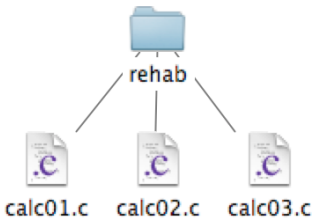
基礎プログラミング演習 II

■ リハビリ演習

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。

□ 00：準備



これから作業するためのディレクトリを一つ作り、そこにこれから作るプログラムは保存するように。右図に示すような格好で整理するつもりでやってください。
（ディレクトリ、ファイルの名前の付け方は任せます。思い浮かばない人は例の通りにするのも良いでしょう。）

□ 01：最初の演算・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。

```
/*
    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;
}
```

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。

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
$
```

□ 02：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。

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。

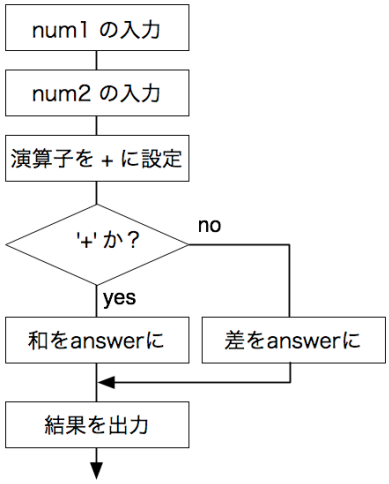
```
$ cp calc01.c calc02.c
$
```

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

□ 03：演算の種類を増やす

演算の種類を指定できるように修正します。
まず演算子はハードコードして与え、その値によって条件分岐が正しく機能することを確認します。

```
int num1, num2, answer;
char operator;
scanf("%d", &num1);
scanf("%d", &num2);
operator = '+';
// printf("num1: %d, num2: %d¥n", num1, num2);
if( operator == '+' ) {
    answer = num1 + num2;
} else {
    answer = num1 - num2;
}
printf("%d %c %d = %d¥n", num1, operator, num2, answer);
```



以下の点に注目してください。
・ char 型による文字変数の宣言
・ if 文による処理の分岐と else 句の意味、および比較演算子の記法
・ printf( ) の %c による文字出力

まずこのままで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。
次に operator に対する代入式で、'+' を代入せず、'−' を代入するようにプログラムを修正し、その結果、正しく差が計算して表示されることを確認して下さい。

□ 04：演算子を入力する

演算子をハードコードではなく、scanf( ) による入力で与えられるように修正します。
ただし（詳しく説明しませんが scanf( ) の性質による問題を避けるために）、以下のように今まで二行に分けていた scanf( ) の入力処理を一行にまとめるようにしてください。

```
scanf("%d %c %d", &num1, &operator, &num2);
```

実行時の入力も 10 + 20 といった具合に一行で入力すると良いです。

```
int num1, num2, answer;
char operator;
scanf("%d %c %d", &num1, &operator, &num2);
printf("num1: %d, op: %c, num2: %d¥n", num1, operator, num2);
if( operator == '+' ) {
    answer = num1 + num2;
} else {
    answer = num1 - num2;
}
printf("%d %c %d = %d¥n", num1, operator, num2, answer);
```

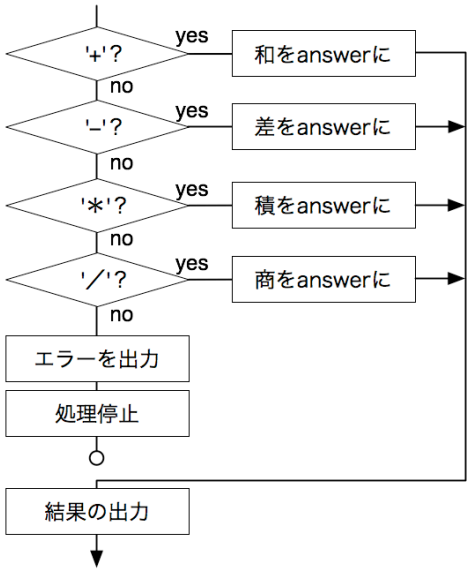
以下の点に注目してください。
・ scanf( ) による文字入力の %c
・ 入力した結果を確認するための printf( ) 文

実際に動作させて、右の例のように + あるいは − を指定して適切な処理が行われることを確認してください。
このとき様々な数値の組み合わせで問題無く動作することを確認してください。たとえば結果が負になるような場合でも正しく動作するか、入力に負の数を与えるとどうなるか、といったことです。

```
$ ./calc04
10 + 20
num1: 10, op: +, num2: 20
10 + 20 = 30
$ ./calc04
20 - 10
num1: 20, op: -, num2: 10
20 - 10 = 10
$
```

□ 05：四則演算への拡張

次に演算子の種類を増やして四則演算に対応するよう修正します。また存在しない演算子を指定した場合はエラーを表示してプログラムを終了します。
なお今回コードは今回の内容に関係する部分だけを抜き出していますが、それ以外にも exit( ) のために冒頭に #include <stdlib.h> の記述が必要です。




```
if( operator == '+' ) {
    answer = num1 + num2;
} else if( operator == '-' ) {
    answer = num1 - num2;
} else if( operator == '*' ) {
    answer = num1 * num2;
} else if( operator == '/' ) {
    answer = num1 / num2;
} else {
    printf("operator error (%c)¥n", operator);
    exit(1);
}
printf("%d %c %d = %d¥n", num1, operator, num2, answer);
```

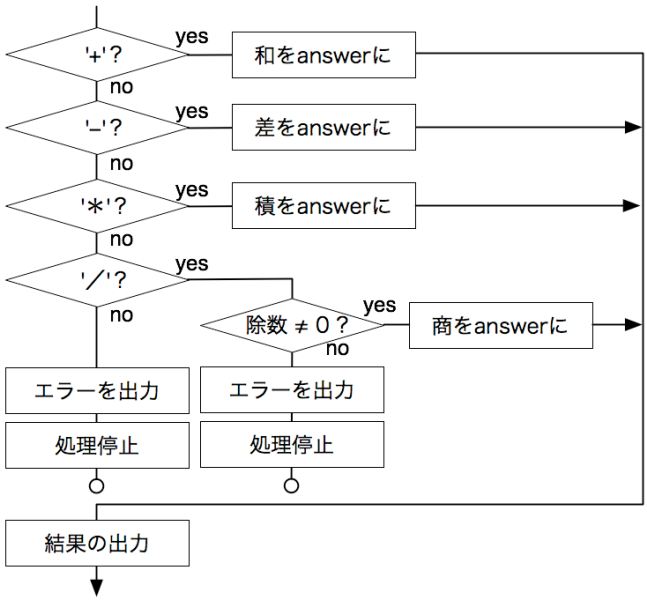
以下の点に注目してください。
・else if の記法、その意味
・ いずれにも該当しない演算子の場合エラーとし、exit( ) を用いてプログラム途中で処理停止する
・ 途中終了しなかった場合は answer を結果として出力するところまで進む

実際に動作させて、各種の演算子を指定して適切な処理が行われること、また規定外の演算子でエラーが表示され、そこで停止することを確認してください。

```
$ ./calc05
100 / 5
100 / 5 = 20
$ ./calc05
10 = 20
operator error (=)
$
```

□ 06：ゼロ除算の対応

慎重な人は 10 / 0 などを入力した場合、ゼロ除算となって C 言語の処理系がエラーを出し、プログラム自体が止まってしまった事に気がついたでしょう。そこでこれを回避するよう修正してください。
つまり「もしゼロで割ろうとしていた場合はエラーメッセージを出す」ように機能追加するのです。
以下に処理の流れと、実行結果のサンプルを示しておきます。今回はコード例をつけません。自分で考えて組み込んでみてください。




```
$ ./calc06
10 * 20
10 * 20 = 200
$ ./calc06
10 / 2
10 / 2 = 5
$ ./calc06
10 / 0
10 / 0 = (divide by zero)
$
```