3　計算をさせてみよう ～四則演算～(続き)

課題 1 (4/12 出題). 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。)

解答例 1.1. 表示の形式などの細かい所は不問。各自で工夫されたし。

```
/*   int3.c   2010-04-12   */

#include <stdio.h>

int main(int argc, char** argv)
{
  int x, y, z;

  x = 2010;
  y = 4;
  z = 12;

  printf("sum: %d + %d + %d = %d\n", x, y, z, x+y+z);
  printf("average: %d / 3 = %d\n", x+y+z, (x+y+z)/3);
}
```

3 で割る部分に関しては、わざわざ変数を用いることもなかろう。但し、もし値 3 が変わることを想定するなら、ここで変数を一つ使い、

```
 int n;
  ...
 n = 3;
  ...
```

などとするのも意味があろう。状況の見極めによる。

`printf()` の引数には、`x+y+z` のような式が来ても良い。単独の変数も式の一つである。その式を計算 (評価) した値が処理される。

解答例 1.2. 上記の解答例では x+y+z を何回も計算していて二度手間である。そこでこの結果を覚えておく為に、もう一つ変数を使ってみる。

```
/*   int3.c   2010-04-12   */

#include <stdio.h>

int main(int argc, char** argv)
{
  int x, y, z, s;

  x = 2010;
  y = 4;
  z = 12;

  s = x + y + z;

  printf("sum: %d + %d + %d = %d\n", x, y, z, s);
  printf("average: %d / 3 = %d\n", s, s/3);
}
```

考察 . このように中間結果を覚えておく為の変数を用いる利点・不利点を挙げよ。用いるのが効果的なのはどんな時か。

今の場合は x+y+z の計算が大した手間ではないので大差ないが、手間 (時間) の掛かる計算であれば計算結果を覚えておくのは効果が大きい。但し、現実に変数 (メモリ領域) を一つ余計に使っているし、代入式を一つ余計に書いてもいる。変数を無闇に増やすと、何の為の変数だったか把握し切れなくなって、読み難いプログラムになることもある。全体を見渡したバランス感覚が必要。

ここの変数名 s は、和 (sum) から取ってみた。変数名は一文字でなくても良いので、そのまま sum などとしても良かろう。判り易い変数名を使うことが読み易いプログラムにも繋がる。

4　変数・型・式・演算・代入

C 言語では、プログラム内で利用する変数は全て、その型と変数名とを予め宣言しておく必要がある。宣言によって、その変数の為の記憶領域がメモリ上に確保される。型は領域の大きさと値の扱い方を規定する。変数宣言文の書式は次の通り。

一般的な記法として、[...] は省略可能な部分を表す。

型名　　変数名 [=初期値] , . . . , 変数名 [=初期値] ;

良く使う C 言語の基本的な型は次の通り。

他に基本的な型としては、
short : (短い) 整数型 (= short int)
long : (長い) 整数型 (= long int)
float : 単精度浮動小数点型 (実数型)
などがある。

- int : 整数型
- double : 倍精度浮動小数点型 (実数型)
- char : 文字型

演算した結果 (式の値) もそれぞれ型を持つ。C 言語では重要な概念である。

これらは全て「符号つき」として扱われるが、型の前に修飾子 unsigned を付けると「符号なし」(正値のみ) となる。
(例. unsigned int : 符号なし整数型)
符号付きと符号なしとでは表せる数値の範囲が異なる。

4–1　四則演算

(原則として同じ型同士の) 式の間で四則演算 (+,-,*,/) が出来る。式には単独の数値・変数なども含む。
主に整数型の四則演算に関する注意は下記の通り:

- 整数 (型) 同士では、剰余演算子 % もあり、他の演算子と同様に 8%5+4 $(= 3 + 4 = 7)$ のように使える。
- 演算の優先順位は、一般的な数学の計算と同じで、*,/,% が優先。同順位なら左から。( ) で囲まれた部分が最優先。
- 整数 (型) 同士の割り算の結果は「整数」となり、小数点以下は切り捨て。
  次のような例に注意: 3/5*5 $(= 0 \times 5 = 0)$, 3*5/5 $(= 15/5 = 3)$.
- どの型においても、0 による割り算は実行時エラー。
- 「冪乗」の演算子はない。(^ は排他的論理和 (XOR) という別の意味を持った演算子)

従って、TEX の癖でうっかり
a^n
と書いても、コンパイル時にエラーと指摘してくれないので注意!!

4–2 代入

宣言した変数へ値を代入 (書き込み) するには = を用いる。代入文の書式は次の通り。

変数名 = 式;

また、変数宣言時に初期値を代入 (初期化) することも出来る (次の例参照)。

実習 4.1. 数学の等式としては一見妙な、次の例を実行してみよ。

```
/*   assign.c   2010-04-19   */

#include <stdio.h>

int main(int argc, char** argv)
{
  int x = 3, y = 5;

  printf("x = %d, y = %d\n", x, y);

  x = x + y;

  printf("x = %d, y = %d\n", x, y);
}
```

考察 4.1.1. 代入文 `x = x + y;` の効果を考えよ。

代入: assignment

数学の “等式” と見た目には同じだが、イメージとしては**「変数名 ← 式の値」**という感じ。
右辺の式を計算 (評価) した値が、左辺の変数名 (又は記憶領域を指し示す式) が指し示す記憶領域に書き込まれる。右辺の式は単独の定数値や変数でも良い。

右辺が単独の変数名の場合でも、単にその時に右辺の変数に入っている値が左辺の変数にコピーされるだけで、右辺の変数の値はそのまま保持されており、その後の両変数の値の変化は連動しない。

変数の型と代入する値の型 (右辺と左辺の型) は一致していなければならない。意図的に違う型の値を代入するときには、明示的な型変換 (キャスト) を行なう必要がある (後述)。

実習 4.2. 上の代入文は、加算代入演算子 (足し込み) += を用いて、

```
x += y;      /* x に y を足し込む */
```

と書いても良い。(しばしばこう書く方が見易い。) これを用いて書き換えよ。

「+=」は 2 文字で 1 つの「演算子」であり、間にスペースを入れてはならない。

その他同様に「-=」,「*=」,「/=」が (int なら「%=」も) 使える (複合代入演算子)。

C 言語らしい cool な演算子だ。

5 画面への表示

printf() 関数による文字列や式の値の表示 (出力) についてまとめておこう。

PRINT Formatted：書式付き出力

```
printf(書式指定文字列 [, 式, ...]);
```

- printf() を用いるには、プログラムの冒頭で stdio.h をインクルードする必要あり。
- 書式指定は " " で囲んだ文字列で行なえる。この中に通常の文字列を書くとそのまま (そういう書式として) 出力されるので、単に文字列 (文字列定数) を表示するには、そのまま " " 内に書けば良い。
- printf() は自動的には改行しないので、改行するにはその位置に改行を表す文字 '\n' を書く。(\ は、日本では環境 (画面のフォントなど) によっては ¥ と表示されるかも。内部的には文字コードは同じ。) '\n' は他の通常の文字と同様な一文字扱い。
- 式の値を表示するには、書式指定文字列中に % で始まる「変換指定記号」を用いる。
  (式の値を表示文字列に「変換」する方法の指定、という意味。)

まぁ大抵は printf() による出力を伴うだろう (そうでないと実行結果が確認できない) から、通常はいつも

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

を書くと思っていて良かろう。

'\' 付きで表す主な特殊文字

\t：水平タブ
\v：垂直タブ
\r：復帰 (行頭へ)
\n：改行 (次行頭へ)
\b：後退 (一文字戻る)
\\：\自身

例. 「整数 + 整数 = 整数 (改行)」の書式で、int 型の式 x, y, x+y の値を出力。

```
printf("%d + %d = %d\n", x, y, x+y);
```

- 値を表示したい式それぞれに対して、その式の型と表示したい形式に合った変換指定記号を左から順に対応させる。主な変換指定記号と対応する型は次の通り。
  - ⋆ `%d`：`int` (符号付き十進表示)
  - ⋆ `%u`：`unsigned int` (十進表示)
  - ⋆ `%x`：`unsigned int` (十六進表示)
  - ⋆ `%f`：`double` (固定小数点表示)
  - ⋆ `%e`：`double` (浮動小数点表示・指数表示)
  - ⋆ `%c`：`char`
  - ⋆ `%s`：`char*` (文字列)
  - ⋆ `%%`：`%` 自身 (式とは対応しない)

例: `123 (%d)`
$=$ `+123 (%+d)`
$=$ `7b (%x)`

例: `123.400 (%.3f)`
$=$ `1.2340e+2 (%.5e)`
$= 1.234 \times 10^2$

- `%` と変換指定記号との間に次のようなオプションを指定して、表示の書式を色々と変えることが出来る。

`%` [フラグ] [フィールド幅] [ピリオド] [精度] [修飾子] 変換指定記号

主なものは次の通り。

- ⋆ フラグ
  - ∗ `-`：左揃えで表示 (通常はフィールド幅内で右揃え)
  - ∗ `+`：常に符号つきで表示
  - ∗ スペース：先頭が符号でなければ (負でなければ) スペースを付ける
  - ∗ `0`：フィールド幅の空きに 0 を詰める
- ⋆ フィールド幅：(最小) 何桁で表示するかを指定
- ⋆ ピリオド：フィールド幅と精度を分離
- ⋆ 精度：小数点以下何桁を表示するかを指定
- ⋆ 修飾子 `l,L`：引数が long であることを表す (`%ld`：`long int` , `%Lf`：`long double` 等)

例. `double` 型の表示指定 `%8.3f` だと、全体で 8 桁、そのうち小数部が 3 桁、小数点自身で 1 桁使うので、整数部は残りの 4 桁。このように、整数部の表示桁数は (フィールド幅) − (精度) − 1 となる。

実習 5.1. double 型を使ってみる。

```
/*   arith2.c   2010-04-19   */

#include <stdio.h>

int main(int argc, char** argv)
{
    double x = 1.23, y = 7.89;

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

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

考察 5.1.1. 演算子 / の動作を int 型の時と比較せよ。

考察 5.1.2. プログラムの printf() の書式指定をいろいろと変えて、表示の変化を確かめてみよ。(幅指定・精度指定・指数表示など)

実数型 (浮動小数点数型) と整数型とが混じった演算では、自動的に “大きい” 型に変換されることになっている (暗黙の型変換)。具体的には、int 型と double 型との演算では、double 型に変換して計算され、結果の型は double 型となる。次の例を以前の例と比べよ: 3.0/5*5 $(= 0.6 \times 5 = 3)$.

明示的に型変換を行うには、変数の前に **(型)** を付ける (型キャスト)。

例 . int 型変数 i の値を double 型にキャスト: (double) i

ここでも「使い回しの心」で。

この例では変数宣言と同時に初期値を代入 (初期化) してみた。以前のように宣言後に改めて代入してもよろしい。

たまたま整数の値を double 型変数に代入する時は、

```
x = 1.0;
```

のようにするか、型キャストを用いて

```
x = (double)1;
```

とする。

6 キーボードからの入力

SCAN Formatted：書式付き入力

`scanf()` 関数を用いると、キーボードやデータファイルから数値や文字を入力し、変数に代入することが出来る。入力した値を代入する為の変数は予め適切な型で宣言しておく。

`scanf()` の書式は次の通り (`printf()` とほぼ対応している)。

```
scanf(書式指定文字列 , &変数名 [, &変数名, ...]);
```

`scanf()` を用いるには、`printf()` 同様、プログラムの冒頭で `stdio.h` をインクルードする必要があるが、これは通常いつも書くということで。

- `scanf()` を用いて数値を入力するときには、**変数名の前に `&` を付ける**。
  (この理由はポインタを習う所で判明する。)
- 書式指定文字列内の変換指定記号と各変数とが左から順に対応する。その変数の型に合った変換指定記号を用いる。変数の型と変換との対応の主なものは次の通り。
  - ⋆ `%d` : `int`
  - ⋆ `%ld` : `long`
  - ⋆ `%lf : double` ⟸ `printf()` **と異なるので注意！**
  - ⋆ `%c` : `char`
  - ⋆ `%s` : `char*` (文字列)

実は、`scanf()` は `printf()` などで生成された予め書式の決まった文字列を受け取るのに適していて、キーボードからの入力のように「何か来るか判らない」ような場合には適していない。(セキュリティホールを生ずる可能性もある。) そのような時には安全の為にちょっと一段階置いた方法を取るのが良いのだが、まぁ初めは安易に `scanf()` で練習しておこう。より安全な方法は追い追い。

例 . 「整数 (空白) 整数」の書式で入力を受け、`int` 型変数 `x`, `y` に代入。

```
scanf("%d %d", &x, &y);
```

改行文字を待ってから 1 行分の入力が処理系に送られるので、実際の入力は
「整数 (空白) 整数 Enter」
となる。

課題 2 (✗切 4/25(日)). 2 つの実数と 1 つの整数を入力し、2 つの実数の和を整数で割った値を計算し、その結果を表示するプログラムを作成せよ。入力は `scanf()` で受け取り、表示の書式は適当に定めよ。