ソフトゼミＡ　第２回 解答と解説

■　練習問題

1. キーボードから 2 つの数を入力すると、その 2 つの数の積を表示するようなプログラムになるように、空所/* (1) */～/* (5) */を埋めよ。

   (1): %d　(2): &x　(3): &y　(4): %d　(5): x * y

```
#include <stdio.h>
int main( void ){
   int x, y;
   scanf( "%d", &x   );
   scanf( "%d", &y   );
   printf( "%d¥n", x * y );
   return 0;
}
```

   scanf の場合には変数名の前に「&」を忘れないようにしてください。忘れると、数値の入力後にプログラムが停止することがあります。なぜ&をつける必要があるかは第 7 回「ポインタ」で紹介予定です。

   Sample Input

   5 10

   Sample Output

   50

2. ある数字(正の実数)を入力すると、その長さを半径とする円の面積を表示するプログラムを書け。ただし、円周率は 3.1416 とする。

```
#include <stdio.h>
int main( void ){
   double r;
   double pi = 3.1416;
   scanf( "%lf", &r );
   printf( "%f¥n", r * r * pi );
   return 0;
}
```

   基本的に先ほどと同じですが、double 型の場合には scanf・・・%lf, printf・・・%f

と、違った書式指定文字を書かなければならないので注意してください。(printf に%lf と書いても一応動くかもしれないけれど、基本的に動作は**未定義**です。(C99 という規格でコンパイルしている場合は printf で%lf を使っても OK です。)

Sample Input

75

Sample Output

17671.500000

■ 追加練習問題

練習問題が終わった人向けに配布した「追加練習問題」(ゼミ∀)の解答です。

1. 2つの整数を読みこむ。先に入力された方をa,後に入力された方をbとしたとき、a + b, a - b, a * b, a / b, a % b(a / bの余り)の計算結果を出力せよ。

```
#include <stdio.h>

int main( void ){
   int a, b;
   scanf( "%d%d", &a, &b );
   printf( "a + b = %d¥n", a + b );
   printf( "a - b = %d¥n", a - b );
   printf( "a * b = %d¥n", a * b );
   printf( "a / b = %d¥n", a / b );
   printf( "a %% b = %d¥n", a % b );

   return 0;
}
```

printf の中で「%」を出力したいときには「%%」と書いてください。

Sample Input

935 75

Sample Output

a + b = 1010

a - b = 860

a * b = 70125

a / b = 12

a % b = 35

2. 2147483647 という値が入っている int 型整数に 1 を足すと何が起こるか。実際に実行してみよ。

```
#include <stdio.h>
int main( void ){
   int lim = 2147483647;
   printf( "%d¥n", lim );
   lim++;
   printf( "%d¥n", lim );
   return 0;
}
```

Output

2147483647

-2147483648

2147483647 に 1 を足すとマイナス符号がついて「-」2147483648 となってしまいました。なぜでしょう？

以前、変数は数字が入る箱というお話をしました。その箱には、実際には 0 と 1 のら列(2 進数にちょいとした加工を施したもの)で数字が入っていると考えてください。その 0 と 1 のら列と 10 進数の整数には以下の関係があります。(int 型の場合)

0 と 1 のら列(2 進数+α)	10 進数
0111 1111 1111 1111 1111 1111 1111 1111	2147483647
0111 1111 1111 1111 1111 1111 1111 1110	2147483646
……	……
0000 0000 0000 0000 0000 0000 0000 0011	3
0000 0000 0000 0000 0000 0000 0000 0010	2
0000 0000 0000 0000 0000 0000 0000 0001	1
0000 0000 0000 0000 0000 0000 0000 0000	0
1111 1111 1111 1111 1111 1111 1111 1111	-1
1111 1111 1111 1111 1111 1111 1111 1110	-2
1111 1111 1111 1111 1111 1111 1111 1101	-3
……	……
1000 0000 0000 0000 0000 0000 0000 0010	-2147483646
1000 0000 0000 0000 0000 0000 0000 0001	-2147483647
1000 0000 0000 0000 0000 0000 0000 0000	-2147483648

なぜこんな割り当てになっているかというのは、「コンピュータが足し算・引き算を簡単に実行できるようにするため」「すべてのビットをより多くの数字に割り当てるようにしたため」の結果なのですが、この結果、2 進数の「0111 1111 1111 1111 1111 1111 1111 1111」に 1 を足すと、「1000 0000 0000 0000 0000 0000 0000 0000」となり、表の 1 番下にある通りこれは「-2147483648」を表します。同様に「-2147483648」から 1 を引くと「2147483647」になります。このような現象を「オーバーフロー」といいます。

要は、何が言いたかったかと言うと、変数の箱にも大きさ(int 型の場合、0 または 1 が 32 個=32bit=4 バイト)があって、その範囲で扱える数値のみが使えるということです。およそ 21 億以上の数値を扱う場合にはおとなしく long 型など、さらに広い範囲が扱えるデータ型を使いましょう。

3. 135 億 + 12 兆 5000 億を計算せよ。ただし、long 型の変数を使ってはならない。

```
#include <stdio.h>

int main( void ){
  int a = 135;
  int b = 125000;
  printf( "%d00000000 + %d000000000 = %d00000000¥n", a, b, a + b );
  return 0;
}
```

先ほど述べたように、int 型では約 21 億以上の数値を扱えません。しかし、足される数も足す数も、1000 万の位(くらい)から下は全て 0 なので、135 + 125000 をやっておいて、式ごと 1 億倍(下に 0 を 8 個つける)してしまえば OK です。

Output

13500000000 + 125000000000000 = 12513500000000

4. 314159265 + 3589793238 を計算せよ。ただし、long 型の変数を使ってはならない。

```
#include <stdio.h>

int main( void ){
  int a1 = 0;
  int a2 = 314159265;
```

(次ページへ続く)

```
  int b1 = 3;
  int b2 = 589793238;
  int c2 = a2 + b2;
  int c1 = a1 + b1 + c2 / 1000000000;
  c2 %= 1000000000;
  printf( "%d + %d%d = %d%d¥n", a2, b1, b2, c1, c2 );
  return 0;
}
```

何通りかやり方はありますが、ここでは一例を。足す数「3589793238」は約 21 億を超えていますので、int 型では表せません。というわけで、上一桁「3」とそれ以外「589793238」に分解し、それぞれを b1, b2 としています。 a1 と a2 も同様ですが、足される数の「314159265」は約 21 億に収まっているので、分解の必要はなく、a1 を 0 としています。

このように、数を 2 つのブロックに分けて考えます。a2, b2 は、1 の位～1 億の位、a1, b1 は 10 億より上の位を取り扱います。この後、a2 + b2 をしますが、1 億の位までで表される整数のうち、最も多いものは 999999999 で、それ同士を足して、999999999 + 999999999 としても 値は約 20 億 。 約 21 億である int 型の最大値より小さいので問題なく足し算ができます。

下 9 ケタ同士(a2, b2)を足して、答えの下 9 ケタ(c2)を出します。しかし、ただ足しただけだと、繰り上がりを含むので、a2 + b2 を 10 億で割って繰り上がりの数字を、上位ケタ(a1, b2)の足し算の結果に足します。最後に a2 + b2 を 10 億で割った余りを公式に答えの下 9 ケタとすることで、計算が完了します。

※a1 を printf で出力すると数字が 0 スタートになって気持ち悪いので、足される数については a2 のみが出力されています。

Output

314159265 + 3589793238 = 3903952503