2010 年度春期

情報処理III　～ C 言語の基本 ～

月曜 4 時限 コンピュータルーム B

担当：角皆　宏

0　授業の準備

(1) 環境設定の確認 (auto-logout, path 等)
(2) 本授業用ディレクトリの作成 (`joho3` 等の適当な名前で)

ホームディレクトリ直下では
作業をしない!!

1　C 言語によるプログラムの記述から実行まで

ここではプログラム名を `hello.c` として、実行までの流れを説明する。

(0) プログラムを書いて何をさせたいか考える。
(1) プログラム (ソース) をエディタ (Emacs 等) で記述する。
　(a) エディタを起動 (例えば `emacs hello.c` Enter)
　(b) 入力・編集
　(c) 記述し終えたら、そのファイルを保存 (`C-x C-s`)
(2) 記述したプログラムを**コンパイル**(compile) して実行形式を生成する。
　(a) Unix での C プログラムのコンパイルには `cc` というコマンドを用いる。
　　`cc -o hello hello.c` Enter
　　とすると、`hello.c` がコンパイルされて `hello` という名前の実行形式が生成される。
　(b) 何らかのエラーが出たら、もう一度エディタに戻って修正する。
(3) 実行する。
　(a) 作成された実行形式は UNIX のコマンドとして次のようにタイプして実行することができる。
　　`./hello` Enter
　(b) 期待通りの動作をしなかった場合、エディタに戻ってデバッグする。
(4) 上記を期待の結果を得るまで繰り返す。(→初めに戻る)
(∞) 更に良い方法はないか、もっと色々なことが出来ないか考える。

C 言語で記述したプログラムは、
ファイル名の最後 (拡張子) が`.c`
であるファイル名で作成する。

`-o` オプションは生成ファイル名の指定。
指定しないと `a.out` という名前になるが、これでは元のソースが何か判らないし、他のソースをコンパイルした時にも上書きされてしまうので良くない。通常はソースファイル名から最後の `.c` を除いたものにする。

`cc`：C Compiler

エラーが出た時は、表示された**エラーメッセージをよく読む**こと!!
バグ (bug): プログラム中の間違い
デバッグ (debug): それを取り除く作業

2 Hello, World! ～ 最初の例・表示 (出力) ～

実習 2.1. ファイル名を hello.c として以下のプログラムを入力し、コンパイルして実行する。

```
/*   hello.c   2010-04-12   */

#include <stdio.h>

int main(int argc, char** argv)
{
  printf("Hello, World!!\n");
}
```

実行例 . “=>” はプロンプトで、それに続く文字列は自分で入力したもの。“Hello, World!!” が表示された (であろう) 文字列。

```
=> cc -o hello hello.c
=> ./hello
Hello, World!!
=>
```

考察 2.1.1. printf() という命令は、" " で囲んだ文字列をそのまま表示してくれるようだが、最後の \n は表示されていない。これは何の意味があるのか？

ファイル名を指定して起動
emacs hello.c Enter
するか、単に
emacs Enter
として起動後に、ファイルの読込 (C-x C-f) でファイル名 hello.c を指定

Emacs は、拡張子 .c のファイルを読み込むと C のプログラムの記述に便利なモード (c-mode) に入る。例えば、自動的に随時適切な位置まで「字下げ」をしてくれたりする (詳細後述)。

入力し終えたらそのファイルを保存 (C-x C-s)。通常は更にそのファイルの編集が続くので、Emacs を終了 (C-x C-c) せずに、中断 (C-z) するか、別ウィンドウを利用すると便利。

実習 2.2. printf() の部分を複数にしてみる。その際、\n を付けたり付けなかったりしてみて、違いを見る。

「使い回しの心」
hello.c をコピーして hello2.c を作成しそれを編集するか、新規に hello2.c を作成する際にファイルの挿入 (C-x i) を利用すると便利。

```
/*   hello2.c   2010-04-12   */

#include <stdio.h>

int main(int argc, char** argv)
{
  printf("Hello, World!! 1");
  printf("Hello, World!! 2\n");
}
```

考察 2.2.1. 例えば、

```
printf("Hello, World!! 1\nHello, World!! 2\n");
```

など、いろいろ試してみよ。\n の役割は？

実習 2.3. hello.c の中の

```
printf("Hello, World!!\n");
```

の最後の ';' を取り除いてコンパイルしてみよ。

C 言語のプログラムの基本的な構造について、先程の `hello.c` を題材に見ていこう。(各項目の詳細は後日)

- `/*` と `*/` とで囲まれた部分はコメント (注釈)。コンパイル時に読み飛ばされる。(文字列定数の `" "` 内を除く。)
  - ⋆ コメントを入れても生成される実行形式は同じで、プログラムの実行には関係ない。
  - ⋆ 複数行に亙る注釈も可。
- `#include` は、そのプログラム (で使われる関数) に応じて必要な「ヘッダファイル (header file)」を、コンパイルする時に読み込むべきことの宣言。
  - ⋆ `stdio.h` は STanDard Input/Output (=標準入出力) の意で、数値や文字の入出力に必須のヘッダファイル。(`hello.c` では、関数 `printf()` を用いるのに必要。)
- C 言語のプログラムは、1 つ以上の「関数」によって構成される。
  - ⋆ プログラムには `main()` という名前の関数がただ 1 つ含まれ、`main()` がプログラム中のどこにあっても、実行はこの `main()` の冒頭から開始される。
  - ⋆ `main()` 以外の関数は他の関数から呼び出される。
  - ⋆ 関数は 0 個以上の引数を `()` 内に取り、値を 1 つ返したり返さなかったりする。
- C 言語では、関数やループの範囲などの “ひとかたまり” (ブロック, 文の集まり) を `{` と `}` とで囲む。
- C 言語の一つのステートメント (文) は、必ず ’`;`’ (セミコロン) で終える。
- `" "`で括ったものは文字列定数を表す。
  - ⋆ この中で改行してはいけない。改行文字を含めたい時は `\n` と書く。

実行速度が遅くなったりする心配は不要。
未来の自分の為に適切に入れるべし。
後述の空白や空行についても同様。

`studio.h` ではない!!

`main()` の所の
`int main(int argc, char** argv)`
という形にはちゃんと意味があるが、取り敢えずこう書くものとしておく。

’`;`’ は文の終わりを示すもので、文の区切りではない。実習 2.3 参照。

注 . 「字下げ (indent)」について:

- 空白や空行は、プログラムのソースのどこにどのように入れてもよく、生成される実行形式は同じで、プログラムの実行には関係ない。(文字列定数の `" "` 内を除く。)
- `{` と `}` とで囲まれた一つのブロックは字下げするなど、字下げの様式を統一して見易くすべし。
- Emacs の c-mode を活用すべし。
  - ⋆ `{` と `}` とが対応する位置になるように自動的に字下げ
  - ⋆ 或る行で Tab キー → その行を適切な位置まで自動的に字下げ

自由書式 (free format) という。

思い通りに字下げがされないのは、`{` と `}` との対応がきちんと取れていない場合が殆ど。良く見直すべし。取り敢えず `}` を打ってみるのも有効。

3　計算をさせてみよう ～四則演算～

プログラムというからには何か計算をさせないと気分が出ないよね。

実習 3.1. 2 つの整数 x, y の加減乗除を行ない、その結果を表示させてみよう。以下のプログラムを入力し、コンパイルしてみよ。

%d 等については次回に。ここでは取り敢えず打ち込んで試してみよう。

```
/*   arith.c   2010-04-12   */

#include <stdio.h>

int main(int argc, char** argv)
{
  x = 2010;
  y = 4;

  printf("%d + %d = %d\n", x, y, x+y);
  printf("%d - %d = %d\n", x, y, x-y);
  printf("%d * %d = %d\n", x, y, x*y);
  printf("%d / %d = %d\n", x, y, x/y);
}
```

arithmetic: 算術
arithmetic operation: 算術 (四則) 演算

ここでも「使い回しの心」を発揮しよう。

実行例． 多分このようなエラーが出て止まってしまったのではなかろうか。

エラーメッセージを読め!!

```
=> cc -o arith arith.c
"arith.c", line 7: undefined symbol: x
"arith.c", line 8: undefined symbol: y
cc: acomp failed for arith.c
```

エラーメッセージを読んでみると、どうやら 7 行目の記述に対し「x は定義されていない記号だよ」と、また、8 行目の記述に対し「y は定義されていない記号だよ」と言っていて、arith.c のコンパイルに失敗したようだ。

実は C 言語では、プログラム内で利用する変数 (詳細は次回) をあらかじめ「宣言」しておく必要がある。「宣言」は「型」(これも詳細は次回) と変数名との組で行ない、それによってメモリ上にその変数用の領域が確保される。変数は型によって、その変数に代入 (これもまた詳細は次回) できる値が決まる。(詳しく言うと、確保するメモリ領域の大きさと、そこに書き込まれたものの扱い方が決まる。)

宣言: declaration

今回のプログラムでは 2 つの整数型の変数 x, y を用いたいので、プログラムの始めに

```
int x, y;
```

として宣言する。

int: 整数型 (INTeger)

注． 一つづつ

```
int x;
int y;
```

としても同じだが、同じ型ならまとめて宣言できる。

実習 3.2. 変数宣言文を次のように一行追加してコンパイル、実行してみよ。

注釈は付け加えなくてもよろしい。

```
/*   arith.c   2010-04-12   */

#include <stdio.h>

int main(int argc, char** argv)
{
  int x, y;     /* add this line! */

  x = 2010;
  y = 4;

  printf("%d + %d = %d\n", x, y, x+y);
  printf("%d - %d = %d\n", x, y, x-y);
  printf("%d * %d = %d\n", x, y, x*y);
  printf("%d / %d = %d\n", x, y, x/y);
}
```

考察 3.2.1. 数値をいろいろ変えて動作・結果を確認せよ (負の整数など)。数値を書き変える度にコンパイルし直すのを忘れずに。

勿論、本当は一々ソースを書き直してコンパイルすることなく、後から数値を入力 (キーボードやデータファイルから) するようにしたいのだが、それは次回以降に。

課題 1 (✗切 4/18(日)). 3 つの整数の和と平均を計算し、その結果を表示するプログラム `int3.c` を作成せよ。(変数の値は適当に与えよ。)