基礎プログラミング演習 II 教材 (#2)

■ プログラムの実行

一つプログラムを作って実行してみましょう。

□ 画面に文字を表示する（だけの）プログラム

Emacs を起動して、下のプログラムを入力してください。ファイル名は任意の名前で結構ですが、例えば hello.c とでもしておいてください。

```
/*
    printfだけのプログラム
*/
#include <stdio.h>
#include <stdlib.h>

main(){
    printf("My name is Enokida Yuuichiro!");
    exit(0);
}
```

入力ができれば、まず保存をしてください。正しく保存できたかどうか、ls コマンドで調べ、cat コマンドで中身を確認してください。
次に cc　コマンドでコンパイルします（このコンパイルという作業の意味は後述）。

```
$ cc –o hello hello.c
$
```

その後 ls すると、hello というファイルが増えていることがわかるでしょう。これを ./hello として実行して下さい。画面に名前（ここでは Enokida Yuuichiro!）が表示されるはずです。以下は実際の作業例です。自分でタイプしたところには下線（と色）をつけてあります。

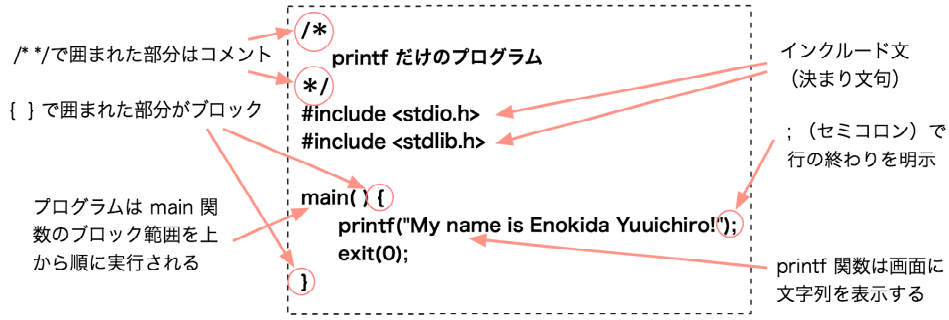
```
$ ls
Desktop        Library        Music          Public         hello.c
Documents      Movies         Pictures       Sites
$ cat hello.c
/*
.... (中略) ......
    exit(0);
}
$ cc -o hello hello.c
Desktop        Library        Music          Public         hello  ←増えた
Documents      Movies         Pictures       Sites          hello.c
$ ./hello
My name is Enokida Yuuichiro!$
```

それができたら、今度は Emacs で ”My name is …” の部分を自分の名前に変更して保存し直し、再び cc コマンドによってコンパイルし、./hello によって実行して、変更が反映されていることを確認してください。

これが「C 言語でプログラムを作成し、実行する（そして修正してまた実行する）」という一連の作業の最も簡単な形になります。

□ プログラムの解説

プログラムの内容を行ごとに説明します。



・ /* と */ で囲まれた部分はコメントです。
・ 冒頭の #include は今は決まり文句として覚えておいて下さい。
・ { } によって囲まれた範囲をブロックと呼んでいます。
・ main() 直後の { でブロックが始まり、最下行の } でブロックが終わっており、これが main() 関数の範囲を示しています。（関数についてはクラス後半で）
・ printf() 関数は画面に文字を表示する機能を提供します。
・ 最後には ；（セミコロン）があり、これが printf() 関数の行の終わりを示しています。
・ プログラムは main() 関数の中身を上から順に一行ずつ実行し、exit()関数で実行停止します。

□ わかりやすいプログラムの表記

冒頭のコメント、6 行目の空白や、printf() 関数の前の空白は、なくても構いません。より見やすくなると思い、入れているものです。これらをすべて省いてもプログラムとしては正しく機能します。つまり、上のプログラムはこのように書くこともできます。

```
#include <stdio.h>
#include <stdlib.h>
main(){printf("My name is Enokida Yuuichiro!");exit(0);}
```

ただ、長いプログラムでこのような書き方をすると、プログラムが読みにくくなってしまいます。上に示したようにプログラムの中の文には構造があり、記号にも役割があるのですから、それらがよりはっきり分かりやすくなるよう、表記について努力すべきです。

□ 画面への出力を制御する

5 行目の printf() 関数と、その実行結果である画面上の表示に注目してください。

プログラム：

```
  printf("My name is Enokida Yuuichiro!");
```

実行結果：

```
My name is Enokida Yuuichiro!$
```

プログラムによって出力された（画面上に表示された）”My name is …” の行のすぐ右にコマンドプロンプト（例では「$」）が表示されて格好が良くありません。（通例、プロンプトは行の左端に表示されるものなので。）

これは printf() の出力指定に「表示したあとで改行する」という指示が含まれていないからです。改行表示がないと、出力は次々に右につながって表示されつづけます。

実験：
以下のような printf() の書き方をするとどのような出力結果になるか試してください
（各行の終わりを示す; を忘れないように注意。）

```
printf("My name ");
printf("is");
printf("Enokida Yuuichiro!");
```

C 言語での改行指定は、表示する文字列に「 \n 」（バックスラッシュと n ）を含めることで行います。具体的には以下のように書きます。

```
printf("My name is Enokida Yuuichiro!\n");
```

つまり「 \n 」は「改行するための文字(*1)」であり、これを画面上に出力すると、その次に出力された文字は次の行の左端から表示がはじまります。

*1 実際に C コンパイラは \n を \ と n という二文字としては扱わず、一文字（ASCII コードで 0a (10 番目)の文字）として扱っています。

□ 制御文字・エスケープシーケンス

こうした \ による特別な文字の表現は他にもあります。

\n -- 改行
\t -- タブコードを表示する
\\ -- \ を表示する
\' -- ' （シングルクォーテーション）を表示する
\" -- " （ダブルクォーテーション）を表示する

など。（興味のある受講生は教科書 p.145 および p.149 表 9.1 も参照）

□ 課題 1.

以下のような実行結果になるよう printf() の記述を変更し、実行してください。ただし名前は自分の名前となるように。

```
$ ./hello
My name is
Enokida Yuuichiro!
I like "Sushi" much.
$
```

余裕のある受講生は printf を一行で行う場合、あるいは逆に三行で行う場合など、さまざまなものを試してみると良い。（どの書き方が分かりやすいか、といったことを考えるチャンスです）

■ 計算式

下のプログラムを入力、コンパイル、実行してください。時、分、秒をもとに、0 時から数えた秒数を計算して表示するものです。
サンプルでは 10 時 32 分 45 秒としていますが、時間は適当に現在時刻を入れて下さい。ファイル名は任意の名前で結構ですが、例えば comp.c とでもしておいてください。

```
#include <stdio.h>
#include <stdlib.h>

/*
  秒数を計算する 473088 榎田裕一郎
*/

main() {

  printf("今は %d 時 %d 分 %d 秒です\n", 10, 32, 45);
  printf("全部で %d 秒めですね\n", 10 * 3600 + 32 * 60 + 45);

  exit(0);
}
```

押さえて欲しいポイント：
・/* と */ で囲まれた範囲はコメントです。
（コンパイル時には無視されるので、プログラム以外の覚え書きなどを書いておくのに便利。）
・C プログラムの中では数値や計算式が書ける。
・数値を printf で表示させるためには %d といった変換文字による指定を行う。
・exit(0) 関数でプログラムの実行を終了する。（0 は正常終了を意味する）

□ 演算子

C プログラムのなかでは各種の計算が記述できます。四則演算の表記法は以下の通りです。
+, * といった記号を演算子と呼んでいます。

演算子	意味	表記例と計算結果	記号の読み方
+	加算（＋）	20 + 6 （結果は 26 になる）	プラス
-	減算（−）	20 - 6 （結果 は 14 になる）	ハイフン
*	乗算（×）	20 * 6 （結果 は 120 になる）	アスタリスク
/	除算（÷）	20 / 6 （結果 は 3 になる）	スラッシュ
%	剰余（割った余り）	20 % 6 （結果 は 2 になる）	パーセント

整数どうしの割り算 (/) の結果は（小数点以下を切り捨てて）整数となります。
例えば 6 / 20 は 0 です。

これらの「記号の左右両側にある二項に対して働く演算子」を二項演算子と呼びます。それに対して「- 2 * 4」のように、符号反転を意味する「-」（ハイフン）演算子もあります。

これは二項演算子としてのハイフンとは異なる単項演算子であり、この場合は − の左に値や計算式が無いことによって単項演算子のマイナスであると判断します。

□ 優先順位

最後に、演算子には通常の計算式と同じく優先順位があります。*, /, % が +, - より高い優先順位で処理されます。単項演算子の – はこれらすべてに優先されます。また、カッコを使った演算順序の制御もできます。
以下の計算式 2 例は、それぞれ同一の結果を返します。

```
10 * 3600 + 32 * 60 + 45
( 10 * 3600 ) + ( 32 * 60 ) + 45
( ( ( ( 10 ) * 3600 ) + ( 32 * 60 ) ) + 45 )
(10*3600)+(32*60)+45
```

```
- 2 * 4 + - 3 * - 4
(- 2) * 4 + (- 3) * (- 4)
-2 * 4 + -3*(-4)
(-2)*4 + (-3)*(-4)
```

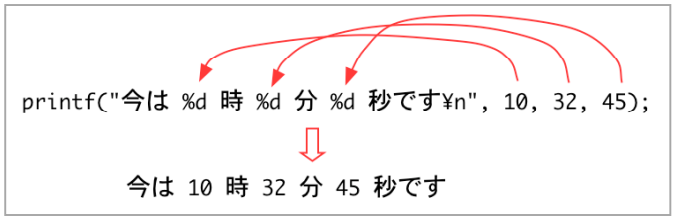
カッコを用いて誤読されない読みやすい表記の重要さがわかります。演算子の優先順位の複雑さについては教科書 p.28 の 2.3.2 「簡単な演算子と優先順位」も参照。

□ printf の変換文字

printf( ) のカッコのなかにはカンマで区切られた幾つかの項目があります。これらは引数と呼ばれます。
最初の引数には書式（フォーマット）を指定するための文字列を与えます。printf はこの文字列の記述に従って出力結果を整形します。第二引数以降は数値や計算式などを書くことが出来ます。

書式指定にある「今は %d 時」の %d は「ここに 10 進数(decimal) の数値をあてはめる」という指定で、第二の引数である 10 が対応します。こういった % に続くを行う文字のことを「変換文字」と呼んでいます。（d 以外にもある。次回以降説明。）

続く二番目の %d には第三引数である 32 が、三番目の %d には第四引数の 45 が対応し、最終的には「今は 10 時 32 分 45 秒です」と整形されて出力（画面に表示）されます。



□ 課題 2.

サンプルのプログラムを、秒数から時分秒を算出するように変更してください。具体的には printf( ) の記述を以下のように変更し、箱抜き部分に適切な計算式を入れます。

```
printf("今は %d 秒めです\n", 12345);
printf("それは %d 時 %d 分 %d 秒ですね\n", 12345/3600, [      ], [      ] );
```

■ 宿題

課題は 1. 2. ともに Moodle で提出すること。
次回の教材に目を通しておくこと。