分割コンパイルと Makefile

山本昌志*

2007 年 5 月 8 日

概 要

ソースプログラムを分割コンパイルする概要を説明する．分割コンパイルによるソースプログラムの記述方法とヘッダーファイルの書き方，Makefile の書き方を簡単に説明する．

1 前回の復習と本日の学習内容

1.1 復習

前回はファイル処理のプログラム作成の演習を行った．ファイル処理を行うためには，四つのことをプログラムに書かなくてはならない;(1) ファイル型のポインターの宣言，(2) ファイルのオープン，(3) ファイルの読み込み/書き込み動作，(4) ファイルのクローズ．実際の例を図 1 に示す．諸君は，このパターンを絶対に忘れてはならない．他のプログラミング言語でもファイル操作は，C 言語と似たようなパターンのものが多い．

1.2 本日の学習内容

本日は，分割コンパイルについて学習する．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．本日は，この分割コンパイルに必要なテクニックのさわりを学習する．今年度末，諸君はある程度長いプログラムを書く．本日，身に付けるテクニックはその時に役立つであろう．

分割コンパイルを使うための最低限のテクニックを身に付けることが本日の学習のゴールである．具体的には，以下の通り．

- ソースファイルを分割する理由が分かる．そして，ソースファイルを分割することができる．
- ヘッダーファイルの書き方が分かる．
- 簡単な Makefile が書ける．

教科書 [1] の pp.2–36 が本日の範囲である．ただし，この範囲のうちコンパイル—実際は gcc ドライバー—の処理については説明をしない．各自教科書を読むこと．

*独立行政法人　秋田工業高等専門学校　電気情報工学科

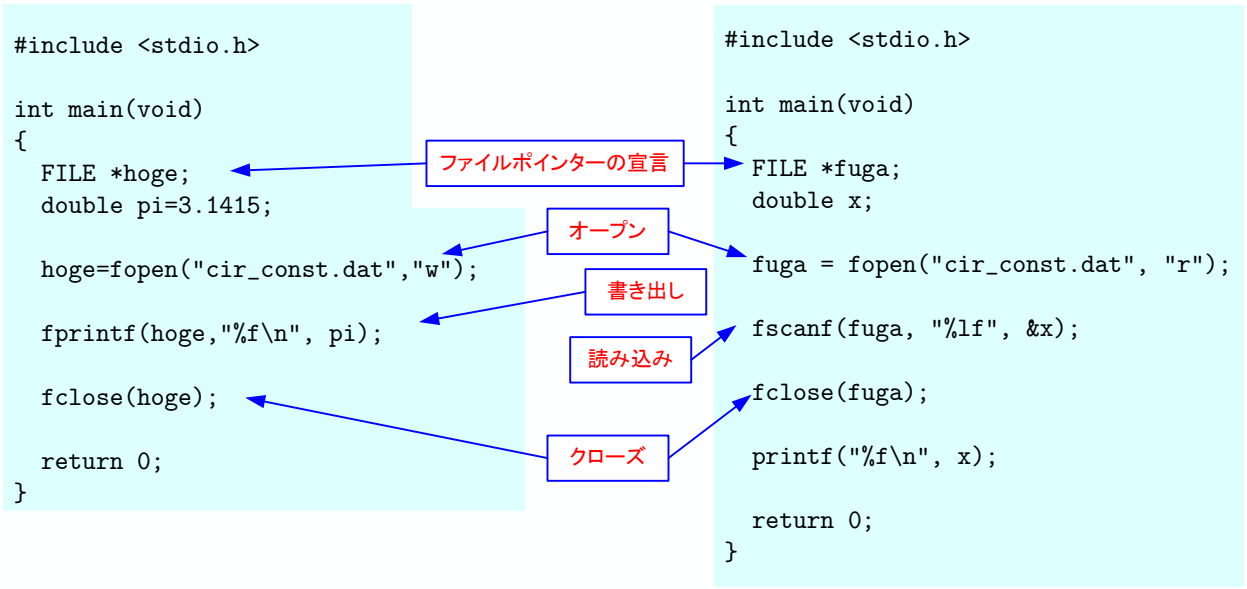

図 1: ファイルへの書き込みとファイルからの読み込みのプログラム例.

2　分割コンパイルの例

簡単な例を使って，分割コンパイルをしめす．この例で示したプログラムならば，わざわざ分割コンパイルを使うまでもない．分割コンパイルは大規模プログラム開発で使うテクニックであるが，そのような例を示すわけにもいかないので，簡単なプログラムを使う．

リスト 1〜4 が分割コンパイルの例である．このプログラムは，関数 $f(x) = x + x\sin(x)$ を指定された数で $[0, 2\pi]$ を分割し，値をファイルに書き出す．最初の 2 つのプログラムに，アルゴリズムは次のようになっていることがわかる．

1. 分割数の入力
2. データの作成しファイルに保存

メイン関数を見れば，大体のプログラムの流れが分かる．詳細は関数の集まりの `functions.c` に書いてある．こうすることによりプログラムがわかりやすくなる．

これまでとはことなり，この動作のプログラムは以下に示す 4 つのファイルからできている．

`main.c`	プログラムのメイン関数を書いている．プログラム全体を統括する役割がある．メイン関数を見れば，プログラム全体の流れが分かる．
`functions.c`	さまざまな関数の記述がある．ここでは，目的の処理を行うために，関数単位で機能を提供する．
`functions.h`	関数の集まりである functions.c のプロトタイプ宣言をまとめている．
`Makefile`	分割ファイルから，実行ファイルを作る手順が書いてある．

これらのファイルの役割や書き方については，次の節で詳しく説明する．

リスト 1: main.c のプログラム．

```
1  #include <stdio.h>
2  #include "functions.h"
3
4  int main(void)
5  {
6    int n;
7
8    n=input_data();
9    make_data(n);
10
11   return 0;
12 }
```

リスト 2: functions.c のプログラム．

```
1  #include <stdio.h>
2  #include <math.h>
3  #include "functions.h"
4
5  //=======================================
6  // データ数の取得の関数
7  //=======================================
8  int input_data(void)
9  {
10   int num;
11
12   printf("How many datas?\t");
13   scanf("%d",&num);
14
15   return num;
16 }
17
18 //=======================================
19 // ファイルの作成
20 //=======================================
21 int make_data(int nd)
22 {
23   int i;
24   double dx, x;
25   FILE *out;
26
27   out=fopen("result.dat", "w");
28
29   dx = 2*M_PI/nd;
30   for(i=0; i<=nd; i++){
31     x=i*dx;
```

```
32      fprintf(out,"%f\t%f\n", x, f(x));
33    }
34
35    fclose(out);
36
37    return 0;
38  }
39
40  //=======================================
41  // 数学関数
42  //=======================================
43  double f(double x)
44  {
45    return x+x*sin(x);
46  }
```

リスト 3: ヘッダーファイル functions.h.

```
1  int input_data(void);
2  int make_data(int nd);
3  double f(double x);
```

リスト 4: メイクファイル Makefile.

```
1  mkdat : main.o functions.o
2          gcc -o mkdat main.o functions.o -lm
3
4  main.o : main.c
5          gcc -c main.c
6
7  function.o : function.c
8          gcc -c function.c
```

3 分割コンパイルのテクニック

3.1 ソースプログラムの分割

3.1.1 分割する理由

いままでのプログラムであれば，リスト 1〜3 をひとつのファイルで書いていた．そして,「gcc」コマンドでコンパイルを行い実行ファイル (機械語) を作っていた．なぜ，ソースプログラムはリストリスト 1〜3 のように分割する必要があるのだろうか? これらのファイルに記述されていることは今までひとつのソースファイルに書いていたことである．

ソースプログラムを分割する理由として，参考文献 [2] には以下のような記述がある．

- 機能単位の開発を可能にする．
- モジュールごとのテスト/デバックを可能にする．
- 必要があれば，ソースコードを隠蔽し，モジュールのみを提供することができる．
- プログラムを修正した際に，必要な部分のみコンパイルすることができる．
- モジュールの再利用性を高める

- プログラムの見通しを良くする．

これらの理由の大部分は，長いプログラムを書くためと考えてよい．世の中で使われているある程度のプログラムはソースコードは長く，複数のプログラマーによって書かれている．このように複数のプログラマーでひとつのプログラムを作成する際に，分割コンパイルは威力を発揮する．プログラマー毎，あるいは機能別にソースプログラムを書くことができ，各々がコンパイルすることができる．分割したファイル毎に，バグ取りができるのである．最後に完成した全てのファイルを集めて，コンパイルすれば実行ファイルを作ることができる．

3.1.2 分割方法

プログラムの分割は，いままで作成したソースプログラムを適当なファイルに書き出すだけで特別なテクニックは無い．実際には，プログラマー毎，あるいは機能別に開発のしやすいように分割すればよい．最初からソースプログラムを分割して開発するのである．こうして分割したファイル毎にコンパイルを行い，エラーの無いプログラムを作成する．全てのファイルができあがったの後，合わせてコンパイルを行いひとつの実行ファイルを作成する．

このテクニックは一人でプログラムを作成する場合にも有効である．私がある程度のプログラムを開発するとき，機能別に分けてソースファイルを作る．そうするとプログラムが整理でき，分かり易くなる．

3.2 ヘッダーファイル

3.2.1 ヘッダファイルが必要な理由

先ほどの例では，リスト 3 がヘッダーファイルである．これには関数のプロトタイプ宣言が書かれている．プロトタイプ宣言は，ソースプログラムをコンパイルする際に，引数や戻り値が正しいか?—のチェックのときに参照される．そのため，リスト 1 やリスト 2 をコンパイルするときに必要である．

これらのプログラムをコンパイルするときに，それぞれのファイルの先頭にプロトタイプ宣言を書くと二度手間である．同じことを 2 つのファイルに書かなくてはならない．100 個のソースファイルに使われている関数があれば，100 個のソースファイルに同じプロトタイプ宣言を書くことになる．そうなると，たぶん間違いが生じる．そこで，プロトタイプ宣言をヘッダーファイルにまとめて，それぞれのソースファイルで読み出すことが考えられた．リスト 1 やリスト 2 の`#include "functions.h"`は，指定されたファイル—ここでは functions.h —を呼び出して，この部分に書けという命令である．

3.2.2 ヘッダーファイルの書き方

参考文献 [2] によると，ヘッダーファイルに書くべきものは，以下の通りである．

- 外部に公開するマクロ定義 (関数型のマクロの定義)
- 外部に公開する定数の定義 (`#define` や `enum` による定義)
- 外部に公開する構造体，共用体の定義
- 外部に公開する方の定義 (typedef による型の定義)

- グローバル関数のプロトタイプ宣言
- グローバル変数の extern 宣言

ようするに分割した他のソースファイルでも使うものをヘッダーファイルに書け—ということである．これらヘッダーファイルに書くべきものは，分割されたソースファイルのコンパイルに必要なものである．リスト 3 は，グローバル関数のプロトタイプ宣言が書かれている．

3.3 Makefile

3.3.1 Makefile が必要な理由

複数のソースファイルがある場合，ひとつひとつコンパイルする作業は大変である．コンパイル方法を書いたファイルをひとつ作り，それを実行させることによりコンパイルできれば便利である．

このような用途のために Makefile がある．リスト 1〜3 から構成されるソースファイルは，リスト 4 のようなメイクファイルにより，コマンド「make」一発でコンパイルできる．極めて便利である．構成するファイルの数が多くなればなるほど，この恩恵にあずかれる．

3.3.2 Makefile の書き方

Makefile の書き方の基本は，次の通りである．

```
ターゲット ： 構成ファイル
        コマンド行
```

ターゲットはコマンド行を実行させて作成されるファイル[1]である．構成ファイルは，ターゲットを構成するファイルである[2]．コマンド行は，コマンドを書く．その先頭は,「Tab」ではじめなくてはならない．スペースはダメである．

リスト 4 の例を見ると

```
main.o :  main.c
        gcc -c main.c
```

となっている．オブジェクトファイル main.o がターゲットで，main.c がその構成ファイルである．オブジェクトファイルは，コマンド行に示すように，gcc にオプション (-c) を付けて作成する[3]．オブジェクトファイルとは，ほとんど機械語でできたファイルで，必要なオブジェクトファイルをまとめる (リンク) と実行ファイルができあがる．オブジェクトファイルから，実行ファイルを作成する部分は Makefile 上で

```
mkdat : main.o functions.o
        gcc -o mkdat main.o functions.o -lm
```

である．

Makefile には，この他にもたくさんの便利な機能がある．それを書き出すだけでも一冊の本になるくらいである．諸君が長いプログラムを書くときに，Makefile のさまざまな機能の書き方に付いて学習すれば良いだろう．

[1]実際には，コマンド行を実行してもターゲットのファイルが作成される必要はない．
[2]必ずしも，実際にターゲットを構成するファイルである必要も無い
[3]ターミナル上でも,「gcc -c main.c」とすると main.o というオブジェクトファイルができる

4　プログラム作成の練習

[**練習 1**]　リスト 1〜4 を書き換えて，

$$f(x) = x^2 + x * cos(x) \tag{1}$$

の値をファイルに書き出すプログラムを作成せよ．

[**練習 2**]　ここで学習した分割コンパイルの手法を使って，次のようなプログラムを作成せよ．

- キーボードから整数を読み込む．
- それをファイルに書き出す．

[**練習 3**]　ここで学習した分割コンパイルの手法を使って，次のようなプログラムを作成せよ．電子タイプライター見たいなもの．

- キーボードから文字列を読み込む．
- それをファイルに書き出す．
- end と打ち込まれたら，ファイルを閉じてプログラムを終了する．

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) ここで学習した分割コンパイルの手法を使って，次のようなプログラムを作成せよ．

- キーボードから文字列を読み込む．
- それをファイルに書き出す．

[**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．

5.2　レポート 提出要領

期限	5 月 15 日 (火) AM 8:45
用紙	A4 のレポート用紙．左上をホッチキスで綴じて，提出のこと．
提出場所	山本研究室の入口のポスト
表紙	表紙を 1 枚つけて，以下の項目を分かりやすく記述すること． 授業科目名「情報処理応用」 課題名「課題　分割コンパイルと Makefile」 提出日 2E　学籍番号　氏名
内容	2 ページ以降に問いに対する答えを分かりやすく記述すること．

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] 坂井弘亮. C 言語 入門書の次に読む本. 技術評論社, 2003.