C言語の学習　プリプロセッサー

山本昌志*

2006年6月6日

概 要

プリプロセッサーのうち，本講義で重要なファイルの挿入`#include`と文字列置換`#define`について述べている．

1　本日の学習内容

本日の内容は教科書 [1] の 15 章プリプロセッサである．主にデータ構造に関わる 12〜14 章は省く．数値計算のデータ構造は単純なので，ほとんど場合，配列で間に合う．しかし，他の多くのアプリケーションでは，アルゴリズムと並んでデータ構造は重要である．いろいろなアプリケーションを作成したい者は，12 章〜14 章を自習せよ．複雑なデータ構造を使う必要がる場合は，12 章の構造体は必須である．また，PIC や H8 などのマイコンのプログラムを作成する場合，ビットフィールド (13 章) と共用体 (14 章) を理解しなくてはならない．

教科書の 15 章プリプロセッサは，たくさんのことを説明している．しかし，本講義では以下のことが分かればよい．

- プリプロセッサーは，コンパイル前に言語のソースプログラムの処理を行う．
- `#include`は，ファイルのテキストを挿入する．
- `#define`は，文字列の置き換えを行う．

2　プリプロセッサの実行タイミング (p.282)

「gcc」を使って，ソースプログラムをマシン語に直すとき，通常はコンパイルすると言うが，実際にはコンパイル以外の作業も行っている．教科書の p.282 に示されているように，コンパイルの前にプリプロセッサーがソースファイルを書き直している．そして，コンパイルにより，ソースファイルをオブジェクトファイルに変換している．最後に，リンカーがいろいろなファイルを寄せ合わせて実行可能なファイルを作成する．

プリプロセッサーは，コンパイルに先だって，ソースファイルを変更している[1]．gcc ではいろいろな処理が行われるが，プリプロセッサーの動作が一番最初に実施される．そこで，諸君が書いた C 言語のプログ

*独立行政法人　秋田工業高等専門学校　電気工学科

[1]教科書に書いてあるとおり，プリプロセッサーには他にも役割がある．それについては，割愛する．

ラムを変更しているのである．preprocessor(pre:あらかじめ，processor:処理するもの) という名前からも，そのことがよくわかる．

プリプロセッサーができることは，教科書の p.283 の表に示されている．いろいろな機能があるが，本講義で使うのは

- ファイルの挿入を行う`#include`
- マクロ定義を行う`#define`

くらいである．

3　プリプロセッサの使い方 (p.282)

3.1　プリプロセッサの書き方

プリプロセッサーの書式は，以下の通りである．今まで，さんざん書いてきているので，大体理解はできるであろう．

```
#コマンド　パラメーターリスト
```

このプリプロセッサーの書き方には，以下の約束がある．

- #の前後に空白が有ってもよい．#とコマンドの間の空白も許されるが，プログラムがわかりにくくなる．通常は#とコマンドの間に空白を入れない．
- プリプロセッサーコマンドは，その行で終わりである．C 言語の文のように';' があらわれるまで有効ということではなく，その行で完結する．このようなことから，文の区切りを表す';' を書いてはならない．
- ただし，プリプロセッサーコマンドを複数行にわたって，記述したい場合，行の終わりに'\'をつけて，次行と接続することができる．

3.2　ファイルの挿入 (p.284)

3.2.1　`#include` の役割と例

`#include "ファイル名"`は，指定したテキストファイルとこの行を置き換える．ファイル名が`<filename>`と鉤括弧<>で書かれた場合には，システムに定義されたファイルを示す．unix では，`/usr/include` にあるファイルで置き換えられる．プログラマーが作成したファイルを挿入したい場合は，　`#include "myfile.h"` のように記述する．

プログラマーが自分でヘッダーファイルを書くことがある．例えば，大規模なプログラムを作成する場合，ファイルは 1 つではなく，いろいろな部分を別々のファイルに書いて，それをあとであわせて，1 つのプログラムにする（分割コンパイル）．その場合には，共有する構造体や大域変数を 1 つのファイルにして，`myfile.h` というようなファイルを作り，それぞれのソースプログラムでインクルードする．このようにすると，同じ文をそれぞれのファイルに記述する必要が無くなり，タイピングが楽になるとともに，ミスが減

る．ここでの学習では，分割コンパイルをするほどのプログラムを書くことはない．将来，比較的大きなプログラムを書くときに，勉強すればよいだろう．

#include の動作の理解のために，へんなプログラム例を示す．Hello World !!を出力するおなじみのプログラムである．

リスト 1: ソースプログラム

```
1 #include <stdio.h>
2 #include "hoge"
3  return 0;
4 }
```

これをコンパイルして，実行ファイルを作るためには，以下のファイル (hoge) が必要である．当然，このファイルはソースプログラムのインクルード 文で取り込まれる．

リスト 2: インクルードするファイル (hoge)

```
1 int main(void){
2  printf("Hello World !!\n");
```

実行の結果，#include により，そこの行が指定のファイルに置き換わっているのが理解できるはずである．使い方によっては，便利そうであることが理解できればよい．この行の置き換えはコンパイルの前に行われることに注意が必要である．

3.2.2 ヘッダーファイル

諸君は，プログラムの最初に必ず#include <stdio.h>とパブロフの犬の如く書いていただろう．ここにきて，#include の意味が分かった．stdio.h というファイルをインクルードしているだけだ．そうなると，stdio.h を見たくなるのが人情というものである．幸い，これはテキストファイルなので見ることができる．以下のコマンドをターミナルへ打ち込んで，stdio.h を見よう．

```
less /usr/include/stdio.h
```

キーボーの f で前のページを，b で後ろのページを見ることができる．終わりたい場合は，q を押す．

その中をみると，関数のプロトタイプ宣言等がごちゃごちゃ書かれているのが分かるだろう．これが，諸君のプログラムの最初の方に追加されるのである．これで，printf() などの関数が使えるようになるのである．

3.3 マクロ定義 (p.287)

3.3.1 文字列置換

#define はファイル内の文字列を指定の通りに書き換える．単純な文字列の置き換えと，引数を含む文字列の置き換えがある．引数をとるマクロ定義は，本講義では使う必要がないので，ここでの学習範囲外とする．

単純な文字列置換をオブジェクト形式マクロと言う．これは，次のように書く

```
#define hogehoge fugafuga
```

このプリプロセッサーコマンドがあると，この行以降の文字列 hogehoge が文字列 fugafuga に，すべて置き換わる．

置き換え前の文字列—ここでは hogehoge—はすべて大文字で記述する習慣となっている．別に小文字でも問題なく動作するが，ほとんどのプログラマーは大文字を使う．その方が，プログラムがわかり易いからである．

3.3.2 プログラム例

リスト 3 に，文字列置換を使ったプログラムラム例を示す．4 行目の文字列置換

```
#define NSTEPS 361
```

により，全ての NSTEPS という文字列が，361 に変えられる．このようにすることにより，同じ値の部分を一度に変えることができる．プログラムミスが減るので，便利である．

リスト 3 のプログラムは，三角関数の数表を出力する．0〜2π ラジアンを 360 等分して，関数の値を計算，そして出力している．例えば，500 等分の値を出力したければ，#define NSTEPS 361 とする．文字列置換を使えば，プログラムの変更が容易であることがわかる．

このプログラムの 31 行目に，M_PI というわけの分からない変数らしきものがある．実は，これも文字列置換の対象である．ヘッダーファイル math.h の中に，

```
# define M_PI 3.14159265358979323846 /* pi */
```

と書かれている．ようするに，円周率 π の値に変換される．ウソだと思うならば，

```
less /usr/include/math.h
```

とコンソールに打ち込んで調べよ．

このプログラムは，数学関数—こでは三角関数—を使っている．そのため，ヘッダーファイル math.h をインクルードしている．さらに，これをコンパイルするためには，

```
gcc -lm -o keisan sankaku.c
```

と lm とオプションを追加する必要がある．

リスト 3: 三角関数表を出力するプログラム

```
1  #include <stdio.h>
2  #include <math.h>
3
4  #define NSTEPS 361
5
6  void cal_function(double x[], double s[], double c[], double t[]);
7
8  /* ================================================================ */
9  /*                メイン関数                                        */
10 /* ================================================================ */
11 int main(void){
12   double x[NSTEPS], s[NSTEPS], c[NSTEPS], t[NSTEPS];
13   int i;
14
```

```
15    cal_function(x, s, c, t);
16
17    for(i=0; i<NSTEPS; i++){
18      printf("%f\t%f\t%f\t%f\n",x[i], s[i], c[i], t[i]);
19    }
20
21    return 0;
22  }
23
24  /* ================================================================== */
25  /*                関数計算                                            */
26  /* ================================================================== */
27  void cal_function(double x[], double s[], double c[], double t[]){
28    int i;
29
30    for(i=0; i<NSTEPS; i++){
31      x[i]=(double)i/NSTEPS*2.0*M_PI;
32      s[i]=sin(x[i]);
33      c[i]=cos(x[i]);
34      t[i]=tan(x[i]);
35    }
36
37  }
```

参考文献

[1] 林春比古. 新訂 C 言語入門 シニア編. ソフトバンク パブリッシング, 2004.