プリプロセッサ，構造体/共用体，データ構造，プロセスとスレッド

GCCでコンパイルする前に行う処理

岸 哲夫

C言語で頭に＃が付いた命令文をよく目にするだろう．これは実際は命令文ではなく，疑似命令文である．本章では，その使い方をわかりやすく説明する．（筆者）

プリプロセッサ ――前処理を行うプログラム

● プリプロセッサとは

「プリプロセッサ」とはコンパイル前にソース・プログラムに対して行われる前処理のことです．

コンパイルのプロセスは，

① 前処理
② コンパイル
③ アセンブル
④ リンク

となります．そして，その前に行う処理を「プリプロセッサ」と呼びます．

オプションの説明に入る前に，プリプロセッサの簡単な使い方について説明します．

なお，gcc -Eでプリプロセスのみ行うことができます．いっしょに-dMを付ければ定義されたマクロ一覧が得られます．

そしてインクルード・パスを通すのは-Iで，たとえば，

```
-DDEBUG=1
```

と書くと，

```
#define DEBUG 1
```

と同様の効果が得られます．

リスト1のソースのようにデバッグ時に使用することが可能

リスト1　-Dオプション使用例のソース・リスト（test67.c）

```
/*
 *
 *#define文その1
 */
#include <stdio.h>
int main(void)
{
    int           ix;
    char      moji[]    =    "0123456789";
    char      *pt       =    moji;    //ポインタptはここで初期化
    ix        =    0;                 //カウンタixも初期化
    if    (DEBUG)    printf("現在はデバッグ中!!¥n");
    do                                //whileループ
    {
        if    (!*pt)    break;        //"moji"の終わりにきたら抜ける
        printf("内容は%c¥n",*pt);
        if    (DEBUG)    printf("whileループ%d回目¥n",ix++);
    }
    while     (*pt++);
    return 0;
}
```

```
岸@main ~
$ gcc test67.c -o test67 -DDEBUG=0

岸@main ~
$ ./test67
内容は0
内容は1
内容は2
内容は3
内容は4
内容は5
内容は6
内容は7
内容は8
内容は9

岸@main ~
$ gcc test67.c -o test67 -DDEBUG=1

岸@main ~
$ ./test67
現在はデバッグ中!!
内容は0
whileループ0回目
内容は1
whileループ1回目
内容は2
whileループ2回目
内容は3
whileループ3回目
内容は4
whileループ4回目
内容は5
whileループ5回目
内容は6
whileループ6回目
内容は7
whileループ7回目
内容は8
whileループ8回目
内容は9
whileループ9回目
```

図1　test67.cをコンパイルして実行した結果

KEYWORD ――プリプロセッサ，#define文，#undef文，#include文，#error文，#if文，構造体，共用体，データ構造，単方向リスト，双方向リスト，環状リスト，配列，スタック，キュー，木，スケジューリング，プロセス，スレッド

リスト2　マクロ使用例のソース・リスト(test68.c)

```
/*
 *単純なマクロの例
 */
#include <stdio.h>
#define square_int(x) ((x) * (x))
int main(void)
{
    int       y    =    20;
    printf("%d^2は?%d¥n",y,square_int(y));
    return  0;
}
```

```
$ ./test68
20^2は?400
```

図2　test68.cをコンパイルして実行した結果

です(**図1**).

また，プリプロセッサはヘッダ・ファイルを管理するために，よく使われます．ヘッダ・ファイルを二重に読み込んで前処理を行わないために，次のように使います．

例)

```
#ifndef _STDIO_H
    #define _STDIO_H

 //ヘッダ・ファイルstdio.hの中身.....

#endif
```

実際に/usr/includeの下にあるstdio.hを見るとわかりますが，#ifndef文と#endif文で囲います．これにより，stdio.hが2回#includeされてもすでにその名前が#defineされているので，#ifndef文と#endif文で囲った部分が無視されます．

● マクロ――プリプロセス時にソースに展開

マクロはプリプロセス時にソースに展開されます．関数呼び出しを行いたくないような簡単な処理は，マクロにすると速くなる場合もあります．

リスト2にマクロを使用した例のソース・リストを，**図2**にその実行結果を示します．

● 定義済みマクロもある

C99規格またはそれ以前から定義されているマクロには，次のようなものがあります．

① __LINE__：行番号
② __FILE__：ソース・ファイル名
③ __DATE__：ソース・ファイルの変更日．mmm dd yyyyの形式で出力
④ __TIME__：ソース・ファイルの変更時間．hh:mm:ssの形式で出力
⑤ __STDC__：C99準拠のコンパイラならば1を出力

非常に便利なマクロですが，デバッグに役立つもっと便利なものもあります．

リスト3　デバッグに役立つ定義済みマクロを使ったソース・リスト(test69.c)

```
/*
 *デバッグに役立つ定義済みマクロ
 */
#define dbg(...) (printf("%s %u @%s:",¥
          __FILE__,__LINE__,__func__),printf(" "__VA_ARGS__))
void foo1(int i);
void foo2(int i);
void foo3(int i);
void foo4(int i);
main()
{
 int x = 1;
 foo1(x++);
 foo2(x++);
 foo3(x++);
 foo4(x);
}
void foo1(int i)
{
  dbg("i=%d¥n", i);
}
void foo2(int i)
{
  dbg("i=%d¥n", i);
}
void foo3(int i)
{
  dbg("i=%d¥n", i);
}
void foo4(int i)
{
  dbg("i=%d¥n", i);
}
```

```
$ ./test69
test69.c 19 @foo1: i=1
test69.c 23 @foo2: i=2
test69.c 27 @foo3: i=3
test69.c 31 @foo4: i=4
```

図3　test69.cをコンパイルして実行した結果

リスト3，**図3**に示すように，変数の値がどの関数を通ってどう変わったかが一目でわかるようになっています．もちろんgdbを使ったほうが速くなります．

マクロに使うコマンドを次に示します．

▶#define文

前に説明したようにマクロに使ったりシンボルを定義するために使用します．

▶#undef文

#undefを使用すると，シンボルを未定義状態にできます．前の行でdefineで定義されたものを取り消すことができます．

▶#include文

おもにヘッダをインクルードするために使いますが，共用できるソース・ファイルをインクルードするためにも使います．

▶#error文

対象の行をエラーにすることができます．

シンボルDEBUGを付けてコンパイルしないといけない**リスト1**(test67.c)のソースにerror文を付加しておけば，-Dオプションを忘れた場合でも，警告を出させることができます