プリプロセッサー

山本昌志*

2007 年 5 月 15 日

概 要

ソースファイルから実行ファイルができあがるまでのプロセス，およびプリプロセッサーの使い方の基礎を学習する．

1 前回の復習と本日の学習内容

1.1 復習

前回は，分割コンパイルの方法と Makefile の書き方について，学習した．大規模なプログラム開発では，ひとつのファイルにプログラムを書くことができなくなる．そのとき，複数のファイルに分けてプログラムを記述する．この複数のプログラムから実行ファイルを作成する操作を分割コンパイルと言う．

コンパイルして実行ファイルの作成を自動化するために，Makefile がある．大量のソースファイルから実行ファイルを作成する場合，いちいち `gcc` コマンドをターミナルから打ち込んでいては，作業が大変である．Makefile に実行ファイルの作成手順を書いておけば，コマンド「`make`」一発で実行ファイルができあがる．大変便利な機能である．

1.2 本日の学習内容

本日は，プリプロセッサについて学習する．ここでの学習のゴールは，以下のとおりである．

- C 言語のソースファイルから機械語の実行ファイルができるまでのプロセスが分かる．
- プリプロセッサの機能の機能のうち，`#include` と `#define` の動作が理解できる．

教科書 [1] の pp.47–67 が本日の範囲である．ただし，この範囲のうち条件付き取り込みと組込みマクロについては説明をしない．当面の間，諸君はこれらの機能を使うことは無いからである．

*独立行政法人　秋田工業高等専門学校　電気情報工学科

2　gccによる実行ファイルの作成

C 言語のソースファイルから，実行ファイルが作られるまでには複雑なステップがある．図 1 に示すように，大まかに 4 つのステップがある; プリプロセス，コンパイル，アセンブル，リンクである．

gcc の場合，適当なオプションを付けると処理を途中で終了させることができる [2]．図 1 の左側に書いてある gcc のオプションを次のように使うことにより，gcc の途中の出力を見ることができる．ここでは，ソースファイルを hoge.c としている．

- ソースファイルはプリプロセッサーにより書き換えられる．書き換えたファイルを見る場合，コマンド「`gcc -E hoge.c > hoge.i`」とする．すると書き換えられた，テキストファイル hoge.i ができる．
- C 言語のソースプログラムは，一度アセンブラ言語[1]に変換される．コマンド `gcc -S hoge.c` とすればアセンブラ言語のファイル hoge.s ができる．
- アセンブラ言語を機械語に変換する作業をアセンブルと言う．コマンド `gcc -c hoge.c` とすれば機械語のオブジェクトファイル[2]である hoge.o ができる．
- 最後にそのた必要なファイルをリンク—ファイルをくっつける—して実行ファイルができる．-o オプションが無いと a.out という機械語のファイルができる．コマンドは,「`gcc hoge.c`」である．

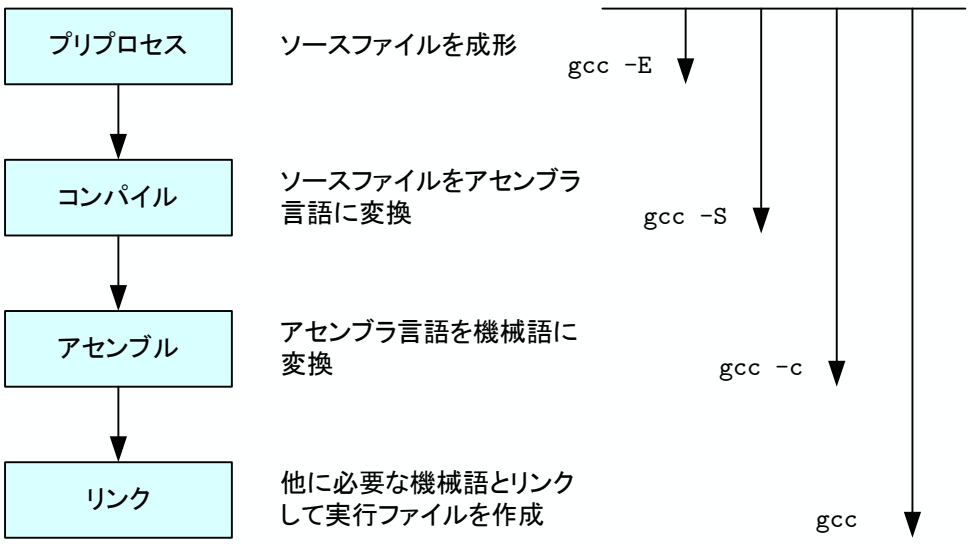

図 1: gcc が実行ファイルをつくるプロセス．左側に示すようにオプション付きで処理をすると，途中までの結果が得られる．

[1]機械語と 1 対 1 に対応した言語.
[2]正確に言うと，全て機械語ではなない．ほとんど機械語ではあるが，他も含まれる．

3　プリプロセッサー

3.1　ファイルの挿入とマクロ定義

3.1.1　#include

諸君は，#include をおまじないのように使ってきた．いよいよ，その役割を学習するときがきた．

#include はファイルの挿入を行う．例えば，#include <stdio.h>はこれが書かれた位置に/usr/include/stdio.h というファイルを挿入する．ただ，それだけのことを行うのである．また，#include "myfunc.h"と書けば，myfunc.h というファイルが挿入される．<ファイル名>と"ファイル名"は，ファイルの検索するパスが異なる．<ファイル名>の場合は，標準ヘッダーファイルがあるディレクトリー—通常は/usr/include—が検索される．一方，"ファイル名"の場合はカレントディレクトリーが検索される．

リスト 1 とリスト 2 をプリプロセッサーで処理した例を示す．このような書き方は通常のプログラムでは行うことは無い．#include の処理を分かり易く示すために極端な例を使っている．通常は，ヘッダーファイルを挿入するため，プログラムの先頭付近にファイル名.h をインクルードする．

リスト 1: #include を使ったプログラム (include.c).

```
1  int main(void)
2  {
3    #include "main.c"
4
5    return 0;
6  }
```

リスト 2: #include により，挿入されるソースプログラム (main.c).

```
1  int a, b;
2
3  b=0;
4  a=b+3;
```

リスト 1 とリスト 2 は，コマンド「gcc -E include.c > include.i」により，プリプロセスの処理が行われる．処理の結果，include.i という処理済のテキストファイルであるリスト 3 ができる．このリスト中で行頭が#ではじまる行はラインマークと呼ばれ，次の処理を行うコンパイラーは，注釈行 (コメント) として取り扱う．すなわち，アセンブラ言語に変換するときに無視する．行頭が#の部分を取り除くと，元のリスト 1 の 3 行目にリスト 2 が挿入されたことが分かるだろう．

リスト 3: プリプロセッサーにより処理されたプログラム (include.i).

```
1  # 1 "include.c"
2  # 1 "<built-in>"
3  # 1 "<コマンドライン>"
4  # 1 "include.c"
5  int main(void)
6  {
7  # 1 "main.c" 1
8  int a, b;
9
10 b=0;
11 a=b+3;
```

```
12 # 4 "include.c" 2
13
14   return 0;
15 }
```

3.1.2 #define

マクロ定義#define には**文字列の定義**と**マクロ関数の作成**というふたつの使い方がある．これらを使った例をリスト 4 に示す．これを，プリプロセッサーで処理した結果をリスト 5 に示す．

マクロ定義は,「#define マクロ名 文字列」と書く．これ以降，マクロ名で指定した文字列が指定の文字列に変換される．C 言語の慣習では，マクロ名は大文字で書く．

マクロ定義は，文字列の置き換えを行っているだけである．これに注意して使わなくてはならない．

リスト 4: #define を使ったプログラム (define.c).

```
1  #define HOGE 123
2  #define FUGA 456
3  #define SUM(i,j) i+j
4
5  int main(void)
6  {
7    int c;
8
9    c=SUM(HOGE,FUGA);
10
11   return 0;
12 }
```

リスト 5: プリプロセッサーにより処理されたプログラム (define.i).

```
1  # 1 "define.c"
2  # 1 "<built-in>"
3  # 1 "<コマンドライン>"
4  # 1 "define.c"
5
6
7
8
9  int main(void)
10 {
11   int c;
12
13   c=123 +456;
14
15   return 0;
16 }
```

3.2 組込みマクロ

デバッグのとき，便利である．興味のある者は，自分で学習せよ．

また，教科書の pp.65–66 の coffee break に書かれている assert マクロは便利である．機能を理解して使ってみると良い．

3.3 条件付き取り込み

これは，いろいなオプションつきでコンパイルするときに使う．興味のある者は，自分で学習せよ．

4 プログラム作成の練習

[**練習 1**] `Hello World!`を表示するプログラム (hello.c) を作成せよ．そして，その実行ファイルが作成されるまでのプロセスを以下の手順により，確かめよ．

1. プリプロセッサーの処理を確かめる．コマンド `gcc -E hello.c > hello.i` によりプリプロセスの処理を行う．そして，できあがったファイル (hello.i) の中身を確認せよ．
2. コンパイラーの処理を確かめる．コマンド `gcc -S hello.c` により，アセンブラー言語のファイル (hello.s) を作成する．このファイルの中身を確認せよ．
3. アセンブルの処理を確かめる．コマンド `gcc -c hello.c` により，オブジェクトファイル (hello.o) を作成する．このファイルの中身を確認せよ．ただし，手順は以下の通り．
   (a) コマンド「`emacs &`」により，emacs を立ち上げる．
   (b) エスケープキー [Esc] を押したのち [x] キーを押す．そして,「`hexl-find-file`」とミニバッファーに入れる．ファイル名を聞いてくるので,「`hello.o`」と入れる．
   (c) ファイルが見えるので確認する．機械語の 16 進数が見える．中身は分からなくてよい．
4. リンクの処理を確かめる．コマンド `gcc hello.c` により，実行ファイル (a.out) を作成する．このファイルの中身を確認せよ．確認方法は，アセンブラーの処理と同じ．実行できる機械語が見えるだろう．

[**練習 2**] `MAX` と `MULTI` を定義して，いかのように表示せよ．

```
1       5
2       10
3       15

長いので省略

100     500
```

これは，`MAX` を 100，`MULTI` を 5 とした場合である．`MAX` の値や `MULTI` の値を変化させてみよ．

[**練習 3**] 引数が度 [deg] の三角関数を`#define` により，定義せよ．そして適当なプログラムを作成して，実行してみよ．

5　課題

5.1　内容

以下の課題を実施し，レポートとして提出すること．

[問 1]　(復予) 教科書 [1] の第 1 章と第 2 章を 2 回読め．レポートには「2 回読んだ」と書け．

[問 2]　(復) 本日配布したプリントを 2 回読め．レポートには「2 回読んだ」と書け．

[問 3]　(復) 以下について，簡単に説明せよ．

- `#include` の処理
- `#define` の処理

[問 4]　(復) ここでの学習内容でわからないところがあれば，具体的に記述せよ．

提出要領はいつものとおり．ただし，期限は 5 月 22 日 (火)AM 8:45，課題名は「課題　プリプロセッサー」とする．

参考文献

[1] 内田智史監修, (株) システム計画研究所編. C 言語によるプログラミング　応用編　第 2 版. (株) オーム社, 2006.

[2] 西田亙. *GNU Development Tools*. オーバーシー・パブリッシング, 2006.