● 先週のプログラムのキーポイント

【コメント】

「/*」から「*/」までの間は**コメント**と呼ばれ, プログラムの制御には直接関係しない部分である. コメントには, あとからプログラムをみたときに容易にわかるように, そのプログラムが何をするものなのかなどを書く.

【プログラムの基本構造】

プログラムは

```
#include <stdio.h>
int main(int argc, char **argv)
{

return 0 ;
}
```

という形をしている.

- 「#include <stdio.h>」と「return 0」の部分は, 今のところは「おまじない」と思っておこう.
- プログラムは main の先頭から順次実行される.
- プログラムの各行は「;」で終わっている. これも, 今のところ「そんなものだ」と思っておこう.

【画面への出力】

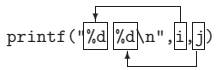
「printf("Hello World\n")」の実行例からわかるように,「printf」はその中に書かれた「"」で囲まれた部分を「画面」に出力する. ここで「\n」は「改行」を表している.

【変数】

- 「int i」,「int i, j」などは「値を格納する場所」を定義している. (「値を格納する場所を定義して, そこに i, j などの名前をつけている」と理解するのがよい.) これを**変数**と呼ぶ.
- 「i = 1」とは, 変数 i に値 1 を**代入**している.
- 「i+j」は, 変数 i に入っている値と, j に入っている値の「和」をとる.

【値の出力】

値を出力するには printf を用いて,「printf("%d %d\n",i,j)」などとする. この時 %d の部分に**対応する**値が出力される. この対応づけは, 左から順に「%d」が見つかるたびに, 後ろに書かれた変数や定数の値が出力される.



★ 重要な注意

「コマンド」を実行する場合には, そのコマンドが何を意味しているか, どのような結果が得られるのかを, 各自で理解した上で実行しなければならない. (もちろん, やってみてから何が起ったかを考えてもよい.) そのためには, 配布資料の該当部分と実行結果 (または, その予測) を見比べることが重要である.

● 実習内容

【UNIX】

【シェルとワイルドカード】

- "ls␣-alg" コマンドを利用して, カレントディレクトリのファイルの一覧を表示させる.
- "ls␣-alg␣|␣more" コマンドを利用して, カレントディレクトリのファイルの一覧を表示させる. 上の場合と比べて何がうれしいのかを考えてみよう. (コマンドのパイプの意味を考えてみよう.) (資料 section 4.4.2.2, section 4.5.2.6)
- foo1.txt, foo2.txt, bar.txt という 3 つのファイルを作成してから, 「ワイルドカード」を用いて, foo1.txt と foo2.txt だけを消去することを考えてみよう. (資料 section 4.4.2)
- foo1.txt, foo2.txt, bar1.txt という 3 つのファイルを作成してから, 「ワイルドカード」を用いて, foo1.txt と bar1.txt だけを消去することを考えてみよう. (資料 section 4.4.2)

★ **発展課題** foo1.txt, foo2.txt という 2 つのファイルのみがあった状態で, これらをそれぞれ bar1.txt, bar2.txt という名前のファイルに置き換えたい. これを "mv␣foo?.txt␣bar?.txt" により望む結果が得られるのか? もし, そうでないならば, その理由は何か?

【ディレクトリ操作 (1)】

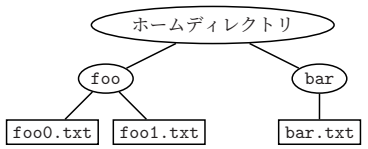
- cd コマンドを実行してみる. (資料 section 4.5.1.1)
  さらに, pwd コマンドを実行してみる. (資料 section 4.5.1.2)
  これは何をしているのか, さらに何が起ったのかを考えよう.
- mkdir コマンドを利用して, サブディレクトリを作成する. (資料 section 4.5.1.3) (サブディレクトリの名前は各自が考えることである.)
- cd コマンドを利用して, 作成したサブディレクトリに移動する. (資料 section 4.5.1.1)
- 上で作成したサブディレクトリを削除する. これには rmdir コマンドを利用すれば良い. (資料 section 4.5.1.4)

【ディレクトリ操作 (2)】

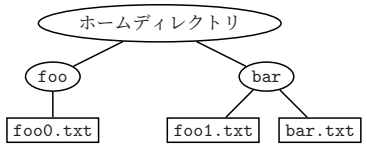
以下では, ここまでに学んだコマンドを利用して, ディレクトリ構造を「いじりまわす」練習である.

その際には「既存のファイル」を消去しないように十分な注意を払う必要がある. 「既存のファイル」を消去してしまうと, ログインできなくなったり, いろいろな不都合が生じる可能性がある.

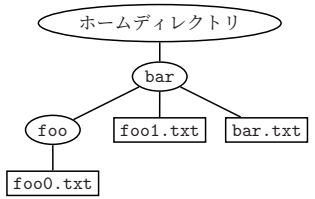
- 下に書いたようなディレクトリ構造を各自で作成しよう.
  (楕円で囲まれたものがディレクトリで, 四角のものはファイル. なお, ファイルは emacs を使って「ちょっとしたもの」を作成すればよい.)



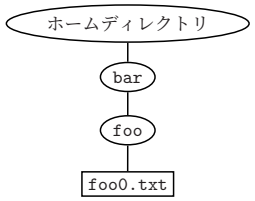
- 各種のコマンドを用いて, 上で作成したディレクトリ構造から, 次のディレクトリ構造へつくり直そう.



- さらに次のディレクトリ構造へつくり直そう.



- 次に,「ワイルドカード」を利用して, 一部のファイルだけを消去して, 次のディレクトリ構造へつくり直そう.



- 最後に, bar 以下をすべて（一度に）消去しよう.

【C言語】

- 第１回の実習で作成した “Hello World” (ex01-1.c) を以下のようにコンパイルする.（そのファイル名を ex01-1.c と仮定している）

```
% gcc␣ex01-1.c␣-o␣ex01-1 Return
```

-o␣ex01-1 をつけたことにより, 生成される実行ファイルの名前が ex01-1 となる.

- 次に, おなじ ex01-1.c を

```
% gcc␣ex01-1.c␣-o␣test Return
```

としてコンパイルしてみよう. さらに,

```
% test Return
```

としたときと

```
% ./test Return
```

としたときの実行結果には何か違いが生じるかを調べ, その理由を考えてみよう. (資料 section 4.4.5.1) なお, 資料中で推奨しているコンパイル方法は,

```
% gcc␣-ansi␣-pedantic-errors␣-Wall␣ex01-1.c␣-o␣test Return
```

である. しかし, 前回および今回のプログラムをこのオプションのもとでコンパイルすると, いくつかの警告が発生する.

以後は以下のようにコンパイルするのが望ましい.

```
% gcc␣-ansi␣-Wall␣ex01-1.c␣-o␣ex01-1 Return
```

- 以下のようなプログラムを書き, printf 関数の意味をもう少し詳しく調べてみよう.（今日配布した資料や参考書をもとに, 来週までに %d とかの意味を調べてこよう.）(資料 section 6.7.2)

電子メールで「今日の講義の感想や意見」を送ってください.

★ ex02-1.c の内容

```
/* １から１０までの和を計算する */
/* $Id: ex02-1.c,v 1.3 2004-04-12 20:20:32+09 naito Exp $ */
/* ex01-1.c     */

#include <stdio.h>

int main(int argc, char **argv)
{
  int   sum ;
  int   i ;

  sum = 0 ;
  for(i=1;i<=10;i++) {
    sum = sum + i ;
  }
  printf("sum = %d\n", sum) ;
  return 0 ;
}
```

★ ex02-2.c の内容

```
/* y = 2*x + 1 の値を順に計算する */
/* $Id: ex02-2.c,v 1.4 2004-04-12 19:16:13+09 naito Exp $ */
/* ex02-2.c     */

#include <stdio.h>

int main(int argc, char **argv)
{
  int   i, j ;

  printf("f(x) = 2 x + 1\n") ;
  for(i=0;i<20;i+=2) {
    j = 2*i + 1 ;
    printf("f(%d) = %d\n", i, j) ;
  }
  return 0 ;
}
```


★ ex02-3.c の内容

```
/* 偶奇を判定する */
/* $Id: ex02-3.c,v 1.2 2004-04-12 17:06:08+09 naito Exp $ */
/* ex02-3.c     */

#include <stdio.h>

int main(int argc, char **argv)
{
  int   i ;

  for(i=0;i<20;i++) {
    if ((i%2) == 0) {
      printf("%d is even\n", i) ;
    }
    else {
      printf("%d is odd\n", i) ;
    }
  }
  return 0 ;
}
```

★ ex02-4.c の内容

```
/* y = x/2*2 の値を順に計算する */
/* $Id: ex02-4.c,v 1.5 2004-04-12 20:23:52+09 naito Exp $ */
/* ex02-4.c     */

#include <stdio.h>

int main(int argc, char **argv)
{
  int   i, j, k ;

  printf("f(x) = x/2*2\n") ;
  for(i=0;i<20;i++) {
    j = i/2*2 ;
    k = i*2/2 ;
    printf("f(%d) = %d, %d\n", i, j, k) ;
  }
  return 0 ;
}
```

【課題】

★ `exercise-02-1` プログラム `ex02-1.c` をもとにして, 次のプログラムを書きなさい.

- $n = 20$ として, $k = 1, \ldots, n$ に対して「1から k までの和」を表示するプログラム.

★ `exercise-02-2` プログラム `ex02-2.c` を「変数 `j`」を使わないように書き直しなさい.

★ `exercise-02-3` プログラム `ex02-3.c` をもとにして, 次のプログラムを書きなさい.

- $n = 20$ として, $k = 1, \ldots, n$ に対して, それぞれの k を 3 で割ったあまりを表示するプログラム.

★ `exercise-02-4` プログラム `ex02-4.c` は「不思議」な出力を出している. それはなぜかを考えよ.