RENESAS ツールニュース

RENESAS TOOL NEWS 2011年06月01日：110601/tn6

R8C, M16Cファミリ用Cコンパイラパッケージ ご使用上のお願い

R8C, M16Cファミリ用Cコンパイラパッケージの使用上の注意事項を連絡します。

1. 引数に浮動小数点定数を使用する場合の注意事項
2. static指定子を使用した関数を割り込み関数として定義する場合の注意事項
3. 関数内で関数または変数をextern宣言する場合の注意事項
4. 共用体メンバに連続して定数を書き込む場合の注意事項
5. sizeof演算子に配列名を使用する場合の注意事項
6. 浮動小数点定数のゼロとNaNとを比較する場合の注意事項
7. K&R形式(旧形式)関数の引数にfloat型を使用する場合の注意事項
8. K&R形式(旧形式)関数の引数に_Bool型を使用する場合の注意事項
9. switch文の制御式の型にsigned charを使用する場合の注意事項

---

1. 引数に浮動小数点定数を使用する場合の注意事項

1.1 該当製品

- R32Cシリーズ用Cコンパイラパッケージ
  V.1.01 Release 00 ～ V.1.02 Release 01A
- M32Cシリーズ用Cコンパイラパッケージ (M3T-NC308WA)
  V.1.00 Release 1 ～ V.5.42 Release 00A
- M16Cシリーズ, R8Cファミリ用Cコンパイラパッケージ (M3T-NC30WA)
  V.2.00 Release 1 ～ V.5.45 Release 01

1.2 内容

関数の引数に負の浮動小数点定数を符号無し整数型にキャストする式を
記述すると、以下の警告が出力されキャスト後の値がゼロになります。

```
[Warning(ccom):sample.c,line 6] underflow in floating value
  converting to integer
    ===>    int j = func((unsigned int)-1.0);
```

1.3 発生条件

以下の条件をすべて満たす場合に発生します。

(1) 関数の引数が、負の浮動小数点型定数である。
　注： この定数には最適化によって定数に置き換わる変数または式も含む
(2) (1)の定数を以下の型のいずれかにキャストしている。
　(a) char
　　ただし、R32Cシリーズ用Cコンパイラでコンパイル
　　オプション-fsigned_char (-fSC) を使用する場合を除く。
　(b) unsigned char
　(c) unsigned short
　(d) unsigned int
　(e) unsigned long
　(f) unsigned long long

発生例:

```
#include <stdio.h>
int func(int x) { return x; }
void main(void)
{
   int i = (unsigned int)-1.0;
   int j = func((unsigned int)-1.0);  /* 発生条件(1)および(2) */
   if (i != j) {
      printf("NG (i, j) = (%d, %d)¥n", i, j); /* -1, 0になる */
   } else {
      printf("OK¥n", i, j);
   }
}
```

1.4 回避策

発生条件(1)の関数を呼び出す前に発生条件(2)のキャストした定数を
一時変数に代入し、その一時変数を関数の引数に使用してください。
例:

```
#include <stdio.h>
int func(int x) { return x; }
void main(void)
{
   int i = (unsigned int)-1.0;
   unsigned int tmp = (unsigned int)-1.0;  /* 一時変数の定義 */
   int j = func(tmp);
   if (i != j) {
      printf("NG (i, j) =1(%d, %d)¥n", i, j);
   } else {
      printf("OK¥n", i, j);
   }
```

```
}
-----------------------------------------------------------------
```

1.5 恒久対策

- R32Cシリーズ用Cコンパイラパッケージ および
  M32Cシリーズ用Cコンパイラパッケージ (M3T-NC308WA)
  改修の予定はありません。
- M16Cシリーズ, R8Cファミリ用Cコンパイラパッケージ (M3T-NC30WA)
  V.5台の次期バージョンで改修する予定です。

2. static指定子を使用した関数を割り込み関数として定義する場合の注意事項

2.1 該当製品

- R32Cシリーズ用Cコンパイラパッケージ
  V.1.01 Release 00 ～ V.1.02 Release 01A
- M32Cシリーズ用Cコンパイラパッケージ (M3T-NC308WA)
  V.5.20 Release 1 ～ V.5.42 Release 00A
- M16Cシリーズ, R8Cファミリ用Cコンパイラパッケージ (M3T-NC30WA)
  V.5.30 Release 1 ～ V.5.45 Release 01

2.2 内容

割り込みベクタ番号を記述した#pragma interruptを使用して割り込み
関数を定義した場合、その関数に対して、コードを出力せず可変割り込み
ベクタテーブルに関数アドレスを設定しない場合があります。
その結果、割り込み発生時にこの関数を呼び出せない場合があります。

2.3 発生条件

以下の条件をすべて満たす場合に発生します。
(1) 以下のコンパイルオプションを使用している。
  R32Cシリーズ用Cコンパイラパッケージの場合：
    -Ofile_inline[=<ファイル名>[,...]] (-OFI[=<ファイル名>[,...]])
  M3T-NC308WAまたはM3T-NC30WAの場合：
    -Oforward_function_to_inline (-OFFTI) と以下のいずれか。
      -O, -O1 ～ -O5, -OR_MAX (-ORM), -OR, -OS_MAX (-OSM), -OS
(2) static記憶クラス指定子を使用して関数を定義している。
(3) (2)の関数に割り込みベクタ番号を記述した#pragma interruptを
    使用している。
(4) (2)の関数の呼び出しやアドレス参照がない。

発生例:

```
-----------------------------------------------------------------
#pragma interrupt func(vect=31)   /* 発生条件(3) */
static void func(void)            /* 発生条件(2) */
{
```

}

参照のないstatic関数を削除してしまうため、関数に対するコードを生成せず、
可変割り込みベクタに関数アドレスも設定しません。このため割り込み
発生時に、当該関数を呼び出せません。

2.4 回避策

-Ofile_inline[=<ファイル名>[,...]] および
-Oforward_function_to_inline (-OFFTI) を使用しないでください。

2.5 恒久対策

- R32Cシリーズ用Cコンパイラパッケージ および
  M32Cシリーズ用Cコンパイラパッケージ (M3T-NC308WA)
  改修の予定はありません。
- M16Cシリーズ, R8Cファミリ用Cコンパイラパッケージ (M3T-NC30WA)
  V.5台の次期バージョンで改修する予定です。

3. 関数内で関数または変数をextern宣言する場合の注意事項

3.1 該当製品

- R32Cシリーズ用Cコンパイラパッケージ
  V.1.01 Release 00 ～ V.1.02 Release 01A
- M32Cシリーズ用Cコンパイラパッケージ (M3T-NC308WA)
  V.5.00 Release 1 ～ V.5.42 Release 00A
- M16Cシリーズ, R8Cファミリ用Cコンパイラパッケージ (M3T-NC30WA)
  V.1.00 Release 1 ～ V.5.45 Release 01

3.2 内容

ファイルスコープで宣言した関数Aまたは変数Aと、同名の関数Bまたは変数Bを
他の関数C内でextern宣言すると、型が異なっていてもエラーにならず、誤った
コードを生成します。
その結果、以下のような問題が発生します。

- 関数の場合
  関数C内で宣言された関数Bの呼び出し時に正しい引数が渡せない
  ことがあります。
  関数C内で宣言された関数Bの返却値を正しく受け取れないことがあり
  ます。
- 変数の場合
  関数C内で宣言された変数Bや隣接する変数が誤った値になることがあり
  ます。

3.3 発生条件

以下の条件をすべて満たす場合に発生します。

(1) ファイルスコープで宣言された関数Aまたは変数Aがある。
(2) (1)と同名で型が異なる関数Bまたは変数Bが、他の関数C内でextern宣言
　されている。
(3) (2)の関数C内で、(2)のextern宣言が有効な位置で、extern宣言された
　変数Bをアクセスしているか、またはextern宣言された関数Bを呼び出して
　いる。

関数の場合の発生例:

```
------------------------------------------------------------
#include <stdio.h>
static long double func(long double, ...);  /* 発生条件(1) */
void main(void)
{
   extern long double func(float, ...);    /* 発生条件(2) */
   long double rtn = func(1.0f);           /* 発生条件(3) */
   if (rtn != 1.0f) {
      printf("NG¥n");
   } else {
      printf("OK¥n");
   }
}
static long double func(long double x, ...) /* 発生条件(1) */
{
   return x;
}
------------------------------------------------------------
```

上記の発生例の場合、エラーにならず、引数xをlong double型で渡すべき
ところ、float型で渡します。

変数の場合の発生例:

```
------------------------------------------------------
#include <stdio.h>
static short x = 0;        /* 発生条件(1) */
static short y = 0;
void main(void)
{
   extern long x;          /* 発生条件(2) */
   x = 0xffff0000UL;       /* 発生条件(3) */
   if (y != 0) {
      printf("NG¥n");
   } else {
      printf("OK¥n");
   }
}
------------------------------------------------------
```

上記の発生例の場合、エラーにならず、変数xを変更すべきところ、変数yを変更します。

3.4 回避策

ファイルスコープに宣言した関数または変数と、関数内でextern宣言している同名の関数または変数の型をあわせてください。

関数の場合の例:

```
-----------------------------------------------------------------
#include <stdio.h>
static long double func(long double, ...);
void main(void)
{
   extern long double func(long double, ...); /* 引数型を整合 */
 . . . . . .
}
 . . . . . .
-----------------------------------------------------------------
```

変数の場合の例:

```
-------------------------------------------------------
#include <stdio.h>
static short x = 0;
static short y = 0;
void main(void)
{
   extern short x;     /* 変数型を整合 */
 . . . . . .
}
 . . . . . .
-------------------------------------------------------
```

3.5 恒久対策

- R32Cシリーズ用Cコンパイラパッケージ および
  M32Cシリーズ用Cコンパイラパッケージ (M3T-NC308WA)
  改修の予定はありません。
- M16Cシリーズ, R8Cファミリ用Cコンパイラパッケージ (M3T-NC30WA)
  V.5台の次期バージョンで改修する予定です。
  (V.6では改修しています)

4. 共用体メンバに連続して定数を書き込む場合の注意事項

4.1 該当製品

- R32Cシリーズ用Cコンパイラパッケージ

V.1.01 Release 00 ～ V.1.02 Release 01A
- M32Cシリーズ用Cコンパイラパッケージ (M3T-NC308WA)
V.3.10 Release 1 ～ V.5.42 Release 00A
- M16Cシリーズ, R8Cファミリ用Cコンパイラパッケージ (M3T-NC30WA)
V.5.00 Release 1 ～ V.5.45 Release 01

4.2 内容

共用体のメンバに連続して定数を書き込む場合に、順序を誤って書き込むことがあります。

4.3 発生条件

以下の条件をすべて満たす場合に発生します。

(1) 最適化オプション-O, -O1 ～ -O5, -OS, -OR, -OS_MAX (-OSM), -OR_MAX(-ORM)のいずれかを選択している。

(2) 定数を代入する代入文が3つ以上連続している。
注： 定数は、最適化により定数に置き換わる変数または式も含む。

(3) (2)の連続する代入文の中に、同じ共用体のメンバへ代入している2つの代入文があり、それぞれの代入先は隣接している。
なお、先行する代入文をA、後続する代入文をBとする。

(4) AとBの間に、同じ共用体のメンバへの代入文Cがある。

(5) AとCの代入先は隣接していない。

(6) BとCの代入先は重なっている。

(7) A, BおよびCの代入先は、いずれもvolatile修飾されていない。

発生例:

```
------------------------------------------------------------------------
 union{
    unsigned short unim01;
    unsigned short unim02;
    unsigned long  unim05;  /* 本例では、uni[0].unim05 と
                               uni[1].unim01 が隣接 */
 } uni[2];                  /* 発生条件(7) */
 int x;
 void func()
 {
    uni[1].unim01 = 0x1111;     /* 発生条件(2)、(3) 代入文A、(5) */
    uni[0].unim02 = 0x2222;     /* 発生条件(2)、(4) 代入文C */
    x             = 0x1234;     /* 発生条件(2) */
    uni[0].unim05 = 0x55555555;  /* 発生条件(2)、(3) 代入文B、(6) */
 }
------------------------------------------------------------------------
```

代入文Bが代入文Cより前に実行されます。この結果、代入文Bの結果が代入文Cによって変更されます。

4.4 回避策

代入文Bの直前に、ダミーのasm関数を挿入してください。
例:

```
------------------------------------------------------
void func()
{
   uni[1].unim01 = 0x1111;
   uni[0].unim02 = 0x2222;
   x          = 0x1234;
   asm();                  /* ダミーの asm() を挿入 */
   uni[0].unim05 = 0x55555555;
}
------------------------------------------------------
```

4.5 恒久対策

- R32Cシリーズ用Cコンパイラパッケージ および
  M32Cシリーズ用Cコンパイラパッケージ (M3T-NC308WA)
  改修の予定はありません。
- M16Cシリーズ, R8Cファミリ用Cコンパイラパッケージ (M3T-NC30WA)
  V.5台の次期バージョンで改修する予定です。
  (V.6では改修しています)

5. sizeof演算子に配列名を使用する場合の注意事項

5.1 該当製品

- R32Cシリーズ用Cコンパイラパッケージ V.1.01 Release 00
- M32Cシリーズ用Cコンパイラパッケージ (M3T-NC308WA)
  V.1.00 Release 1 ～ V.5.41 Release 01A
- M16Cシリーズ, R8Cファミリ用Cコンパイラパッケージ (M3T-NC30WA)
  V.1.00 Release 1 ～ V.5.45 Release 01

5.2 内容

整数と配列名の加算式にsizeof演算子を使用すると、結果がポインタサイズ
ではなく配列サイズになります。

5.3 発生条件

以下の条件をすべて満たす場合に発生します。
(1) sizeof演算子により、整数定数と配列名の加算式の型のサイズを取得して
いる。
(2) (1)の加算式は、加算演算子の左オペランドが整数定数で、右オペランドは
配列の名前である。
ただし、左オペランドは、定数畳み込みにより整数定数になる定数式をも
含む。

発生例:

```
-------------------------------------------------------
short far arr[30], i;
void func(void)
{
   i = sizeof(0+arr);    /* 発生条件(1)および(2) */
}
-------------------------------------------------------
```

上記の発生例では、&arr[0]のアドレス式のサイズ(4)ではなく、誤って配列のサイズ(60)になります。

5.4 回避策

加算演算子のオペランドの左右を入れ替えて、左に配列名を、右に整数定数を記述してください。

例:

```
-------------------------------------------------------
short arr[30],i;
void test(void)
{
   i = sizeof(arr+0); /* 左右を入れ替える */
}
-------------------------------------------------------
```

5.5 恒久対策

- R32Cシリーズ用Cコンパイラパッケージ V.1.02 Release 00で改修済みです。
- M32Cシリーズ用Cコンパイラパッケージ (M3T-NC308WA) V.5.42 Release 00で改修済みです。
- M16Cシリーズ, R8Cファミリ用Cコンパイラパッケージ (M3T-NC30WA) V.5台の次期バージョンで改修する予定です。
  (V.6では改修しています)

6. 浮動小数点定数のゼロとNaNとを比較する場合の注意事項

6.1 該当製品

- R32Cシリーズ用Cコンパイラパッケージ V.1.01 Release 00
- M32Cシリーズ用Cコンパイラパッケージ (M3T-NC308WA) V.1.00 Release 1 ～ V.5.41 Release 01A
- M16Cシリーズ, R8Cファミリ用Cコンパイラパッケージ (M3T-NC30WA) V.2.00 Release 1 ～ V.5.45 Release 01

6.2 内容

0.0とNaNを等価または不等価演算すると、等値と判定してしまう場合があります。

NaN (Not a Number) は、浮動小数点形式で表現される特殊値で、無効演算

(ゼロ同士の除算、ゼロと無限大の乗算、正の無限大と負の無限大の加算など)
の結果返される値です。

6.3 発生条件

以下の条件をすべて満たす場合に発生します。
(1) コンパイルオプション -O、-O1 ～ -O5、-OR、-OS、-OR_MAX (-ORM)および
-OS_MAX (-OSM)のいずれかを使用している。
(2) 等価または不等価演算で比較している。
(3) (2)の演算のオペランドの一方は、NaNの値を持つ浮動小数点定数式である。
注： 浮動小数点定数式は、最適化により定数に置き換わる変数または式も
含む。
(4) (2)の演算のオペランドのもう一方は、0.0 の値を持つ定数である。
注： 定数は、最適化により定数に置き換わる変数または式も含む。

発生例:

```
-------------------------------------------------------------------------
int test(void)
{
   float f  = 1.797e+308L + 0.001e+308L;
   float f1 = 1.797e+308L + 0.001e+308L;
                    /* オーバフローにより変数fとf1は無限大になる */
   f /= f1;            /* 無限大同士の除算によりNaNになる */
   if (f == 0.0f) {    /* 発生条件(2)(3)および(4) */
      return 0;
   }
   return 1;
}
-------------------------------------------------------------------------
```

上記の例では、該当する演算が最適化により誤って常に真と判定されます。

6.4 回避策

発生条件(4)に該当する定数を、ゼロの値を持つvolatile変数に変更して
ください。

例:

```
-----------------------------------------------------------
volatile float zero = 0.0f;
int test(void)
{
   float f  = 1.797e+308L + 0.001e+308L;
   float f1 = 1.797e+308L + 0.001e+308L;
   f /= f1;
   if (f == zero) {   /* ゼロの値を持つvolatile変数に変更 */
      return 0;
   }
```

```
    return 1;
}
-----------------------------------------------------------
```

6.5 恒久対策

- R32Cシリーズ用Cコンパイラパッケージ V.1.02 Release 00で改修済みです。
- M32Cシリーズ用Cコンパイラパッケージ (M3T-NC308WA) V.5.42 Release 00で改修済みです。
- M16Cシリーズ, R8Cファミリ用Cコンパイラパッケージ (M3T-NC30WA) V.5台の次期バージョンで改修する予定です。
  (V.6では改修しています)

7. K&R形式 (旧形式) 関数の引数にfloat型を使用する場合の注意事項

7.1 該当製品

- R32Cシリーズ用Cコンパイラパッケージ
  V.1.01 Release 00 ～ V.1.02 Release 01A
- M32Cシリーズ用Cコンパイラパッケージ (M3T-NC308WA)
  V.1.00 Release 1 ～ V.5.42 Release 00A
- M16Cシリーズ, R8Cファミリ用Cコンパイラパッケージ (M3T-NC30WA)
  V.1.00 Release 1 ～ V.5.45 Release 01

7.2 内容

double型の引数を持つ関数をプロトタイプで宣言し、この関数をK&R形式 (旧形式) で定義、かつ引数をfloat型で定義すると、関数内で引数の値を正しく参照できません。

7.3 発生条件

以下の条件をすべて満たす場合に発生します。

(1) 以下のコンパイルオプションのいずれも使用していない。
  -fdouble_32 (-fD32)
  -OR_MAX (-ORM)
  -OS_MAX (-OSM)
(2) double型引数を持つ関数をプロトタイプで宣言している。
(3) (2)の関数を K&R形式 (旧形式) で定義している。
(4) (3)の関数定義において、(2)の引数をfloat型で定義している。
(5) (4)の引数の最初のアクセスが読み込みである。
(6) R32Cシリーズ用Cコンパイラパッケージの場合、(4)の引数が3番目以降の引数である。

発生例:

```
-----------------------------------------------------------
 #include <stdio.h>
 float func(int, int, double); /* 発生条件(2) */
```

```
float func(x, y, f)          /* 発生条件(3)(4)(6) */
int x;
int y;
float f;                     /* 発生条件(4) */
{
   long z = 0;
   if (x) {
      z = 1;
   }
   return f;                 /* 発生条件(5) */
}
void main(void)
{
   float r = func(0, 0, 1.0);
   if (r == 1.0) {
      printf("OK¥n");
   } else {
      printf("NG¥n");
   }
}
```

発生条件(3)に該当する関数の入り口において、プロトタイプでdoubleと宣言された引数を、旧形式関数定義の引数宣言のfloat型に変換します。
しかし、変換後のデータを読み取りません。

7.4 回避策

発生条件(2)の関数を、K&R形式 (旧形式) で定義せず、プロトタイプ形式で定義してください。

例:

```
float func(double f)  /* プロトタイプ形式に変更する */
{
. . . . . .
   return f;
}
```

7.5 恒久対策

- R32Cシリーズ用Cコンパイラパッケージ および
  M32Cシリーズ用Cコンパイラパッケージ (M3T-NC308WA)
  改修の予定はありません。
- M16Cシリーズ, R8Cファミリ用Cコンパイラパッケージ (M3T-NC30WA)
  V.5台の次期バージョンで改修する予定です。
  (V.6では改修しています)

8. K&R形式 (旧形式) 関数の引数に_Bool型を使用する場合の注意事項

8.1 該当製品

- R32Cシリーズ用Cコンパイラパッケージ
  V.1.01 Release 00 ～ V.1.02 Release 01A
- M32Cシリーズ用Cコンパイラパッケージ (M3T-NC308WA)
  V.5.00 Release 1 ～ V.5.42 Release 00A
- M16Cシリーズ, R8Cファミリ用Cコンパイラパッケージ (M3T-NC30WA)
  V.5.00 Release 1 ～ V.5.45 Release 01

8.2 内容

K&R形式 (旧形式) の関数定義において_Bool型の引数に2以上の偶数を渡して
呼び出すと、その引数を1として受け取れません。

8.3 発生条件

以下の条件をすべて満たす場合に発生します。
(1) 関数を旧形式 (K&R形式) で定義している。
(2) (1)の関数の引数が_Bool型である。
(3) (2)の引数の最初のアクセスが読み取りである。
(4) (2)の引数に対する実引数の値が2以上の偶数である。

発生例:

```
#include <stdio.h>
_Bool func(x)     /* 発生条件(1) */
_Bool x;          /* 発生条件(2) */
{
   return x;      /* 発生条件(3) */
}
void main(void)
{
   _Bool x = func(2); /* 発生条件(4) */
   if (x == 1) {
      printf("OK¥n");
   } else {
      printf("NG¥n");
   }
}
```

発生条件(1)に該当する関数の入り口で引数を_Bool型に変換しません。
そのため、2以上の値がそのまま参照で使用されます。

8.4 回避策

該当する関数をK&R形式 (旧形式) で定義せず、プロトタイプ形式で定義してください。

例:

```
-------------------------------------------------------
   _Bool func(_Bool x) /* プロトタイプ形式に変更する */
   {
      return x;
   }
-------------------------------------------------------
```

8.5 恒久対策

- R32Cシリーズ用Cコンパイラパッケージ および
  M32Cシリーズ用Cコンパイラパッケージ (M3T-NC308WA)
  改修の予定はありません。
- M16Cシリーズ, R8Cファミリ用Cコンパイラパッケージ (M3T-NC30WA)
  V.5台の次期バージョンで改修する予定です。
  (V.6では改修しています)

9. switch文の制御式の型にsigned charを使用する場合の注意事項

9.1 該当製品

- M32Cシリーズ用Cコンパイラパッケージ(M3T-NC308WA)
  V.1.00 Release 1 ～ V.3.10 Release 3
- M16Cシリーズ, R8Cファミリ用Cコンパイラパッケージ(M3T-NC30WA)
  V.2.00 Release 1 ～ V.4.00 Release 2 および
  V.5.10 Release 1 ～ V.5.45 Release 01

9.2 内容

switch文の制御式の型がsigned charの場合、正しいcaseラベルに分岐しない場合があります。

9.3 発生条件

以下の条件をすべて満たす場合に発生します。

(1) switch文の制御式の型がsigned charである。

(2) (1)のswitch文が、以下のいずれかを満たす。

(a) case値の最小値が "-127" かつ最大値が "127" で、そのcase値が -127から127まで隙間なく連続している

(b) case値の最小値が "-128" かつ最大値が "126" で、そのcase値が -128から126まで隙間なく連続している

(c) case値の最小値が "-128" かつ最大値が "127" で、そのcase文の総数が135個以上存在する。
注： case文にはdefaultを含まない

(3) (1)の制御式の値がゼロより小さい。

ただし、(2)(a)に該当し、かつ(1)の制御式が-128の場合を除く。

発生例:

```
------------------------------------------------------------------
#include <stdio.h>
signed char d = -1;                     /* 発生条件(3) */
void main( void )
{
   switch( d ) {                        /* 発生条件(1) */
   case -127 : printf( "NG[-127]¥n"); break; /* 発生条件(2)(a) */
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
   case -1 : printf( "OK¥n"); break;
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
   case 127 : printf( "NG[127]¥n"); break;   /* 発生条件(2)(a) */
   }
}
------------------------------------------------------------------
```

caseラベルの値を分岐テーブル要素番号に変換する際に、ゼロ拡張するため、defaultラベルに分岐します。

9.4 回避策

発生条件(1)に該当する制御式をintにキャストしてください。

例:

```
------------------------------------------------------------
#include <stdio.h>
signed char d = -1;
void main( void )
{
   switch( (int)d ) {       /* int型へのキャストを追記 */
   case -127 : printf( "NG[-127]¥n"); break;
. . . . . . . . . . . . . . . . . . . . . . . . . . . . .
   case -1 : printf( "OK¥n", d); break;
. . . . . . . . . . . . . . . . . . . . . . . . . . . . .
   case 127 : printf( "NG[127]¥n"); break;
   }
}
------------------------------------------------------------
```

9.5 恒久対策

- M32Cシリーズ用Cコンパイラパッケージ (M3T-NC308WA)
  V.5.00 Release 01で改修済みです。
- M16Cシリーズ, R8Cファミリ用Cコンパイラパッケージ (M3T-NC30WA)
  V.5台の次期バージョンで改修する予定です。
  (V.6では改修しています)

[免責事項]
過去のニュース内容は発行当時の情報をもとにしており、現時点では変更された情報や無効な情報が含まれている場合があります。
ニュース本文中のURLを予告なしに変更または中止することがありますので、あらかじめご承知ください。