8-1. ROMable Processing

Table 8-8. Initial Values of Variables in ROM Area

Type of variable	Segment	Start label	End label
External variable (a) with initial value	@@R_INIT	_@R_INIT	_?R_INIT
External variable (b) w/o initial value	@@R_DATA	_@R_DATA	_?R_DATA
sreg variable (c) with initial value	@@R_INIS	_@R_INIS	_?R_INIS
sreg variable (d) w/o initial value	@@R_DATS	_@R_DATS	_?R_DATS

Table 8-9. Initial Values of Variables in RAM Area  
(Copy Destination)

Type of variable	Segment	Start label	End label
External variable (a) with initial value	@@INIT	_ <b>@</b> INIT	_ <b>?@</b> INIT
External variable (b) w/o initial value	@@DATA	_ <b>@</b> DATA	_ <b>?@</b> DATA
sreg variable (c) with initial value	@@INIS	_ <b>@</b> INIS	_ <b>?@</b> INIS
sreg variable (d) w/o initial value	@@DATS	_ <b>@</b> DATS	_ <b>?@</b> DATS

- ② Error check on ROMable processing (estarttr.asm only)  
With `estarttr.asm`, upon copying the initial values of variables to the RAM, a check is made on whether or not the ROMable processing has been carried out properly. If not, the error handling routine "`errini.asm`" is called. In this check, the data size on the ROM is compared with the data size on the RAM (copy destination).

(4) **main** function execution and postprocessing

Steps ① to ③ in the **main** function execution and post-processing phase of `estarttr.asm` are explained here.

- ① **main** function execution  
Calls the **main** function.
- ② **exit** function execution
  - o Calls the **exit** function.
  - o **exit** function executes all the library functions registered by **atexit** function in sequence starting from the last registered function. The data of the respective functions registered by **atexit** are stored in the areas `_@FNCTBL` and `_@FNCENT` defined in the Initialization phase.

- ③ Segment/label definition to be used in ROMable processing
- o Defines segments and labels to be used in ROMable processing for each of variables (a) to (d).  
A segment indicates an area to store the initial value of each variable, whereas a label indicates the start address of each segment.

(Example) File "rom.asm" for ROMable processing

```

NAME    @rom
;
PUBLIC  _?R_INIT, _?R_DATA, _?R_INIS, _?R_DATS
PUBLIC  _?INIT, _?DATA, _?INIS, _?DATS
;
@@R_INIT      CSEG
_?R_INIT:
@@R_DATA      CSEG
_?R_DATA:
@@R_INIS      CSEG
_?R_INIS:
@@R_DATS      CSEG
_?R_DATS:
@@INIT DSEG
_?INIT:
@@DATA DSEG
_?DATA:
@@INIS DSEG   SADDRP
_?INIS:
@@DATS DSEG   SADDRP
_?DATS:
;
END

```

① Definitions of labels for use in ROMable processing

## 8.4.3 Points for improvement

## (1) Symbols to be used for library functions

If any of the library functions shown in Table 8-10 is not to be used, the symbol corresponding to the library function in a start-up routine may be deleted.

However, symbols `_@FNCTBL` and `_@FNCENT` cannot be deleted, because the `exit` function will be used in all start-up routines.

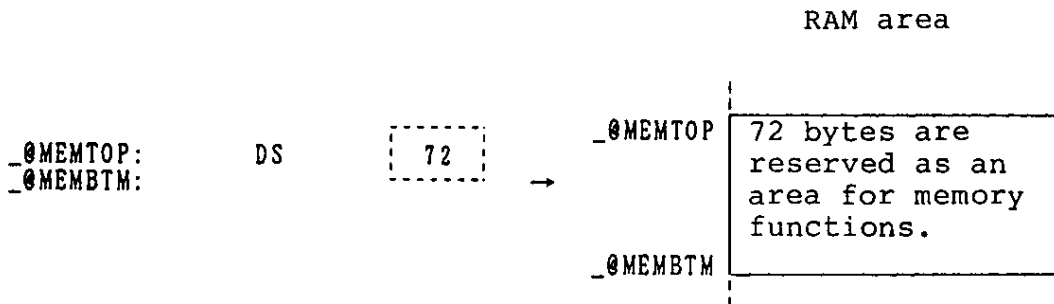
Table 8-10. Symbols to Be Used in Library Functions

Library function		Symbol to be used
Type of function	Function name	
Character/string function	<code>strtol</code>	<code>_errno</code>
	<code>strtoul</code>	
Memory function	<code>brk</code>	
	<code>sbrk</code>	
Program control function	<code>exit</code>	<code>_@FNCTBL</code>
		<code>_@FNCENT</code>
Memory function	<code>malloc</code>	<code>_@MEMTOP</code>
	<code>calloc</code>	<code>_@MEMBTM</code>
	<code>realloc</code>	<code>_@BRKADR</code>
	<code>free</code>	
	<code>brk</code>	
	<code>sbrk</code>	
Mathematical function	<code>rand</code>	<code>_@SEED</code>
	<code>srand</code>	
	<code>div</code>	<code>_@DIVR</code>
	<code>ldiv</code>	<code>_@LDIVR</code>
Character/string function	<code>strtok</code>	<code>_@TOKPTR</code>

## (2) Area to be used for memory functions

If you are to define the size of an area to be used for memory functions, set the area size as shown in the following example:

Example: If you wish to secure a 72-byte area for memory functions, set the area size in the Initialization phase of the start-up routine as follows:



If the specified size is too large, the area cannot be secured in the RAM area, resulting an error at Linking time.

Avoid this error by either of the following two methods:

(a) To reduce the specified size

Example)

`_@MEMTOP: DS 72` → Change to 40 (bytes).

(b) To enlarge the RAM size in the link directive file

Example) With standard link directive file

LINK320.ROM

	Start address	Size	
-----			
memory CLT :	( 00040h ,	00040h )-	
memory ROM :	( 02000h ,	0B000h )	
memory RAM :	( 0D000h ,	01000h )	→
memory SDR :	( 0FE2Ch ,	00050h )	
merge CALT :	=	CLT	
merge CNST :	=	ROM	
merge CODE :	=	ROM	
merge R_DATA :	=	ROM	
merge DATA :	=	RAM	
merge R_INIT :	=	ROM	
merge INIT :	=	RAM	
merge R_DATS :	=	ROM	
merge DATS :	=	SDR	
merge R_INIS :	=	ROM	
merge INIS :	=	SDR	
merge BITS :	=	SDR	

Enlarge this size.  
When changing the size  
of an area, take care  
to avoid overlap with  
another area.

The standard link directive file is stored under the  
directive LIB in the floppy disk 1.

## 8.5 Error Handling Routines

### 8.5.1 Outline of error handling routine

This C compiler package supports a library which allows error checks to be made during program execution. The errors subject to check are listed in paragraph (1) below. If any of these errors occurs, an error handling routine (a program to handle the error) is called. The purpose of the error handling routine is to allow debugging according to the type of error or the target device.

#### (1) Functions

If an error occurs during the execution of a library function, the C compiler will take an appropriate action according to the type of error or the target device. The errors subject to check are the following six types:

- o Stack overflow
- o Error due to division by zero
- o Pointer access error
- o Overflow in the result of an operation
- o sfr access error
- o ROMable area error

## (2) Configuration

The configuration of the error handling routine sample program is shown in Fig. 8-2.

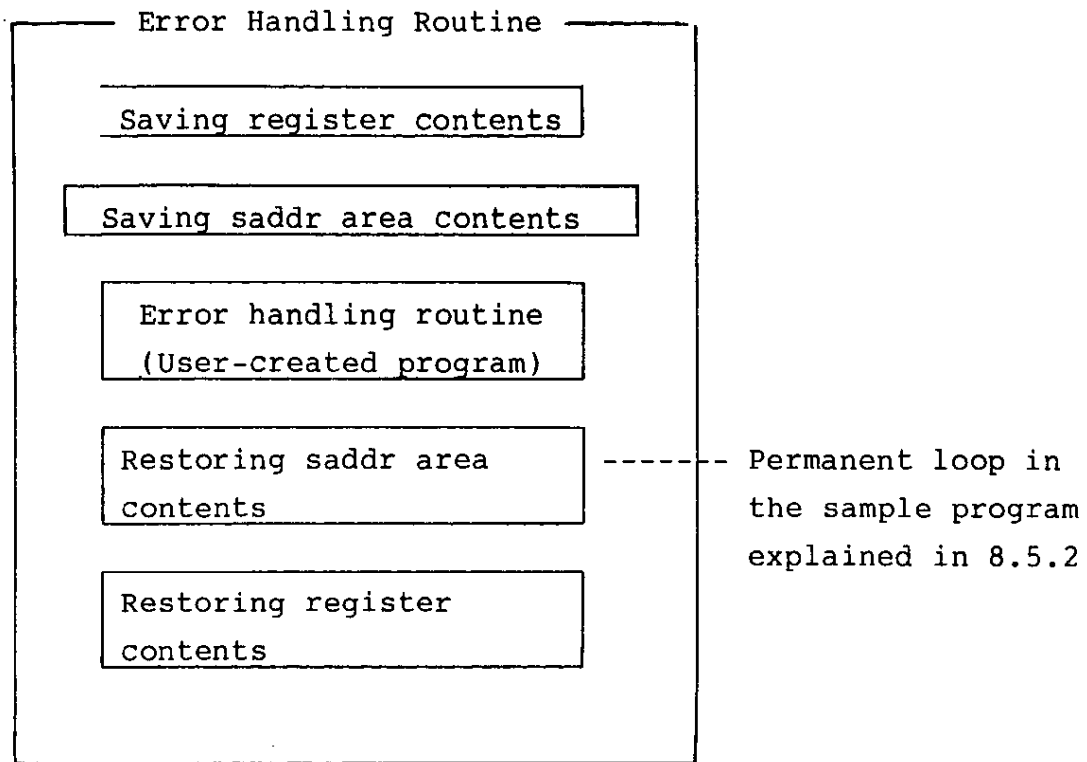


Fig. 8-2. Configuration of Error Handling Routine



## (3) Relations with library functions

Table 8-11 shows the cause of each error handling routine call, the function that calls the error handling routine in the event of an error, and the condition under which the error handling routine is to be called.

Table 8-11. Error Handling Routines

Error handling routine	Cause of call	Calling function	Condition
errstk.asm	Stack overflow	Any function in library file	If execution time error check option is specified at Compile time
errdiv.asm	Divide by zero		
errptr.asm	Pointer access error		
errovf.asm	Overflow in operation		
errsfr.asm	sfr access error		
errini.asm	ROMable process error	Start-up routine estartr	If estartr is used for linking

For information on the type of error check that can be made by each library function, see Section 12.4 (3), List of run time library functions in Chapter 12 of the CC78K Series C Compiler Package User's Manual for Language.

For example, mathematical function `div` checks an overflow in the result of an operation if the execution time error check option (`-L`) is specified at Compile time. If an overflow occurs during the execution of the `div` function, the error handling routine "errovf.asm" will be called.

## 8.5.2 Description of sample program

Of the six error handling routine sample programs, `errstk.asm` is explained here. The other sample programs are the same as `errstk.asm` with the exception of the symbol name for external definition declaration.

The sample program used here is applicable to the 78K/III series.

## o Error handling routine "errstk.asm"

```

$INCLUDE 'CHIP.INC'
$INCLUDE 'DEFINE.INC'

PUBLIC _errstk ...The contents of the other error handling
routines are the same except this symbol
name.

@@CODE CSEG

_errstk:
    PUSH    RP0, RP1, RP2, RP3, RP4, RP5, RP6, RP7    — ① Saving register contents
;
;    MOVW    AX, _D_FUNC
;    PUSH    AX
;    MOVW    AX, _D_LINE
;    PUSH    AX
;    ;
;    MOVW    AX, _FARG1H
;    PUSH    AX
;    MOVW    AX, _FARG0L
;    PUSH    AX
;
;    ;
;    ; _errstk + 9AH
L_A:    BR     $L_A    ] ③ Error handling routine
;
;    POP     AX
;    MOVW    _FARG0H, AX
;    POP     AX
;    MOVW    _FARG0L, AX
;    ;
;    POP     AX
;    MOVW    _D_LINE, AX
;    POP     AX
;    MOVW    _D_FUNC, AX
;
;    POP     RP0, RP1, RP2, RP3, RP4, RP5, RP6, RP7    — ⑤ Restoring register contents
;    RET
;
END

```

② Saving saddr area to be used by C compiler

④ Restoring saddr area

Note: With the CC78K3 V2.0 or higher and CC78K2 V1.0 or higher, ①, ②, ④, and ⑤ are described as comments. In the actual user program, these comments should be removed.

Steps ① to ⑤ in the error handling routine "errstk.asm" are explained below.

- ① Saving register contents  
Saves the contents of register pairs RP0 to RP7.
- ② Saving saddr area to be used by C compiler  
Saves the contents of the saddr area to be used by the C compiler. For details, see Appendix D - List of saddr Area Labels in the CC78K Series C Compiler Package User's Manual for Language.
- ③ Error handling routine  
Create an error handling routine and insert it in here. In the sample program, this part forms a permanent loop.
- ④ Restoring saddr area  
Restores the previously saved contents of the saddr area.
- ⑤ Restoring register contents  
Restores the previously saved contents of register pairs RP0 to RP7.

The contents of the other five error handling routines (errdiv.asm, errptr.asm, errovf.asm, errsfr.asm, and errini.asm) are the same as errstk.asm with the exception of the symbol name. If an error occurs, the following error information will be stored in an area reserved for debugging in the saddr area.

\_@D\_FUNC .... Stores the string of the function name in which the error has occurred.

-@D\_LINE .... Stores the line number in the C source file at which the error has occurred.

### 8.5.3 Point for improvement

With all the error handling routine sample programs offered by this C compiler package, the error handling part forms a permanent loop. For this part, take appropriate actions such as embedding an assembler program, etc. according to the type of error or the target system to be developed by the user.

- Examples)
- o Sounding a buzzer to inform the user of the occurrence of an error
  - o Illuminating an LED to inform the user of the occurrence of an error
  - o Outputting the error information onto the printer
  - o Calling other library functions



## CHAPTER 9. ERROR MESSAGES

## 9.1 Types of Error Messages

The error messages to be output by the C compiler are available in the following 10 types or groups:

- (0) Error messages related to command lines
- (1) Error messages related to internal errors and memories
- (2) Error messages related to characters
- (3) Error messages related to constituent elements
- (4) Error messages related to conversions
- (5) Error messages related to expressions
- (6) Error messages related to statements
- (7) Error messages related to declarations and function definitions
- (8) Error messages related to preprocessor directives
- (9) Error messages related to I/O and optimization

## 9.2 List of Error Messages

Before using the list of error messages contained in this section, you should familiarize yourself with the format of an error number for each error message.

An error number indicates the type of error message and the action to be taken by the C compiler for the error. Therefore, if no compiler action is described for an error in the error message list, refer to the error number explanation below.

An error number is expressed by four alphanumeric characters in the following format:

A/F/Wnnn
----------

The letter symbol A, F, or W denotes the action to be taken by the C compiler as follows:

A: ABORT ..... The C compiler terminates its processing immediately after the output of the error message and outputs neither an object module file nor an assembler source module file.

F: FATAL ..... The C compiler ignores the error portion of the source program and continues its processing after the output of the error message. Neither an object module file nor an assembler source module file will be output.

W: WARNING ... The C compiler continues its processing after the output of the warning message and outputs the file(s) specified by option(s).

"nnn" is a three-digit number indicating the type of error as follows:

001 - 100 ..... Error messages related to command lines  
101 - 200 ..... Error messages related to internal errors  
                  and memories  
201 - 300 ..... Error messages related to characters  
301 - 400 ..... Error messages related to constituent elements  
401 - 500 ..... Error messages related to conversions  
501 - 600 ..... Error messages related to expressions  
601 - 700 ..... Error messages related to statements  
701 - 800 ..... Error messages related to declarations  
                  and function definitions  
801 - 900 ..... Error messages related to preprocessor  
                  directives  
901 - 999 ..... Error messages related to I/O and  
                  optimization

Note: If a syntax error exists in a filename specification, the filename will be added to the error message. If a device type file is specified incorrectly in the command line, the specified character string will be output as is (without change). In cases other than above, the drive name, pathname, and filename extension (file type) will always be added to each filename. Error messages are subject to addition, change, or deletion according to the language specifications of the C compiler to be developed.



## Group 0: Error Messages Related to Command Lines

A001	Message	Missing input file
	Cause	No input file has been specified with the specification of only options other than -F and -- or no Help file exists when the C compiler is started up with the specification of only the executable filename.
A002	Message	Too many input files
	Cause	Input files have been specified by exceeding the limit value (for the number of input files).
A003	Message	Unrecognized string 'string'
	Cause	Characters other than options have been specified in the command line input in the Conversational mode.
A004	Message	Illegal file name 'file name'
	Cause	The specified filename contains character(s) not recognized by OS, the number of characters in the filename specification exceeded the limit value, or a syntax error exists in the filename specification.
A005	Message	Illegal file specification 'file name'
	Cause	The specified filename contains illegal character(s).
A006	Message	File not found 'file name'
	Cause	The specified input filename does not exist.

A007	Message	Input file specification overlapped 'file name'
	Cause	The same named input file has been specified twice in the command line.
A008	Message	File specification conflicted 'file name'
	Cause	The same named input and output files have been specified.
A009	Message	Unable to make file 'file name'
	Cause	The specified output file cannot be created, because it already exists as a read-only file.
A010	Message	Directory not found 'file name'
	Cause	The output filename specification contains a drive or directory which does not exist.
A011	Message	Illegal path 'file name'
	Cause	In the option specification which requires a pathname as its parameter, other than a path-name has been specified.
A012	Message	Missing parameter 'option'
	Cause	The required option has not been specified.
A013	Message	Parameter not needed 'option'
	Cause	The parameter not required for the option has been specified.

A014	Message	Out of range 'option'
	Cause	The specified value for the option parameter is outside the prescribed value range.
A015	Message	Parameter is too long 'option'
	Cause	The number of characters specified for the option parameter exceeded the limit value.
A016	Message	Illegal parameter 'option'
	Cause	A syntax error exists in the option parameter specification.
A017	Message	Too many parameters 'option'
	Cause	The number of parameters described for the option exceeded the limit value.
A018	Message	Option is not recognized 'option'
	Cause	The specified option is not recognized by the C compiler.
A019	Message	Parameter file nested
	Cause	An -F option has been specified in the parameter file.
A020	Message	Parameter file read error 'file name'
	Cause	An attempt to read the parameter file failed.
A021	Message	Memory allocation failed
	Cause	An attempt to allocate a memory area failed.

W022	Message	Same category option specified - ignored 'option'
	Cause	Options contradicting each other have been specified at the same time.
	Action	The C compiler accepts as valid the option whichever you specified later.
W023	Message	Incompatible chip name
	Cause	The processor type specified in the command line differs from that in the C source program.
	Action	The C compiler accepts the processor type specified in the command line as valid.
A024	Message	Illegal chip specifier on command line
	Cause	The processor type specified in the command line is illegal.
W025	Message	'-G' option specified - ignored '-QSZ'
	Cause	With CC78K3 C compiler: The optimize option "-Q" is ignored, because the -G (debug) option has been specified. With C compiler other than CC78K3: The optimize option "-QSZ" is ignored, because the -G (debug) option has been specified.

## Group 1: Error Messages Related to Internal Errors and Memories

F101	Message	Internal error
	Cause	An internal error has occurred.
F102	Message	Too many errors
	Cause	The total number of fatal errors exceeded 30.
	Action	The C compiler continues its processing but will suppress subsequent error message output.
F103	Message	Intermediate file error
	Cause	An error exists in the contents of the intermediate file.
F104	Message	Illegal use of register
	Cause	An error exists in the use of a register.
F105	Message	Register overflow : simplify expression
	Cause	The expression is too complex and no free register is available for the expression.
A106	Message	Stack overflow
	Cause	An overflow has occurred in the stack area.
F107	Message	Symbol table overflow
	Cause	An overflow has occurred in the symbol table area.

F108	Message	Compiler limit : too much automatic data in function
	Cause	The area allocated to the auto variables of a function exceeded the limit value of 64 KB.
F109	Message	Compiler limit : too much parameter of function
	Cause	The area allocated to the parameters of a function exceeded the limit value of 64 KB.
F110	Message	Compiler limit : too much code defined in file
	Cause	The area allocated to codes within a file exceeded the limit value of 64 KB.
F111	Message	Compiler limit : too much global data defined in file
	Cause	The area allocated to global variables within a file exceeded the limit value of 64 KB.

## Group 2: Error Messages Related to Characters

F201	Message	Unknown character 'hexadecimal number'
	Cause	The character which has the specified internal code cannot be recognized by the C compiler.
F202	Message	Unexpected EOF
	Cause	EOF (end-of-file) code is found in the body of a function.

## Group 3: Error Messages Related to Constituent Elements

F301	Message	Syntax error
	Cause	A syntax error has occurred.
F303	Message	Expected identifier
	Cause	An identifier is required for this data object.
F304	Message	Compiler limit: too long identifier 'identifier'
	Cause	The specified identifier is too long.
F305	Message	Compiler limit : too many identifiers with block scope
	Cause	The number of symbols which have block scope within a block is excessive.
F306	Message	Illegal index, indirection not allowed
	Cause	A subscript has been used for an expression which cannot take a pointer value.
F307	Message	Call of non-function ' <b>variable name</b> '
	Cause	The specified variable name cannot be used as a function name.
F308	Message	Improper use of a typedef name
	Cause	The <b>typedef</b> name has not been used properly.



W309	Message	Unused 'variable name'
	Cause	The specified variable has been declared in the source program but has not been used at all.
W310	Message	'variable name' is assigned a value which is never used
	Cause	The specified variable has been used in an assignment statement but has never been used elsewhere.
F311	Message	Number syntax
	Cause	The constant representation is incorrect.
F312	Message	Illegal octal digit
	Cause	The described octal number contains an illegal character.
F313	Message	Illegal hexadecimal digit
	Cause	The described hexadecimal number contains an illegal character.
F314	Message	Too big constant
	Cause	The described constant is too large to be represented.

F315	Message	Too small constant
	Cause	The described constant is too small to be represented.
F316	Message	Too many character constants
	Cause	The character constant was described with two or more characters.
F317	Message	Empty character constant
	Cause	The character constant in ' ' (single quotes) is null.
F318	Message	No terminated string literal
	Cause	The string literal is not terminated with the closing double quote (").
F320	Message	No null terminator in string literal
	Cause	A NULL character has not been added to mark the end of the string literal.
F321	Message	Compiler limit : too many characters in string literal
	Cause	The number of characters in the string literal exceeded 509.
F322	Message	Ellipsis requires three periods
	Cause	The compiler found ".." but "..." must appear at the end of the parameter list as ellipsis.

F323	Message	Missing 'delimiter'
	Cause	The displayed delimiter is missing.
F324	Message	Too many ) 's
	Cause	Opening and closing brace brackets have not been paired properly.
F325	Message	No terminated comment
	Cause	The closing "*/" is missing from the comment statement.

## Group 4: Error Messages Related to Conversions

W401	Message	Conversion may lose significant digits
	Cause	An attempt was made to convert from long to int.
F402	Message	Incompatible type conversion
	Cause	The type conversion not allowed for an assignment statement was attempted.
F403	Message	Illegal indirection
	Cause	The * operator has been used for an integral type expression.
F404	Message	Incompatible structure type conversion
	Cause	In an assignment statement between structures or to a structure, the variable on the left of the assignment is not of the same type as that on the right.
F405	Message	Illegal lvalue
	Cause	The given value is incorrect as an lvalue.
F406	Message	Cannot modify a const object 'variable name'
	Cause	An attempt was made to modify the variable declared <b>const</b> .
F407	Message	Cannot write for read/only sfr 'sfr name'
	Cause	An attempt was made to write the displayed sfr which is read only.

F408	Message	Cannot read for write/only sfr 'sfr name'
	Cause	An attempt was made to read the displayed sfr which is write only.
F409	Message	Write invalid data for sfr 'sfr name'
	Cause	An attempt was made to write invalid data into the displayed sfr.
W410	Message	Illegal pointer conversion
	Cause	An attempt was made to convert between a pointer type and a type other than the pointer.
W411	Message	Illegal pointer combination
	Cause	Pointers that differ in type have been used together.
W412	Message	Illegal pointer combination in conditional expression
	Cause	Pointers that differ in type have been used in a conditional expression.
W413	Message	Illegal structure pointer combination
	Cause	Pointers to structures that differ in type have been used together.
F414	Message	Expected pointer
	Cause	A pointer is required.

## Group 5: Error Messages Related to Expressions

F501	Message	Expression syntax
	Cause	An syntax error exists in the expression.
F502	Message	Compiler limit : too many parentheses
	Cause	Nesting of parentheses in this expression exceeded 32 levels.
W503	Message	Possible use of 'variable name' before definition
	Cause	The specified variable has been used before assignment of a value to the variable.
W504	Message	Possibly incorrect assignment
	Cause	In an if, while, or do-while statement, the main operator of the conditional expression is an assignment operator.
F507	Message	Expected integral index
	Cause	Only an integral type expression is allowed as the subscript of an array.
W508	Message	Too many actual arguments
	Cause	The number of arguments specified in the function call is more than the number of parameters specified in the argument type list or function definition.

W509	Message	Too few actual arguments
	Cause	The number of arguments specified in the function call is less than the number of parameters specified in the argument type list or function definition.
W510	Message	Pointer mismatch in function 'function name'
	Cause	The given arguments have pointer types different from those specified in the argument type list or function definition.
W511	Message	Different argument types in function 'function name'
	Cause	The types of arguments given by the function call do not match with those specified in the argument type list or function definition.
F512	Message	Function call in norec function
	Cause	An attempt was made to call a function within the norec function.
F513	Message	Illegal structure/union member 'member name'
	Cause	The displayed member has not been defined in the structure or union reference.
F514	Message	Expected structure/union pointer
	Cause	The expression before the -> (arrow) operator must be a pointer to a structure or union, not the name of a structure or union.

F515	Message	Expected structure/union name
	Cause	The expression before the . (dot) operator must be the name of a structure or union, not a pointer to a structure or union.
F516	Message	Zero sized structure 'structure name'
	Cause	The size of the displayed structure is 0.
F517	Message	Illegal structure operation
	Cause	An operator that cannot be used for structures has been specified.
F518	Message	Illegal structure/union comparison
	Cause	Two structures or unions cannot be compared.
F519	Message	Illegal bit field operation
	Cause	An illegal description was made for a bit field.
F520	Message	Illegal use of pointer
	Cause	Only +, -, assignment, relational, indirection (*), or arrow (->) operator can be used for a pointer.
W522	Message	Ambiguous operators need parentheses
	Cause	Two shift, relational, or bitwise logical operators have been described in succession without parentheses.



F523	Message	Illegal bit type operation
	Cause	An attempt was made to execute an illegal operation on a <code>bit</code> type variable.
F524	Message	'&' on constant
	Cause	The address of a constant cannot be obtained.
F525	Message	'&' requires lvalue
	Cause	The & operator can be used only for an expression which is to be assigned to an lvalue.
F526	Message	'&' on register variable
	Cause	The address of a register variable cannot be obtained.
F527	Message	'&' on ignored
	Cause	The address of a bit field or <code>bit</code> type variable cannot be obtained.
W528	Message	'&' is not allowed array/function, ignored
	Cause	The & operator is not allowed for a array name or function name.
F529	Message	Sizeof returns zero
	Cause	The value of the <code>sizeof</code> expression is 0.

F530	Message	Illegal sizeof operand
	Cause	The operand of the <code>sizeof</code> expression must be an identifier or type name.
F531	Message	Disallowed conversion
	Cause	An attempt was made to perform an illegal <code>cast</code> operation.
W532	Message	Pointer on left, needs integral right : 'operator'
	Cause	Because the left operand of this operator is a pointer, its right operand must be an integral value.
F533	Message	Invalid left-or-right operand : 'operator'
	Cause	The left or right operand of this operator is invalid for the operator.
F534	Message	Divide check
	Cause	The divisor of a <code>/</code> (divide) or <code>%</code> (modulo) operation is 0.
F535	Message	Invalid pointer addition
	Cause	Addition between two pointers is not allowed.
F536	Message	Must be integral value addition
	Cause	Only an integral value can be added to a pointer.

F537	Message	Illegal pointer subtraction
	Cause	Subtraction is allowed only between pointers which have the same type.
F538	Message	Illegal conditional operator
	Cause	The condition operator has not been described correctly.
F539	Message	Expected constant expression
	Cause	A constant expression is required.
W540	Message	Constant out of range in comparison
	Cause	One constant partial expression is compared with a value outside the range allowed for the type of the other partial expression.
F541	Message	Function argument has void type
	Cause	The argument of the function is a void type.
F542	Message	Arguments mismatch: 1M byte function '%s
	Cause	An error exists in the argument of the 1M-byte function (applicable to 78K/II only).

## Group 6: Error Messages Related to Statements

F602	Message	Compiler limit : too many characters in logical source line
	Cause	The number of characters of the logical source line exceeded 509.
F603	Message	Compiler limit : too many labels
	Cause	The number of labels exceeded 33.
F604	Message	Case not in switch
	Cause	A case clause has not be described in the correct position in the switch body.
F605	Message	Duplicate case 'label name'
	Cause	The same case label has been described more than once in the switch body.
F606	Message	No constant case expression
	Cause	Other than an integer constant has been described in a case clause.
F607	Message	Compiler limit : too many case labels
	Cause	The number of case labels in the switch statement exceeded 257.
F608	Message	Default not in switch
	Cause	The default label has not been described in the correct position in the switch body.

F609	Message	More than one 'default'
	Cause	Two or more default labels have been described in the <b>switch</b> body.
F610	Message	Compiler limit : block nest level too depth
	Cause	The nesting of blocks exceeded 45 levels.
F611	Message	Inappropriate 'else'
	Cause	ifs and elses have not been paired properly.
F613	Message	Loop entered at top of switch
	Cause	A <b>while</b> , <b>do-while</b> , or <b>for</b> loop has been specified at the end of a <b>switch</b> statement.
F615	Message	Statement not reached
	Cause	A permanent loop exists.
F617	Message	Do statement must have ' <b>while</b> '
	Cause	A <b>while</b> statement must follow the <b>do</b> statement.
F620	Message	Break/continue error
	Cause	The <b>break</b> or <b>continue</b> statement has not been described in the correct position.

F621	Message	Void function 'function name' cannot return value
	Cause	The displayed function has been declared void and thus cannot return a value.
W622	Message	No return value
	Cause	The function which should return a value has not returned any value.
F623	Message	No effective code and data, _ cannot create output file
	Cause	No intermediate statement has been created during syntax analysis.

Group 7: Error Messages Related to Declarations and  
Function Definitions

F701	Message	External definition syntax
	Cause	The function has not been defined properly.
F702	Message	Too many callt functions
	Cause	Too many callt functions have been declared.
F703	Message	Function has illegal storage class
	Cause	The function has been specified with an invalid storage class.
F704	Message	Function returns illegal type
	Cause	The type of the return value of the function is illegal.
F705	Message	Too many parameters in noauto or norec function
	Cause	Too many parameters have been specified for the noauto or norec function.
F706	Message	Parameter list error
	Cause	An error exists in the function parameter list.
F707	Message	Not parameter 'character string'
	Cause	The character string which is not a parameter has been declared in the function definition.

F710	Message	Illegal storage class
	Cause	<code>auto</code> or <code>register</code> variables have been declared outside the function.
F711	Message	Undeclared ' <code>variable name</code> ' in function ' <code>function name</code> '
	Cause	The variable used in the function has not been declared.
F712	Message	Declaration syntax
	Cause	A syntax error exists in the declaration statement.
F713	Message	Redefined ' <code>variable name</code> '
	Cause	The same variable name has been defined more than once.
F714	Message	Too many register variables
	Cause	Too many <code>register</code> variables have been declared.
F715	Message	Too many <code>sreg</code> variables
	Cause	Too many <code>sreg</code> variables have been declared.
F716	Message	Not allowed automatic data in <code>noauto</code> function
	Cause	No automatic variable can be used in the <code>noauto</code> function.



F717	Message	Too many automatic data in norec function
	Cause	Too many automatic variables have been declared in the <b>norec</b> function.
F718	Message	Too many bit type variables
	Cause	Too many bit type variables have been used.
F719	Message	Illegal use of type
	Cause	The type name used is illegal.
F720	Message	Illegal void type for 'identifier'
	Cause	The identifier has been declared <b>void</b> .
F721	Message	Illegal type for register declaration
	Cause	The <b>register</b> declaration is not allowed for the data type of the variable.
	Action	The compiler ignores the <b>register</b> declaration and continues processing.
F722	Message	Illegal type for sreg declaration
	Cause	The <b>sreg</b> declaration is not allowed for the data type of the variable.
F723	Message	Illegal type for parameter in noauto or norec function
	Cause	The parameter type of the <b>noauto</b> or <b>norec</b> function is illegal (too large).

F724	Message	Structure redefinition
	Cause	The same structure has been defined again.
F725	Message	Illegal zero sized structure member
	Cause	The size of the area for structure members must be 1 or more.
F726	Message	Function cannot be structure/union member
	Cause	The function cannot be a structure or union member.
F727	Message	Unknown size structure/union 'name'
	Cause	The size of the structure or union has not been defined.
F728	Message	Compiler limit : too many structure/union members
	Cause	The number of structure or union members exceeded 127.
F729	Message	Compiler limit : structure/union nesting
	Cause	The nesting of structures or unions exceeded 15 levels.
F730	Message	Bit field outside of structure
	Cause	A bit field has been declared outside the structure.

F731	Message	Illegal bit field type
	Cause	Other than an integer type has been specified as the type of the bit field.
F732	Message	Too long bit field size
	Cause	The number of bits specified in the bit field declaration exceeded the limit value for the type.
F733	Message	Negative bit field size
	Cause	The number of bits specified in the bit field declaration is a negative value.
F734	Message	Illegal enumeration
	Cause	A syntax error exists in the enumerated type declaration.
F735	Message	Illegal enumeration constant
	Cause	The enumerated constant is illegal.
F736	Message	Compiler limit : too many enumeration constants
	Cause	The number of enumerated constants exceeded 127.
F737	Message	Undeclared structure/union/enum tag
	Cause	No tag has been declared for the structure, union, or enumeration.

F738	Message	Compiler limit : too many pointer modifying
	Cause	The number of indirection (*) operators in the pointer definition exceeded 12.
F739	Message	Expected constant
	Cause	In the array declaration, a variable has been used for the subscript.
F740	Message	Negative subscript
	Cause	A negative value has been specified as the size of an array.
F741	Message	Unknown size array 'array name'
	Cause	The size of the specified array is unknown.
F742	Message	Compiler limit : too many array modifying
	Cause	Array declarations exceeded 12 dimensions.
F743	Message	Array element type cannot be function
	Cause	The array element type cannot be a function.
F744	Message	Zero sized array 'array name'
	Cause	The number of elements of the defined array is 0.
F745	Message	Expected function prototype
	Cause	A function prototype declaration is required.

F747	Message	Function prototype mismatch
	Cause	An error exists in the function prototype declaration.
F748	Message	A function is declared as a parameter
	Cause	A function has been declared as a parameter.
F749	Message	Unused parameter ' <b>parameter name</b> '
	Cause	The displayed parameter has not been used.
F750	Message	Initializer syntax
	Cause	A syntax error exists in the declaration that contain initializers.
F751	Message	Illegal initialization
	Cause	The constant for initialization does not match the variable in type.
F752	Message	Undeclared initializer name ' <b>name</b> '
	Cause	The displayed initializer name has not been declared.
F753	Message	Cannot initialize static with automatic
	Cause	The <b>static</b> variable cannot be initialized with an <b>auto</b> variable.

F754	Message	Cannot initialize array in function 'function name'
	Cause	Arrays inside the function cannot be initialized.
F755	Message	Cannot initialize structure/union in function 'function name'
	Cause	Structures or unions inside the function cannot be initialized.
F756	Message	Too many initializers 'array name'
	Cause	The number of initialization values is more than the number of elements of the declared array.
F757	Message	Too many structure initializers
	Cause	The number of initialization values is more than the number of members of the declared structure.
F758	Message	Cannot initialize a function 'function name'
	Cause	The function cannot be initialized.
F759	Message	Compiler limit : initializers too deeply nested
	Cause	The nesting of initializers exceeded the limit value.
F760	Message	Not including floating function
	Cause	Floating-point arithmetic is not supported.

F770	Message	Parameters are not allowed for interrupt function
	Cause	No parameter is allowed for the interrupt function.
W708	Message	Already declared symbol ' <b>variable name</b> '
	Cause	The same variable name has already been declared.

## Group 8: Error Messages Related to Preprocessor Directives

F801	Message	Undefined control
	Cause	A word which begins with # cannot be recognized as a keyword.
F802	Message	Illegal preprocess directive
	Cause	An error exists in the preprocessor directive.
F803	Message	Unexpected non-whitespace before preprocess directive
	Cause	Other than a white-space character precedes the preprocessor directive.
W804	Message	Unexpected characters following 'preprocess directive' directive newline expected
	Cause	Unwanted characters follow the preprocessor directive.
F805	Message	Misplaced else or elif
	Cause	#if, #ifdef, or #ifndef directive has not been properly paired with #else or #elif directive.
F806	Message	Misplaced endif
	Cause	#if, #ifdef, or #ifndef directive has not been properly paired with #endif directive.



F807	Message	Compiler limit : too many conditional inclusion nesting
	Cause	The nesting of conditional inclusions exceeded 255 levels.
F810	Message	Cannot find include file 'file name'
	Cause	The specified Inclusion file cannot be found.
F811	Message	Too long file name 'file name'
	Cause	The specified filename is too long.
F812	Message	Include directive syntax
	Cause	In the #include directive, the filename has not been enclosed in double quotes ( " ") or angle brackets ( < > ).
F813	Message	Compiler limit : too many include nesting
	Cause	The nesting of Inclusion files exceeded 8 levels.
F814	Message	Illegal macro name
	Cause	An error, exists in the macro name description.
W816	Message	Redefined macro name 'macro name'
	Cause	The macro name has already been defined.

W817	Message	Redefined system macro name 'macro name'
	Cause	The system macro name has already been defined.
F818	Message	Redeclared parameter in macro 'macro name'
	Cause	The same identifier appeared again in the parameter list of the macrodefinition.
W819	Message	Mismatch number of parameters 'macro name'
	Cause	The number of parameters defined with the <code>#define</code> directive differs from the number of parameters to be referenced.
F821	Message	Illegal macro parameter 'macro name'
	Cause	In the function type macro, names in ( ) have not been described properly.
F822	Message	Missing ) 'macro name'
	Cause	In the function type macro, the closing right parenthesis cannot be found within the same <code>#define</code> directive line.
F823	Message	Too long macro expansion 'macro name'
	Cause	The actual parameter length is too long in the expansion of the macro.
F824	Message	Compiler limit : too long macro name 'macro name'
	Cause	The macro name is too long.

W825	Message	Macro recursion 'macro name'
	Cause	The <b>#define</b> definition is recursive.
F826	Message	Compiler limit : too many macro defines
	Cause	The number of macrodefinitions exceeded 1,024.
F827	Message	Compiler limit : too many macro parameters
	Cause	The number of parameters per macrodefinition or macrocall exceeded 31.
F828	Message	Not allowed <b>#undef</b> for system macro name 'macro name'
	Cause	The macro name specified with the <b>#undef</b> directive is a system macro name which cannot be undefined.
W829	Message	Unrecognized pragma 'character string'
	Cause	This character string is not supported by the C compiler.
F830	Message	No chip specifier : <b>#pragma pc ( )</b>
	Cause	No processor type has been specified in the <b>#pragma pc ( )</b> directive.
F831	Message	Illegal chip specifier : ' <b>#pragma pc (processor type)</b> '
	Cause	An error exists in the processor type specification with <b>#pragma pc ( )</b> directive.

F832	Message	Duplicated chip specifier
	Cause	The duplicated processor type has been specified.
F833	Message	Expected #asm
	Cause	A #asm statement is required.
F834	Message	Expected #endasm
	Cause	A #endasm statement is required at the end of the #asm statement.
F835	Message	Too many characters in assembler source line
	Cause	The assembler source line contains too many characters.
W836	Message	Expected assembler source
	Cause	No assembler source statement exists in the #asm-#endasm block.
W837	Message	Output assembler source file, not object file
	Cause	An assembler source module file has been output in place of an object module file.
W838	Message	Duplicated pragma VECT '%s'
	Cause	#pragma VECT 'character string' has been specified in duplication.

F839	Message	Unrecognized pragma VECT '%s'
	Cause	Unrecognized #pragma VECT 'character string' exists.
F840	Message	Undefined interrupt function '%s'
	Cause	The interrupt function '%s' has not been declared.
F841	Message	Unrecognized pragma TABLE '%s'
	Cause	Unrecognized #pragma TABLE 'character string' exists.

## Group 9: Error Messages Related to I/O and Optimization

A901	Message	File I/O error
	Cause	A physical I/O error has occurred during data input/output to or from a file.
A902	Message	Cannot open 'file name'
	Cause	An error has occurred while opening the file.
A903	Message	Cannot open overlay file 'file name'
	Cause	An error has occurred while opening the overlay file.
A904	Message	Cannot open temp
	Cause	An error has occurred while opening a temporary file for input.
A905	Message	Cannot create 'file name'
	Cause	Creation of the specified file failed.
A906	Message	Cannot create temp
	Cause	Creation of a temporary file for output failed.
A907	Message	No available data block
	Cause	Creation of a temporary file failed due to the insufficient file space capacity of the drive.

A908	Message	No available directory space
	Cause	Creation of a temporary file failed due to the insufficient directory area of the drive.
A909	Message	R/O : read/only disk
	Cause	Creation of a temporary file failed due to the read-only attribute of the drive.
A910	Message	R/O file : read/only, file opened read/only mode
	Cause	<p>An error has occurred while writing a temporary file by either of the following two reasons:</p> <ol style="list-style-type: none"> <li>1. The file which has the same name as the temporary file already exists on the drive and the file has read-only attribute.</li> <li>2. A temporary file for output has been opened with read-only attribute due to internal conflict.</li> </ol>
A911	Message	Reading unwritten data, no available directory space
	Cause	<p>An I/O error has occurred by either of the following two reasons:</p> <ol style="list-style-type: none"> <li>1. An attempt was made to input data beyond the EOF.</li> <li>2. Creation of a temporary file failed due to the insufficient directory area of the drive.</li> </ol>

A912	Message	Write error on temp
	Cause	An error has occurred while writing a temporary file.
A913	Message	Requires MS-DOS V2.11 or greater
	Cause	The OS is not MS-DOS (V2.11 or higher).
A914	Message	Insufficient memory in host machine
	Cause	The C compiler cannot be started up due to insufficient memory space in the host machine.

The following messages are applicable to the CC78K3 only:

W915	Message	Asm statement found. optimize skip this function
	Cause	The optimization of this function is discontinued, because an ASM statement was found.
W916	Message	Too many basic blocks for optimizing
	Cause	Basic blocks are too many to execute optimization.
W917	Message	Too many nesting for optimizing
	Cause	Functions are too deeply nested to execute optimization.



W918	Message	Too many function arguments for optimizing
	Cause	The arguments of a function are too many to execute optimization.
W919	Message	Too many function nesting for optimizing
	Cause	Functions are too deeply nested to execute optimization.
W920	Message	Too many symbols for optimizing
	Cause	Symbols are too many to execute optimization.
W921	Message	Too many value_ no for optimizing
	Cause	Value numbers are too many to execute optimization.

## APPENDIX A. SAMPLE PROGRAMS

## A.1 C Source Module File

```

#define TRUE      1
#define FALSE     0
#define SIZE      200

char    mark[SIZE+1];

main()
{
    int i, prime, k, count;

    count = 0;

    for ( i = 0 ; i <= SIZE ; i++)
        mark[i] = TRUE;
    for ( i = 0 ; i <= SIZE ; i++) {
        if (mark[i]) {
            prime = i + i + 3;
            printf("%6d", prime);
            count++;
            if ((count%8) == 0) putchar('\n');
            for ( k = i + prime ; k <= SIZE ; k += prime)
                mark[k] = FALSE;
        }
    }
    printf("\n%d primes found.", count);
}

printf(s, i)
char *s;
int i;
{
    int j;
    char *ss;

    j = i;
    ss = s;
}

putchar(c)
char c;
{
    char d;
    d = c;
}

```

## A.2 Execution Example

```
A>cc78K3 -c310 sampleYprime.c -a -p -x -e
```

```
78K/III Series C Compiler Vx.xx [xx xxx xx]  
Copyright (C) NEC Corporation xxxx
```

```
SAMPLEYPRIME.C(18) : W745 Expected function prototype  
SAMPLEYPRIME.C(20) : W745 Expected function prototype  
SAMPLEYPRIME.C(26) : W622 No return value  
SAMPLEYPRIME.C(37) : W622 No return value  
SAMPLEYPRIME.C(44) : W622 No return value  
Compilation complete. 0 error(s) and 5 warning(s) found.
```

## A.3 Output Lists

## (1) Assembler Source Module File

```
; 78K/111 Series C Compiler V1.10 Assembler Source
;                                     Date:xx xxx xxxx Time:xx:xx:xx
; Command      : -c310 YsampleYprime.c -a -p -x -e
; In-file      : YSAMPLEYPRIME.C
; Asm-file     : PRIME.ASM
; Para-file    :
```

```
$PROCESSOR(310)
$NODEBUG
```

```
NAME      PRIME
EXTRN     @@isrem
PUBLIC    _mark
PUBLIC    _main
PUBLIC    _printf
PUBLIC    _putchar

@@CODE    CSEG
; line    5
; line    8
_main:
    push    hl
    movw    ax, sp
    subw    ax, #08H
    movw    hl, ax
    movw    sp, ax
; line    11
    movw    ax, #00H ; 0
    mov     [hl+1], a      ; count
    xch     a, x
    mov     [hl], a      ; count
; line    13
    movw    ax, #00H ; 0
    mov     [hl+7], a      ; i
    xch     a, x
    mov     [hl+6], a      ; i
L0003:
    mov     a, [hl+6]      ; i
    xch     a, x
    mov     a, [hl+7]      ; i
    cmpw    ax, #0C8H      ; 200
    bgt     $L0004
; line    14
    movw    up, #_mark
    mov     a, [hl+6]      ; i
    xch     a, x
    mov     a, [hl+7]      ; i
    addw    up, ax
    mov     a, #01H ; 1
    mov     [up], a
L0005:
    mov     a, [hl+6]      ; i
    xch     a, x
    mov     a, [hl+7]      ; i
    addw    ax, #01H ; 1
    mov     [hl+7], a      ; i
    xch     a, x
    mov     [hl+6], a      ; i
```

```

        br      $L0003
L0004:
; line    15
        movw    ax, #00H ; 0
        mov     [hl+7], a      ; i
        xch     a, x
        mov     [hl+6], a      ; i
L0006:
        mov     a, [hl+6]      ; i
        xch     a, x
        mov     a, [hl+7]      ; i
        cmpw    ax, #0C8H      ; 200
        ble     $$+6
        br      !L0007
; line    16
        movw    up, #_mark
        mov     a, [hl+6]      ; i
        xch     a, x
        mov     a, [hl+7]      ; i
        addw    up, ax
        mov     a, [up]
        xch     a, x
        mov     a, #0FFH ; 255
        bt      x.7, $L0011
        inc     a
L0011:
        cmpw    ax, #00H ; 0
        bne     $$+5
        br      !L0009
; line    17
        mov     a, [hl+6]      ; i
        xch     a, x
        mov     a, [hl+7]      ; i
        movw    bc, ax
        mov     a, [hl+6]      ; i
        xch     a, x
        mov     a, [hl+7]      ; i
        addw    ax, bc
        addw    ax, #03H ; 3
        mov     [hl+5], a      ; prime
        xch     a, x
        mov     [hl+4], a      ; prime
; line    18
        mov     a, [hl+4]      ; prime
        xch     a, x
        mov     a, [hl+5]      ; prime
        push    ax
        movw    ax, #L0012
        push    ax
        call    !_printf
        movw    ax, sp
        addw    ax, #04H ; 4
        movw    sp, ax
; line    19
        mov     a, [hl] ; count
        xch     a, x
        mov     a, [hl+1]      ; count
        addw    ax, #01H ; 1
        mov     [hl+1], a      ; count
        xch     a, x
        mov     [hl], a ; count
; line    20
        movw    rp2, #08H      ; 8
        mov     a, [hl] ; count
        xch     a, x
        mov     a, [hl+1]      ; count
        movw    bc, ax

```

```

        call    !@@isrem
        movw    ax, bc
        cmpw    ax, #00H ; 0
        bne     $L0013
        movw    ax, #0AH ; 10
        push    ax
        call    !_putchar
        pop     ax
L0013:
L0014:
; line      21
        mov     a, [hl+4]      ; prime
        xch     a, x
        mov     a, [hl+5]      ; prime
        movw    bc, ax
        mov     a, [hl+6]      ; i
        xch     a, x
        mov     a, [hl+7]      ; i
        addw    ax, bc
        mov     [hl+3], a      ; k
        xch     a, x
        mov     [hl+2], a      ; k
L0015:
        mov     a, [hl+2]      ; k
        xch     a, x
        mov     a, [hl+3]      ; k
        cmpw    ax, #0C8H      ; 200
        bgt     $L0016
; line      22
        movw    up, #_mark
        mov     a, [hl+2]      ; k
        xch     a, x
        mov     a, [hl+3]      ; k
        addw    up, ax
        mov     a, #00H ; 0
        mov     [up], a
L0017:
        mov     a, [hl+4]      ; prime
        xch     a, x
        mov     a, [hl+5]      ; prime
        movw    bc, ax
        mov     a, [hl+2]      ; k
        xch     a, x
        mov     a, [hl+3]      ; k
        addw    ax, bc
        mov     [hl+3], a      ; k
        xch     a, x
        mov     [hl+2], a      ; k
        br      $L0015
L0016:
L0009:
L0010:
L0008:
        mov     a, [hl+6]      ; i
        xch     a, x
        mov     a, [hl+7]      ; i
        addw    ax, #01H ; 1
        mov     [hl+7], a      ; i
        xch     a, x
        mov     [hl+6], a      ; i
        br      !L0006
L0007:
; line      25
        mov     a, [hl] ; count
        xch     a, x
        mov     a, [hl+1]      ; count
        push    ax

```

```

        movw    ax, #L0018
        push    ax
        call    !_printf
        movw    ax, sp
        addw    ax, #04H ; 4
        movw    sp, ax
        movw    ax, hl
        addw    ax, #08H
        movw    sp, ax
        pop     hl
        ret
; line      31
_printf:
        push    hl
        movw    ax, sp
        subw    ax, #04H
        movw    hl, ax
        movw    sp, ax
; line      35
        mov     a, [hl+10]      ; i
        xch     a, x
        mov     a, [hl+11]      ; i
        mov     [hl+3], a       ; j
        xch     a, x
        mov     [hl+2], a       ; j
; line      36
        mov     a, [hl+8]       ; s
        xch     a, x
        mov     a, [hl+9]       ; s
        mov     [hl+1], a       ; ss
        xch     a, x
        mov     [hl], a        ; ss
        movw    ax, hl
        addw    ax, #04H
        movw    sp, ax
        pop     hl
        ret
; line      41
_putchar:
        push    hl
        movw    ax, sp
        subw    ax, #02H
        movw    hl, ax
        movw    sp, ax
; line      43
        mov     a, [hl+6]       ; c
        mov     [hl+1], a       ; d
        movw    ax, hl
        addw    ax, #02H
        movw    sp, ax
        pop     hl
        ret

@@CNST  CSEG
L0012:  DB      '%6d'
        DB      00H
L0018:  DB      0AH
        DB      '%d primes found.'
        DB      00H

@@R_DATA CSEG
        DB      (201)

@@DATA  DSEG
_mark:  DS      (201)
        END

```

## (2) Preprocess List File

```

/*
78K/111 Series C Compiler V1.10 Preprocess List      Date:xx xxx xxxx Page:  1

Command   : -c310 YsampleYprime.c -a -p -x -e
In-file   : YSAMPLEYPRIME.C
PPL-file  : PRIME.PPL
Para-file :
*/

1 : #define TRUE      1
2 : #define FALSE     0
3 : #define SIZE      200
4 :
5 : char    mark[SIZE+1];
6 :
7 : main()
8 : {
9 :     int i, prime, k, count;
10 :
11 :     count = 0;
12 :
13 :     for ( i = 0 ; i <= SIZE ; i++)
14 :         mark[i] = TRUE;
15 :     for ( i = 0 ; i <= SIZE ; i++) {
16 :         if (mark[i]) {
17 :             prime = i + i + 3;
18 :             printf("%6d", prime);
19 :             count++;
20 :             if((count%8) == 0) putchar('\n');
21 :             for ( k = i + prime ; k <= SIZE ; k += prime)
22 :                 mark[k] = FALSE;
23 :         }
24 :     }
25 :     printf("\n%d primes found.", count);
26 : }
27 :
28 : printf(s,i)
29 : char *s;
30 : int i;
31 : {
32 :     int j;
33 :     char *ss;
34 :
35 :     j = i;
36 :     ss = s;
37 : }
38 :
39 : putchar(c)
40 : char c;
41 : {
42 :     char d;
43 :     d = c;
44 : }

```



## (3) Cross-reference List File

78K/III Series C Compiler V1.10 Cross reference List Date:xx xxx xxxx Page: 1

Command : -c310 YsampleYprime.c -a -p -x -e  
 In-file : YSAMPLEYPRIME.C  
 Xref-file : PRIME.XRF  
 Para-file :

ATTRIB	MODIFY	TYPE	SYMBOL	DEFINE	REFERENCE
EXTERN		array	mark	5	14 16 22
EXTERN		func	main	7	
AUTO1		int	i	9	13 13 13 14
15	15	15	16	17	17
					21
AUTO1		int	prime	9	17 18 21 21
AUTO1		int	k	9	21 21 22
AUTO1		int	count	9	11 19 20 25
EXTERN		func	printf	28	18 25
EXTERN		func	putchar	39	20
PARAM		pointer	s	29	36
PARAM		int	i	30	35
AUTO1		int	j	32	35
AUTO1		pointer	ss	33	36
PARAM		char	c	40	43
AUTO1		char	d	42	43
		#define	TRUE	1	14
		#define	FALSE	2	22
		#define	SIZE	3	5 13 15 21

(4) Error List File

```
SAMPLEYPRIME.C( 18) : W745 Expected function prototype
SAMPLEYPRIME.C( 20) : W745 Expected function prototype
SAMPLEYPRIME.C( 26) : W622 No return value
SAMPLEYPRIME.C( 37) : W622 No return value
SAMPLEYPRIME.C( 44) : W622 No return value
```

Compilation complete, 0 error(s) and 5 warning(s) found.



## APPENDIX B. LIST OF HINTS ON USE

## o At Compile Time

Item No.	Point to Bear in Mind
1	<p>About MS-DOS (PC-DOS) Based System:</p> <p>When starting up this C compiler, the parameter FILES must be set to 25 or more in the environment setting file "CONFIG.SYS" of MS-DOS (PC-DOS).</p>
2	<p>About Option Specification:</p> <ul style="list-style-type: none"> <li>o If two or more options which are not allowed to be specified with any other options are specified at the same time, whichever you specified last will take precedence over the preceding options.</li> <li>o With the -C option, the processor type cannot be omitted. If this parameter is omitted, an Abort error will result. If the processor type of the target device is not to be specified with the -C option, be sure to specify the processor type in the C source module file by using the <code>#pragma pc (processor type)</code> directive.</li> </ul> <p>Also note that if the processor type specified with the -C option differs from that specified in the C source module file, the C compiler will accept the one specified with the -C option as valid and output a warning message.</p> <ul style="list-style-type: none"> <li>o If the -- (HELP) option is specified, all other option specifications will become invalid.</li> </ul>

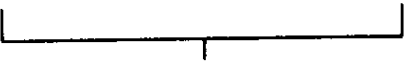
## o At Compile Time (contd)

Item No.	Point to Bear in Mind
3	About Output Destination of File: Only a disk type file can be specified as the output destination of object module files.
4	About Error Messages: If a syntax error exists in a filename specification, the filename will be added to the error message. If a device type file is specified incorrectly in the command line, the specified character string will be output as is. In cases other than above, the drive name, pathname, and filename extension (file type) will always be added to each filename.

## o At Assembly Time

Item No.	Point to Bear in Mind
1	About Assembly Language Descriptions in C Source Program: If any descriptions in the assembly language exist in a C source program, create a load module file by executing the C compiler, Assembler, and Linker in the order named. When assembling an object module file output by the C compiler, the assembler option -NCA must be specified to distinguish symbol names written in uppercase letters from those in lowercase letters.

## o At Linking Time

Item No.	Point to Bear in Mind
1	<p>About ROMable Processing:</p> <p>With the CC78Kn, the C compiler's default assumption at Compile time is ROMable. Therefore, when linking an object module file compiled based on this default assumption, the object module file must be linked with the applicable start-up routine for ROMable processing.</p> <p>Start-up routine: csxxxr.rel romxxx.rel</p> <p>Example: With uPD78320</p> <p>A&gt;lk78K3 <u>cs320r.rel</u> test.rel <u>rom320.rel</u></p> <div style="text-align: center;">  <p>Required for ROMable processing</p> </div> <p>test.rel: Object module file of user program</p>
2	<p>About Option for Stack Solving Symbol Creation Specification (-S):</p> <p>With the CC78Kn, the user cannot reserve a stack area. To secure the stack area, be sure to specify the linker option -S.</p>

## o At Linking Time (contd)

Item No.	Point to Bear in Mind
3	<p>About Option for Directive File Specification (-D):</p> <p>The following error message may be output if a linking operation is executed in the memory area defined by default assumption:</p> <p>*** ERROR F206 Segment 'xxx' can't allocate to memory-ignored</p> <p>[Cause]</p> <p>The Linker cannot allocate the specified segment to the memory area (internal RAM area) defined by default assumption because of insufficient space.</p> <p>[Actions]</p> <p>As a recommended action for this, the following three major steps should be observed.</p> <ol style="list-style-type: none"><li>1. Check the size of the segment that cannot be allocated.</li><li>2. Based on the segment size checked, enlarge the size of an area in which segments in the directive file are allocated.</li><li>3. Link the object module file by specifying the linker option -D (for directive file specification).</li></ol>

## o At Linking Time (contd)

Item No.	Point to Bear in Mind
3 (contd)	<p>In the step 1 above, the method of checking the segment size differs as outlined below depending on the type of the segment indicated by the error message.</p> <p>(1) With segments automatically generated at Compile time</p> <p>Segment names  @@CALT, @@CNST, @@CODE, @@R_DATA, @@DATA, @@R_INIT, @@INIT, @@R_DATS, @@DATS, @@R_INIS, @@INIS, @@BITS</p> <p>Step 1: In the directive file, reserve a sufficiently large memory area for allocation of the segment indicated by the error message.</p> <p>Step 2: Check the size of the segment from the map file created as a result of linking the object module.</p> <p>(2) With user-created segments</p> <p>Check the size of the segment indicated by the error message from the assembly list file.</p> <p>REMARKS: Use the standard directory file of each target device which is stored as a sample file in the directory named "LIB" in the floppy disk 1.</p>



Item No.	Point to Bear in Mind
4	<p data-bbox="435 212 1227 289">About Option for Library File Specification (-B):</p> <ul data-bbox="435 304 1318 625" style="list-style-type: none"> <li>o When linking object files compiled with the CC78Kn, specify the common library file supplied with the C compiler as shown in Example 1. However, with the processor types 310, 312, 310A, and 312A in the 78K/III series, specify the library file applicable to each processor type as shown in Example 2.</li> </ul> <p data-bbox="435 640 878 674">Example 1: With uPD78320</p> <pre data-bbox="456 735 1284 894">A&gt;lk78K3  cs320r.rel test.rel rom320.rel                                      -s <u>-bcl3com.lib</u> test.rel: Object module file of user program</pre> <p data-bbox="435 953 899 987">Example 2: With uPD78312A</p> <pre data-bbox="456 1050 1300 1209">A&gt;lk78K3  cs312ar.rel test.rel rom312a.rel                                      -s <u>-bcl312a.lib</u> test.rel: Object module file of user program</pre> <ul data-bbox="435 1224 1318 1638" style="list-style-type: none"> <li>o When sfr access check is specified with the option -L (option for execution-time error check specification) at compile time, also specify the library file CLxxx.LIB as shown by ~~~~ in Example 3. With the processor types 310, 312, 310A, and 312A in the 78K/III series, this error check library file specification is not required (same as Example 2).</li> </ul> <p data-bbox="435 1654 899 1688">Example 3: With uPD78320</p> <pre data-bbox="456 1749 1292 1908">A&gt;lk78K3  cs320r.rel test.rel rom320.rel                                      -s -bcl3com.lib <u>-bcl320.lib</u> test.rel: Object module file of user program</pre>

## APPENDIX C. LIST OF COMPILER OPTIONS

Item	Classification	Description Format	Function	Relation with Other Options	Default Assumption	See Page
1	Option for processor type specification	-C X (X: processor type)	Specifies the processor type of the target device.	Independent	None (Cannot be omitted)	5-8
2	Options for object module file creation specification	-O [filename]	Specifies the output of an object module file.	If -O and -NO are specified at same time, whichever you specified later takes precedence.	-O[input filename .REL]	5-12
		-NO	Specifies suppression of object module file output.			
3	Options for symbol name length specification	-S	Specifies extension of symbol name length to max. 30 characters.	Independent	-NS	5-14
		-NS	Specifies symbol name length as 8 characters max.			
4	Options for symbol name upper-/lower-case specification	-CA	Specifies that symbol names in uppercase letters are not to be distinguished from those in lowercase letters.	Independent	-NCA	5-17
		-NCA	Specifies that symbol names in uppercase letters are not to be distinguished from those in lowercase letters.			

Item	Classification	Description Format	Function	Relation with Other Options	Default Assumption	See Page
5	Options for ROMable object file creation specification	-R	Specifies creation of ROMable object module file.	If neither object module nor assembler source is to be output, this option will become invalid.	-R	5-20
		-NR	Specifies creation of non-ROMable object module file.			
6	Options for optimization specification	-Q[X] (X: optimization type)	Specifies that optimization is to be performed.	Ditto.	-NQ	5-24
		-NQ	Specifies that optimization is not to be performed.			
7	Options for debug information output specification	-G	Specifies output of symbol information for debugging to object module file.	Ditto.	-NG	5-27
		-NG	Specifies suppression of debug information output to object module file.			
8	Options for execution-time error check specification	-L[X] (X: error check type)	Specifies addition of execution-time error check library to object module file.	Ditto.	-NL	5-29
		-NL	Specifies suppression of error check library output to object module file.			

Item	Classification	Description Format	Function	Relation with Other Options	Default Assumption	See Page
9	Options for preprocess list file creation specification	-P[filename]	Specifies output of a preprocess list file.	Independent	-NP	5-32
		-NP	Specifies suppression of preprocess list file output.			
		-K[X] (x: process type)	Specifies type of process required for the preprocess list to be output.	If -P option is not specified at same time or if -NP is specified at same time, -K will become invalid.	-KFLN	5-34
		-NK	Specifies that no special process is required for the preprocess list.			
10	Options for preprocessing specification	-D macro name=[definition name][,name[=definition]] ...	Specifies that macro definitions are to be executed just the same as <b>#define</b> statements.	If same macro name is specified by -D and -U options, whichever you specified later will take precedence.	Accepts only macro definitions within C source as valid.	5-38
		-ND	Specifies that -D option is to be invalidated.			
		-U name[,name] ...	Specifies that macrodefinitions are to be undefined just the same as <b>#undef</b> statements.	Ditto.	Accepts macro definitions specified by -D as valid.	5-40
		_NU	Specifies that -U option is to be invalidated.			

Item	Classification	Description Format	Function	Relation with Other Options	Default Assumption	See Page
10	Options for preprocessing specification (contd)	-I directory[, directory] ...	Specifies input of Inclusion file(s) specified by <b>#include</b> directive from specified directory.	Independent	Directory containing directory source file specified by INC78Kn (n=0,2,3).	5-42
11	Options for assembler module file creation specification	-A[filename]	Specifies output of assembler source module file to file.	If -A and -SA are specified at same time, -SA option will become invalid.	-NA	5-44
		-NA	Specifies suppression of assembler source module file output to file.			
		-SA[filename]	Specifies output of assembler source module file with C source added to it.	Ditto.	-NSA	5-47
		-NSA	Specifies suppression of C source output to assembler module file.			
12	Options for error list file creation specification	-E[filename]	Specifies output of error list file to file.	If -W0 option is specified at same time, no warning message will be output.	-NE	5-50
		-NE	Specifies suppression of error list file output.			
		-SE[filename]	Specifies output of error list file with C source added to it.	Ditto.	-NSE	5-52
		-NSE	Specifies suppression of C source output to error list file.			

Item	Classification	Description Format	Function	Relation with Other Options	Default Assumption	See Page
13	Options for crossreference list creation specification	-X [filename]	Specifies output of cross-reference list to file.	Independent	-NX	5-55
		-NX	Specifies suppres- sion of cross- reference list output to file.			
14	Options for list format specification	-LW no. of columns	Specifies no. of columns per line of a list file.	If no list file is specified, this option become invalid.	-LW132 (-LW80: console output)	5-57
		-LL no. of lines	Specifies no. of lines to be printed per page of a list.	Ditto.	-LL66 (No page ejection: console output)	5-59
		-LT no. of columns	Specifies no. of columns for tabulation.	Ditto.	-LT8	5-62
		-LF	Specifies addition of formfeed code to the end of list file.	Ditto.	None	5-65

Item	Classification	Description Format	Function	Relation with Other Options	Default Assumption	See Page
15	Option for warning output specification	-W[level]	Specifies output of warning messages to the console.	If -E or -SE option is specified with -W, warning messages will also be output to error list file.	-W1	5-67
16	Options for execution status output specification	-V	Specifies output of execution status to the console.	Independent	-NV	5-69
		-NV	Specifies suppres- sion of execution status output to the console.			
17	Option for parameter file input specification	-F filename	Specifies input of compiler options or input filename from specified parameter file.	Independent	Allows input of options or filenames from only command line.	5-71

Item	Classification	Description Format	Function	Relation with Other Options	Default Assumption	See Page
18	Option for temporary file creation path specification	-T directory	Specifies creation of a temporary file on a specified path.	Independent	Drive or directory specified by TMP or current drive or directory if TMP is not speci- fied.	5-73
19	Option for HELP message output specification	--	Specifies output of HELP message to the console.	If this option is specified, all the other options will be ignored.	No HELP message will be output.	5-75





**Phase-out/Discontinued**

**NEC**