8 分割コンパイル

プログラムコードの規模が大きくなったり、複数の人で分担してプログラムを作成すると、1 つのファイルでプログラムを保存するのが非効率となるため、複数のファイルに分けてプログラムを残すようになります。今回は複数のファイルに分けて C プログラムを作った場合のコンパイルについて説明します。

8.1 準備: 複数のファイルに C プログラムを作る

まず、複数のファイルに分かれた C プログラムを作る必要があります。C プログラムは、**同じ目的として使う可能性のある関数を 1 つのファイルにまとめて保存する**ことが多いです（第 6 回の `math.h` や `ctype.h` のように）。

まず簡単な例として「大きい方の整数を返す」関数を考えます。次のように関数の**プロトタイプ宣言だけをヘッダファイルに**入力します。ファイル名は「compare.h」と保存します。（ヘッダファイルには、**複数のファイルで共通で使う**関数の宣言やマクロの定義を含めます。）

```
int max(int x, int y);
```

次に以下のような**ファイルを分けて**作ります。自分で作成したヘッダファイルは**<>ではなく""で囲みます**。

```
/* ファイル名は compare.c */
#include "compare.h"

int max(int x, int y)
{
    int result;
    if(x>y) {
        result = x;
    } else {
        result = y;
    }
    return result;
}
```

```
/* ファイル名は main08.c */
#include <stdio.h>
#include "compare.h"

int main()
{
    printf("mac: %d\n", max(5, 10));
   return 0;
}
```

8.2 Setp1: 複数のファイルをまとめてコンパイルする

このように分けて作成したファイルを簡単にまとめてコンパイルするには、以下のように cc コマンドにファイルを**複数指定します**。

```
$ cc compare.c main08.c
```

また、コンパイルして生成する実行ファイルを「a.out」以外のファイル名を指定する場合は、以下のように「-o」オプションを使います。

```
$ cc -o myexe compare.c main08.c
$ ./myexe
```

しかし、この方法では「ひとつのファイル内の変更」でも、毎回「**全ての C プログラムファイルをコンパイルし直す**」ことになってしまい無駄なコンパイルが増えてしまいます。

8.3 Setp2: オブジェクトファイルを生成してからリンクする

「変更された C プログラムファイルだけのコンパイルで済む」ようにするために、C プログラムをコンパイルする際に、一度オブジェクトファイルにしてからリンクして実行ファイルを作るという方法を使います。

以下のように「-c」オブションを使うと、オブジェクトファイルが生成されます（拡張子が「.o」のファイル)。その後、オブジェクトファイルに対して、これまで通りに cc を使うとオブジェクトファイルをリンクしてくれます。

```
$  cc  -c  compare.c
$  cc  -c  main08.c
$  cc  -o  myexe  compare.o  main08.o
```

このようにすると、**「変更した C プログラムファイルだけのオブジェクトファイル生成で済む」ことになり**、無駄なコンパイルがなくなります。しかし、ファイル数が増えると、どれが変更したファイルなのか把握できなくなり、手作業では大変になってきてしまいます。

8.4 Setp3: make コマンドを利用して分割コンパイルする

「変更された C プログラムファイルだけをコンパイルして実行ファイルを生成する」作業を自動化するために「make」というコマンドを使います。

次のような「**make のための設定ファイル**」を入力します。ファイル名は「Makefile」と保存します。

```
myexe: compare.o main08.o
        cc -o myexe compare.o main08.o
compare.o: compare.c
        cc -c compare.c
main08.o: main08.c compare.h
        cc -c main08.c
clean:
        rm -f myexe compare.o main08.o
```

Makefile の中身は、基本的に以下のような要素がいつくか組み合わさって構成されています。「ターゲットを作るには、ソース（複数ある場合は並べて書く）となるファイルを必要とし、コマンドによって作られる」ことを表しています。なお、**コマンドの字下げはタブを 1 つ入れます**（空白ではエラーになります)。

```
ターゲット: ソース
        コマンド
```

このようにして作った Makefile を使って、コンパイルするには以下のように make コマンドを入力します (make のように複数のコンパイルをして 1 つの実行ファイルを作ることを「ビルド」と言います)。

```
$  make
$  ./myexe
```

make コマンドは、コマンドの後ろにターゲット名を指定することができますが、省略すると Makefile の先頭にあるターゲットをビルドしようと実行します。以下のようにターゲット clean を指定すると「オブジェクトファイルと実行ファイルを強制的に削除」してくれます。

```
$  make  clean
```

8.5 課題

【練習 8-1】 8.1 節〜8.4 節の説明にしたがって、分割コンパイルを試してみましょう。

【課題 8-1】 8.4 節の make では、「全てのファイルをコンパイルする」場合は以下のようになります。

```
$ make
cc -c compare.c
cc -c main08.c
cc -o compare compare.o main08.o
```

「main08.c」に、別の整数の大小を比較する関数 max() の呼び出しを追加してみましょう。その際の make の結果は以下のように「main08.c だけがコンパイルされる（つまり、compare.c は変更されていないためコンパイルされない)」ことも確認しましょう。

```
$ make
cc -c main08.c
cc -o compare compare.o main08.o
```

【課題 8-2】 ファイル「compare.c」に、「2 つの整数を比較した結果を返す」機能をもった関数 compare() を追加してください。この関数のプロトタイプ宣言は以下のようになります。

```
int compare(int x, int y);
  /* x と y を比較しその結果に応じて以下のように整数を返す */
  /* x が y より大きい場合、1 を返す */
  /* x が y より小さい場合、-1 を返す */
  /* x と y が等しい場合、0 を返す */
```

- 関数のプロトタイプ宣言を「compare.h」に追加しましょう。
- 「compare.c」に関数 compare() を作りましょう。
- 「main08.c」の main() に動作確認の処理を追加しましょう（以下参照)。

```
[main での処理]
printf("compare: %d\n", compare(10, 20));
printf("compare: %d\n", compare(10, 10));
printf("compare: %d\n", compare(10, -10));
[実行結果]
compare: -1
compare: 0
compare: 1
```

【課題 8-3】 新しいファイル（例えば「range.c」）に、「引数 x で指定した整数の階乗（1〜x のかけ算）を計算して返す」機能をもった関数 factorial() を作成してください。この関数のプロトタイプ宣言は以下のようになります。

```
int factorial(int x);
  /* 引数 x に対して階乗を計算し、その計算結果を返す */
  /* 1〜x を繰り返しながらかけ算をして計算する */
```

- 関数のプロトタイプ宣言を「range.h」に追加しましょう。
- 「range.c」に関数 factorial() を作りましょう。
- 「main08.c」の main() に動作確認の処理を追加しましょう（以下参照）。
- Makefile に以下の内容を追加しましょう。
  ターゲット「range.o」を追加（ターゲット「compare.o」参考）
  ターゲット「myexe」「clean」へ、range.o に関する部分を追加
  ターゲット「main08」へ、range.h に関する部分を追加

[main での処理]
```
printf("factorial: %d\n", factorial(4));
```
[実行結果]
```
factorial: 24
```

【課題 8-4】 （まだ余裕のある人は…）課題 8-3 で作成したファイル（例えば「range.c」）に、「引数で指定した整数の加算を計算して返す」機能をもった関数 summation() を作成してください。この関数のプロトタイプ宣言は以下のようになります。

```
int summation(int x);
  /* 引数 x に対して 1〜x の合計を計算し、その計算結果を返す */
  /* 1〜x を繰り返しながら加算して計算する */
```

課題 8-2 の手順を参考に、この関数を追加してみましょう。

[main での処理]
```
printf("summation: %d\n", summation(4));
```
[実行結果]
```
summation: 10
```