基礎プログラミング演習 II 教材 (#1)

■ リハビリ演習 1（変数・入力・演算・出力）

春学期にやった C プログラミングの基礎的な部分を短時間でおさらいします。以下の課題を順にやってください。

□ はじめに

このクラスでは大量のプログラムを書きます。作ったプログラムを捨てることはありません (捨ててはいけません) から、最後には大量のファイルが手元に残るでしょう。分かりやすいファイル名をつけて、整理して開発することを試してください。組織的に作業することを訓練するのです。
コーディング（プログラムコードを書くこと）に限らず、全ての作業を含めてプログラミングのスキル（技能、つまり訓練を経て身につける能力）をできるだけ高めることがこのクラスの目的です。

□ 復習：演算・printf( )

まず簡単な演算を試します。データはハードコード（プログラム中に埋め込む形）で与えています。

プログラム中の以下の要素について注目してください。
・ コメント /* ……. */
・ #include 文
・ main( ) 関数と、{ } によるブロック
・ 各文はセミコロン ; で終わる
・ 最後は return 文で実行を終了
（終了は exit(0) でも良いが、その場合は #include <stdlib.h> が必要）

このプログラムは実行されると、main() 関数の { } ブロック中を上から下に向かって一文ずつ処理します。

```
/*
    calc01.c
    演算を試す
    数値はハードコードで与える
*/
#include <stdio.h>
int main()
{
    int num1, num2, answer;
    num1=100;
    num2=200;
    answer = num1 + num2;
    printf("answer: %d\n", answer);
    return 0;
}
```

ブロック中の以下の要素について注目してください。
・ int による変数宣言（ここでは三つの変数が宣言されています）
・ 代入式による変数への値の代入
・ 演算式の記述
・ printf( ) 関数による値の出力（表示）

このプログラムを作成し、保存し、コンパイルし、実行して、結果を確認してください。

```
$ gcc -o calc01 calc01.c
$ ./calc01
answer: 300
$
```

□ scanf( ) による実行時の入力

データをハードコードするのではなく実行時に与えられるように機能追加します。
右例のようにプログラムを修正してください。
scanf( ) 関数による入力だけでなく、scanf( ) 関数が正しく動作している事を確認するために入力された値をそのまま出力する printf( ) 文も加えています。
（例は main( ) 関数内の必要な記述だけを抜き出しています。return 文も含まれていません。）

```
int num1, num2, answer;
scanf("%d", &num1);
scanf("%d", &num2);
printf("num1: %d, num2: %d\n", num1, num2);
answer = num1 + num2;
printf("answer: %d\n", answer);
```

以下の要素について注目してください。
・ scanf( ) 関数による値の入力（変数の指定には & 記号をつける）
・ 修正した箇所は常に正しく機能しているか確かめながら作業を進める
・

このプログラムを作成し、保存し、コンパイルし、実行して動作を確認してください。

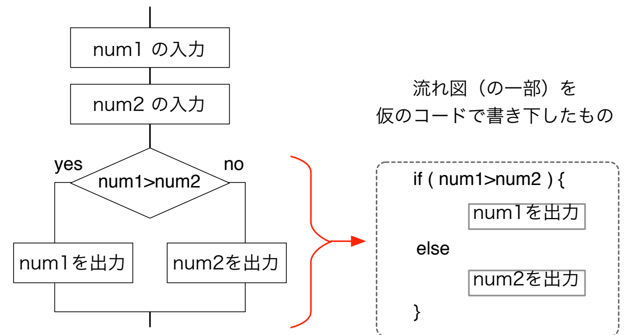
```
$ ./calc02
10
20
num1: 10, num2: 20
answer: 30
$ ./calc02
20
30
num1: 20, num2: 30
answer: 50
$
```

注：プログラムを作るときはそのまま直さず、ファイル名を変えて作るようにして、古いプログラムをなるべく残すほうが良いです。

```
$ cp calc01.c calc02.c
$
```

■ リハビリ演習 2（流れ図・条件分岐・場合分け・テスト）

□ 二数のうちの大きい方を出す



入力した二数のうち、より大きな方を出力するプログラムについて考えます。右に流れ図と、それを仮の（不充分な）コードで書き下したものを示します。

流れ図等を描く意味：
　これによって必要な変数（登場人物）とその動き（振る舞い）がハッキリし、正しく動作するかどうか調べることができます。（紙の上でシミュレーションするのです）

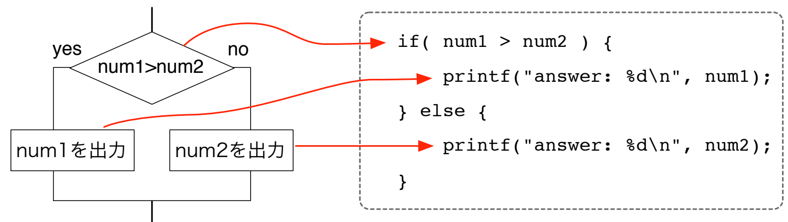
いきなりコンピュータに向かわない：
　プログラムを書くときは、タイプしながら考えることをせず、流れ図や仮のコードの形で処理手順を明確に書いてからコンピュータに向かうように。
　いきなりコンピュータに向かって手を動かすと、あなたの脳はかなりの部分を機器の操作に持って行かれてしまい、あなたの思考能力は半分以下に落ちてしまいます。

上のことに注意して、実際に動作するプログラムを作って下さい。

ただし、少しずつ書いて、少しずつ動作確認しながら進めるように。
1. まず二つの変数に値を入れて、そのまま出力するだけのプログラムを作り、動作を確認する
2. 次に変数の内容によって分岐する処理（先述）を加え、再び実行して動作を確認する

最初から完全なプログラムを書こうとしてはいけません。このくらい簡単なプログラムは一発で書けるでしょうが、すぐにそれができない規模に突入します。いま作法として身につけて下さい。

□ プログラムの構造を意識する



右図に示すように、多くの場合プログラムは立体的な構造を持っています。プログラマはこれを何とかしてテキスト（文字の列）の型式に押し込めて記述しなければなりません。
この種の「構造」を表現する手段として { } による「ブロック」や「インデント（字下げ）」が用いられます。

- ブロックは文法として C 言語に組み込まれています。その扱いを間違えるとコンパイルエラーとなったり、実行時にプログラマの意図と異なる処理が実行されてしまいます。
- インデントは文法では無くプログラマの慣習・作法です。作法ですから正誤はなく、善し悪しがあるだけで、流儀すらあります。（つまり人によって「より良い」と考えるインデントの付け方が微妙に異なる）

インデントをどのようにしてもエラーにはなりませんし、実行結果も変わりません。しかしその分、プログラマが気を配って構造が分かりやすく見えるように書かなければなりません。

注意：どこの世界でも、一人で仕事をすることはほとんど無く、実際にはチームで作業するのです。
　　　ソフトウェア開発は特にその傾向が強いことを意識せねばなりません。

□ プログラムをテストする

```
$ ./larger01
10
20
answer: 20
$ ./larger01
20
10
answer: 20
$ ./larger01
10 10
answer: 10
$
```

最後の動作確認での、入力値の組み合わせ（右の実行例）に注目してください。
まず 10, 20 と 20, 10 の二通りを入れることで、用意されている条件分岐の両側を通過して、それぞれ正しく機能することを確認しています。

次に 10, 10 を試しているのはそれが「特殊な条件」だからです。このプログラムでは「num1 > num2」あるいは「それ以外」だけで判定しており「num1 == num2」の場合どのようにするか書いていません。実際には同値であった場合は「偽（条件不成立）」と判定され、else 側に進んで num2 が出力されますが、num1 と num2 は同値なので、どちらが出力されてもそれで良いのです。

動作確認は「あり得る状況をすべて想定して網羅的に試験する」ことを覚えてください。

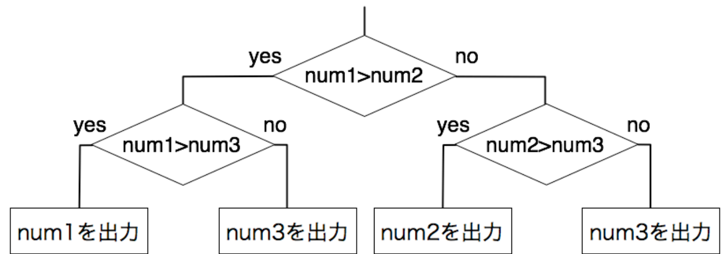
■ リハビリ演習 3（場合分け・ロジックの書き下し・テスト）

□ 三数のうち最も大きい数を出す

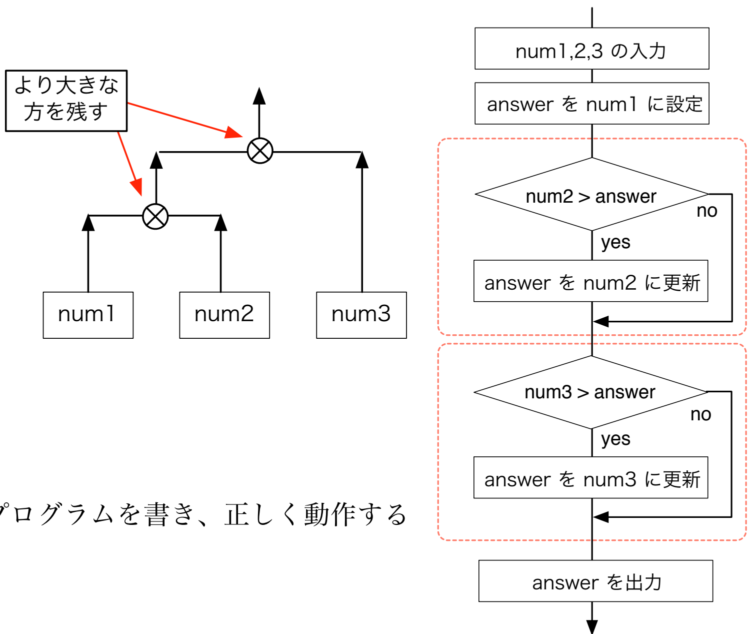
入力した三数のなかで最も大きな数を出力するプログラムを作成します。考え方は何通りかあると思いますが、代表的なものを挙げてみます。

1. 一つずつ較べ、あり得る大小関係に対して網羅的に反応する処理を書く。（右の流れ図）

   （右の図に示したものとは異なるロジックで網羅的な反応を列挙することも可能です。自分に分かりやすい方法で進めれば良い。）



2. トーナメントのように二数を較べて「より大きな数」を残す作業を二回行う。最後に残ったものが「最も大きな数」であるはず。
   右にトーナメント的な図と、それを実現する手順を流れ図で示します。「暫定的な最大値」を示す変数 answer を用意し、はじめ num1 を入れて順に num2, num3 と比較し、必要なら暫定一位を置き換えます。最後に残った値が「真の最大値」です。（納得できますか？）



1. 2. あるいはあなたが独自に思いついた方法で、プログラムを書き、正しく動作することを確かめて下さい。

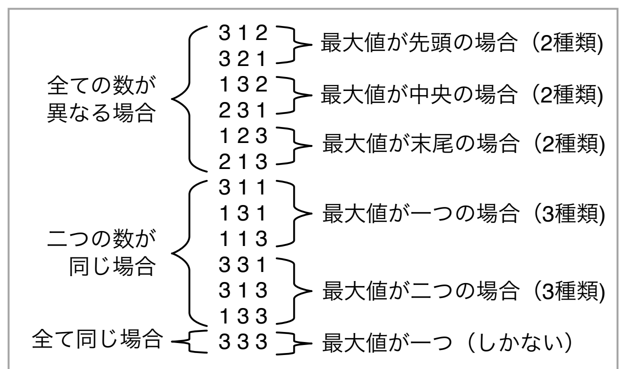
ただし、以下の点に注意して作業すること。
1. いきなりコンピュータに向かわない（充分に流れ図などを書く）
2. 全てのコードを一発で書いたりしない（少しずつ書いて動作確認）
3. 構造を意識した、より良いコードを書く（インデントなど）
4. テストしながら開発する
5. 出来上がった（と思った）ら、全ての場合について（再び）網羅的にテストする

□ 動作確認

まず「あり得る全ての状況（入力データの列）を列挙する」事から始めます。

右に「常に最大の数が 3 になるような三つの入力を網羅的に列挙」してみます。

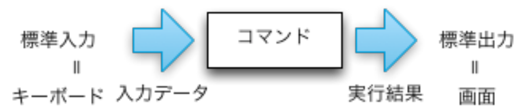
これは「1, 2, 3 のいずれかの数字が書かれたカードを三回引いた場合、全部で何通りのパターンがあり得るか」という問題と似ています。（数学の「組み合わせ」問題）
そこから「常に最大の数が 3 となる場合に絞る」組み合わせだけを残したものです。



これらの組み合わせ全てについて正しい結果（つまり３）を答えることを確認してください。

■ 処理の自動化

基礎プロⅠの「UNIX コマンドの応用（五週目の単元）」で「標準入出力」「リダイレクション」「パイプライン」について学んでいます。右の図に見覚えがありますか？
身についていますか？



この機能を使ってテスト処理を自動化します。
（コンピュータ屋は自動化できる処理はすべて自動化します。オペレーションミスは天敵だと心得て下さい。）

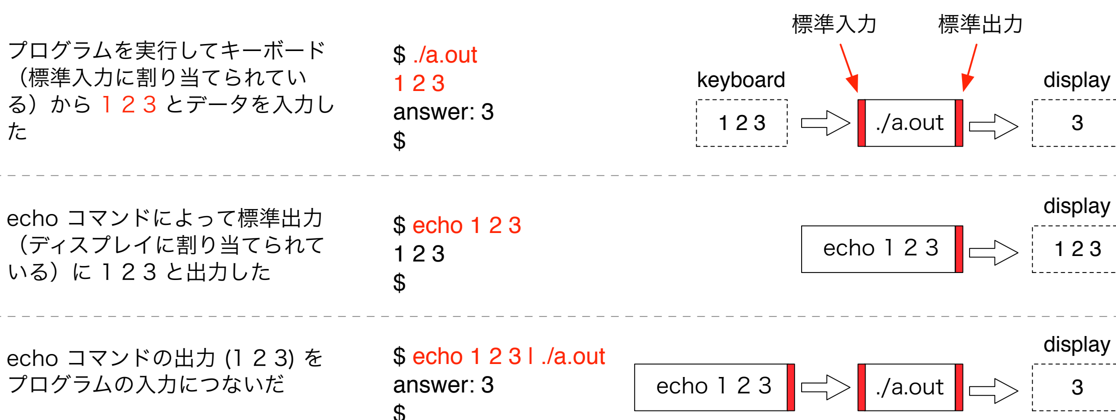
先に示した「全ての結果が 3 となるような入力データを与えてプログラムを実行する」ことを自動的に（オペレーション一発で）処理する方法を以下に示します。

□ パイプによってプログラムのデータ入力を与える

実行するとキーボードから入力データを得ようとするプログラムに対して、右のようにすることで、コマンドの出力としてデータ入力を与えることができます。

```
$ echo 1 2 3 | ./a.out
```

以下にその構造を図示します。



□ 一度の操作で実行する

つまり入力データとして前ページに示した数字のパターンを与えるためには、右のように書き連ねれば良いのです。最も簡単な実行方法は、この記述をテキストエディタに書き、コマンドシェルに対してコピー＆ペーストすることです。

```
echo 3 1 2 | ./a.out
echo 3 2 1 | ./a.out
（中略）
echo 1 3 3 | ./a.out
echo 3 3 3 | ./a.out
```

ただ、やってみると結果表示が今ひとつ分かりにいでしょう。そこでエディタに書いたコマンド記述の列を（ファイル名は例えば max.sh）保存し、右のように実行して下さい。するとすべての結果が正しく 3 になった（つまりプログラムは正しかった）ことがハッキリと確認できるでしょう。

```
$ bash < max.sh
answer: 3
answer: 3
answer: 3
（以下略）
answer: 3
$
```

コマンドシェルに対する不等号記号「 < 」はリダイレクションです。機能的には記号に続くファイルの中身をプログラム（ここでは bash つまりコマンドシェルそのもの）の標準入力に与えます。コマンドシェルは標準入力の内容をコマンド列として解釈・実行するので、これで望みのコマンド列が一回の操作で自動的に実行されます。

なお bash は引き数としてファイル名を与えると、そのファイル名の内容をコマンド列として解釈・実行する機能があります。つまり右のようにもできます。

```
$ bash max.sh
```

このようなシェル用のコマンド列が書かれたファイルをシェルスクリプトと呼びます。
重要なことは繰り返して間違いなく同じ内容でテストが出来ることです。
手で毎回作業すれば必ず操作ミスが起きます。常に自動化するのです。手作業は敵です。

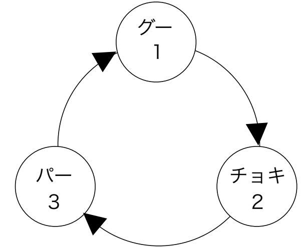
今回は 13 行しか出力が無いので目視確認ですが、私なら $ bash max.sh | uniq –c とします。自動化大事！

■ 課題：じゃんけん判定

以下のような「じゃんけん」プログラムを作成してください。
- 二つの数を入力として受けて勝敗判定をする
- 一つ目は「自分の手」、二つ目は「相手の手」
- 入力は 1, 2, 3 のいずれかで、それぞれグー、チョキ、パーに対応する
- 勝ち負けは “win”, “lose”, “even” で表示する（右例）

```
$ ./a.out
1 2
win
$
```

処理の方法はいろいろあると思います。総当たり的に if 文を並べる方法もあるでしょう。論理演算子でスッキリ書く事もできます。
右図に示すように循環している数列と考えて、その中での数値（1〜3）の性質を利用すると、もっとスマートに書けます。(大小関係をみると「おおよそ」勝敗が分かります。例外的な場合について、どう対処しますか？)



異なる手法（アルゴリズム）に基づく、二種類のじゃんけんプログラムを作成し、完成したものを Moodle に提出してください。

ただし、以下の点に注意して作業すること。
1. いきなりコンピュータに向かわない（充分に流れ図などを書く）
2. 全てのコードを一発で書いたりしない（少しずつ書いて動作確認）
3. 構造を意識した、より良いコードを書く（インデントなど）
4. テストしながら開発する

□ プログラムをテストする（完成を待たずテストしながら開発する）

必ず「あり得る全ての場合について」テストしてください。

そのために「あり得る全ての状況を列挙する」事から始めます。(右図)
（9 通りで全ての場合を網羅していることは数学の「場合の数（順列・組み合わせ etc)」で学んだことから分かるでしょう）

```
あいこの場合
  1-1, 2-2, 3-3
勝ちの場合
  1-2, 2-3, 3-1
負けの場合
  1-3, 2-1, 3-2
```

これを「片っ端から手入力して動作確認」することも可能ですが、もちろんシェルスクリプトを使って自動化してください。

プログラムは標準入出力を利用しているので、右例のように echo コマンドとパイプを用いて実行させることができます。
（分からない人は基礎プロ Ｉ 「UNIX コマンドの応用」を要復習）

```
$ echo 1 2 | ./a.out
win
$
```

右のようなシェルスクリプトを作成し、これを実行して結果が正しく右のようになることを確認してください。
（確認しながら開発をしてください）

```
$ cat janken.sh
# あいこの場合
echo 1 1 | ./a.out
echo 2 2 | ./a.out
echo 3 3 | ./a.out
# 勝ちの場合
echo 1 2 | ./a.out
echo 2 3 | ./a.out
echo 3 1 | ./a.out
# 負けの場合
echo 1 3 | ./a.out
echo 2 1 | ./a.out
echo 3 2 | ./a.out
$
```

```
$ bash janken.sh
even
even
even
win
win
win
lose
lose
lose
$
```