目次

★6　ソースファイルを分けよう

だんだん長いプログラムを書くようになると，

- 一つのソースファイルにだらだら書くのは鬱陶しいからどうにかしたい
- よく使う関数等を「部品」として，いろんなプログラムで使い回したい

という気がしてくるかもしれない．今回説明するように「**ソースを複数のファイルに分けて作っておき，あとでそれらを一つの実行ファイルに合体させる**」という方法をとることで，このような欲求を満たすことができる．

★6.1　ソースファイルを分割する

右のソースファイルを例に説明する．このプログラムをコンパイルして onesource01 という実行ファイルを作成するには，

```
$ cc onesource01.c  -o  onesource01
```

とすればよい．作成された実行ファイルを実行するには，

```
$ ./onesource01
```

とすればよい．

このプログラムの場合，↓のようにソースファイルを二つに分割することができる．main01.c は関数 main の定義を含み，hoge01.c は関数 hoge の定義を含んでいる（この例では一つのソースファイルに一つの関数定義しかないが，一つのソースファイルに複数の関数定義を含めても構わない）．ソースファイルの名前はこれまでと同様に自由につけることができる．

onesource01.c

```
 1  #include <stdio.h>
 2
 3  // 関数 hoge のプロトタイプ宣言
 4  int hoge(int x);
 5
 6  int main(void)
 7  {
 8    int a = 123;
 9    printf("%d\n", hoge(a));
10    return 0;
11  }
12
13  // 関数 hoge の定義
14  int hoge(int x)
15  {
16    return x * 2;
17  }
```

main01.c

```
 1  #include <stdio.h>
 2
 3  // 関数 hoge のプロトタイプ宣言
 4
 5
 6  int main(void)
 7  {
 8    int a = 123;
 9    printf("%d\n", hoge(a));
10    return 0;
11  }
```

hoge01.c

```
1  // 関数 hoge の定義
2  int hoge(int x)
3  {
4    return x * 2;
5  }
```

hoge01.c では printf() 等の関数を用いていないので，stdio.h をインクルードしなくともよい（もちろんしても構わない）．

★ 6.2 分割したソースファイルのコンパイルの仕方

上記ソースファイルをコンパイルして実行ファイルを作るには次のようにする．

このように，ソースファイルを分割した場合，コンパイル作業も複数の手順に分割される．実は，われわれが普段コンパイルと呼んでいる作業は，以下のように「(狭い意味の) コンパイル」と,「リンク」という二つの段階に分かれており，ソースファイルを分割した場合には「コンパイル」と「リンク」 を個別に行わなければならないからである (☆ 1)．

- **コンパイル**： 個々のソースを機械語の命令に翻訳して，**オブジェクトファイル**と呼ぶものを作る
- **リンク**： 複数のオブジェクトファイルや**ライブラリ** (☆ 2) を結合するなどして実行可能なプログラムを作る

ソースファイルを 2 つではなく 3 つに分割した場合，3 つのソースファイルをそれぞれコンパイルして 3 つのオブジェクトファイルを作成し，それらをリンクして 1 つの実行ファイルを作ることになる．

☆ 1) ［発展］実は，次のようにまとめてコンパイルすることもできる `cc main01.c hoge01.c -o prog01`

☆ 2) あらかじめ用意されたオブジェクトファイルの集まり．例えば `printf()` の定義は標準 C ライブラリに含まれており，コンパイラが自動的にリンクしてくれている．また，`sqrt()` などの定義は数学ライブラリに含まれており，コンパイラオプションに`-lm` を指定することでリンクしてくれる．ちなみに，`printf()` と `sqrt()` のプロトタイプ宣言はそれぞれ `stdio.h` と `math.h` の中にある．

★ 6.3 分割コンパイルのメリット

ソースファイルを複数に分割するとコンパイル作業も複雑になって面倒なだけのように思えるかもしれないが，このようにする（**分割コンパイル**する，という）ことには次のように大きなメリットがある．

1. **プログラムを部品化して再利用しやすくなる／保守性がよくなる**
   ソースファイルを分割した場合，`main` を含むソースの他は，あちこちのプログラムで部品として利用することができる．よく行う処理をいくつかの関数にまとめておいて自分で再利用することもできるし，他人が作った関数を利用することもできる．プロのプログラマは，複数人でプログラミング作業を分担しながら大きなプログラムを作成するのが普通であるから，分割コンパイルが必須である．
2. **コンパイル時間を短縮できる**
   大規模なプログラムを開発する場合，ソースのコンパイルに要する時間が無視できなくなってくる．ソースファイルを分割してあれば，関係するソースファイルのみ再コンパイルしてリンクすればよいので，実行ファイルを作成するのにかかる時間を大幅に減らすことができる．

Q1. 前頁の main01.c と hoge01.c をコンパイルしてオブジェクトファイル main01.o と hoge01.o を作成した後に，main01.c の 8 行目の 123 を 456 に変更した．このとき，コンパイルし直して実行ファイル prog01 を作るにはどうしたらよいか．

Q2. Q1 の変更を行ってコンパイル・実行した後に，今度は関数 hoge が引数の 3 倍の値を返すようにソースを変更したとする．このとき，コンパイルし直して実行ファイル prog01 を作るにはどうしたらよいか．

★6.4 再利用してみよう

ソースファイル hoge01.c で定義された関数 hoge を別のプログラムに使い回してみよう．hoge01.c と同じディレクトリ内に次のようなソースを作ったとする．

main02.c

```
 1  #include <stdio.h>
 2
 3  // 関数 hoge のプロトタイプ宣言
 4  int hoge(int x);
 5
 6  int main(void)
 7  {
 8    int a = 5, b;
 9    b = hoge(hoge(a));
10    printf("%d\n", b);
11    return 0;
12  }
```

Q3. hoge01.c をコンパイルしてオブジェクトファイル hoge01.o をすでに作成してあった場合，main02.c をコンパイルして prog02 という実行ファイルを作成するにはどのようにしたらよいか．

★6.5 ヘッダファイルを作ろう

次の例を考えてみよう．

main03.c

```
 1  #include  <stdio.h>
 2
 3  // 関数 f, g, h のプロトタイプ宣言
 4
 5
 6
 7
 8  int main(void)
 9  {
10    double a = 0.25, b = 2, c = 1e-1, x;
11
12    x = f(a);
13    printf("%f\n", x);
14    g(&b);
15    printf("%f\n", b);
16    h(c);
17    return 0;
18  }
```

hoge03.c

```
 1  #include <stdio.h>
 2
 3  #define PI 3.14159
 4
 5  double f(double x)
 6  {
 7    return PI * x;
 8  }
 9
10  void g(double *x)
11  {
12    *x = PI * *x;
13  }
14
15  void h(double x)
16  {
17    printf("%f x %f = %f\n", x, PI, PI * x);
18  }
```

この 2 つのソースファイルをコンパイル＆リンクして prog03 という実行ファイルを作成したとすると，実行結果は次のようになる．

```
$ ./prog03
0.785397
6.283180
0.100000 x 3.141590 = 0.314159
```

この例のように，hoge03.c に定義された関数を別のソースファイルから利用するなら，そのソース中に関数のプロトタイプ宣言を記述する必要がある．しかし，この例くらいの数なら問題ないかもしれないが，たくさんの関数を利用する複雑なプログラムを作るようになってくると，プロトタイプ宣言を忘れない＆間違えないようにするのも大変になってくる．そのような面倒を避けるために，**ヘッダファイル** (☆ 3) を作ってそれをインクルードする方法を学ぼう．

☆ 3) ヘッダファイル: header file. C 言語では，なんたら.h という名前のファイル．stdio.h や math.h などがおなじみ．**インクルードファイル**ということもある．

上の例では，以下のような内容のヘッダファイルを作るとよい．

hoge04.h

```
 1  #ifndef HOGE04_H  // HOGE04_H が定義されてなければ（if not defined）
 2  #define HOGE04_H  // 定義しておく（これの役割は講義資料参照）
 3
 4  // 定数の定義
 5  #define PI 3.14159
 6
 7  // 関数 f, g, h のプロトタイプ宣言
 8  double f(double x);
 9  void g(double *x);
10  void h(double x);
11
12  #endif              // #ifndef に対応
```

そして，各ソースは hoge04.h を次のようにインクルードすればよい．こうすることで，main の側にたくさんの関数のプロトタイプ宣言を書く手間を省くことができる．

main04.c

```
 1  #include  <stdio.h>
 2
 3
:   以下 main03.c の 8 行目以降と同じ
```

hoge04.c

```
 1  #include <stdio.h>
 2
 3
:   以下 hoge03.c の 5 行目以降と同じ
```

上記ヘッダファイル中の 1,2,12 行目の役割についてはこの授業ではきちんとは説明しないが，hoge04.h を他のヘッダファイルでインクルードしているような状況で，2 重にインクルードしようとしてしまいコンパイルエラーとなるのを防ぐための技である．上記のように書いてあると，はじめて hoge04.h をインクルードする時はちゃんと中身がインクルードされるが，その時点で，HOGE04_H が定義されるので，2 回目のインクルード以降は #ifndef から #endif までが無視されることになって，多重にインクルードする（そしてエラーとなる）のを防ぐことができる (☆ 4)．

☆ 4) したがって，HOGE04_H のところは，区別がつくような名前であればなんでもよい．

ところで，関数のプロトタイプ宣言や分割コンパイルのことがわかってくると，普段，printf や cos などの関数を利用するプログラムを書く際にそれらのプロトタイプ宣言を書かずに済んでいることが逆に不思議に思えるだろう．これらのプロトタイプ宣言をわざわざ書かなくてもコンパイル&実行ができるのは，システム上に用意されたヘッダファイル stdio.h や math.h （これらの中に関数のプロトタイプ宣言が記述されている）をインクルードしているからである (☆ 5)．

☆ 5) ［発展］ コンパイラは，#include の後ろのヘッダファイル名が <> で囲んであればデフォルトのインクルードファイル置き場からそのファイル名に一致するものを探し，"…"で囲んであればカレントディレクトリから探す．計算機室の環境では，stdio.h や math.h は /usr/include ディレクトリ内に存在する．