入門講習会 第一回

1. Practice A が解けるようになるために
   今回は AtCoder の Practice Contest A を解くことを目標にする.
   問題を見てみよう.

A - はじめてのあっとこーだー(Welcome to AtCoder)

問題文

高橋君はデータの加工が行いたいです.

整数 a,b,c と,文字列 s が与えられます.

整数 $a+b+c$ と,文字列 s を並べて表示しなさい.

入力

入力は以下の形式で与えられる.

```
a
b c
s
```

- 1 行目は,整数 a ($1 \leqq a \leqq 1,000$) が与えられる.
- 2 行目は,整数 b,c ($1 \leqq b,c \leqq 1,000$) が与えられる.
- 3 行目は,文字列 s が与えられる.この文字列の長さは 1 文字以上 100 文字以下である.

出力

$a+b+c$ と s を空白区切りで 1 行に出力せよ.

また,出力の末尾には改行を入れること.

(https://practice.contest.atcoder.jp/tasks/practice_1 より引用)

これを解くためには,以下のことが必要になる.

- キーボードから整数 a,b,c と文字列 s を入力して,それらをどこかに保持しておく
- 整数 a,b,c,を足し算する

- 足し算の結果と文字列 s を表示する

これらを実現するために,以下のことについて順に説明する.

- 出力処理
- 演算
- 入力処理
- 配列

2. プログラムの基本と出力処理

まずは画面上に

```
Hello, World!
Good Bye, World.
```

を出力するプログラムを書いてみよう.

Sample1_1.c

```c
#include <stdio.h>

int main(void) {
    printf("Hello, World!¥n");
    printf("Good Bye, World.¥n");
    return 0;
}
```

このように,プログラムの内容を書いた文章を「ソースコード(もしくは単にコード)」という.ソースコードは人間が読めるように作られている.しかしコンピューターはソースコードが理解できない.そこで,コンピューターに理解できる言葉にそれを翻訳してやる必要がある.その翻訳のことを「コンパイル」と呼ぶ.その後「リンク」と呼ばれる工程を経て,ようやく実行ファイルが作られる.

コンパイル(とリンク)を行うには,コマンドプロンプトで以下のコマンドを実行する.

```
gcc ファイル名
```

ファイル名には自分の書いたソースコードのファイルの名前を入れる.さて,これを実行し,特にエラーも出ないようであれば,a.exe という名前で実行ファイルが出来上がっている.a.exe を次のコマンドで実行すれば,目的のプログラムが動く.

```
a.exe
```

さて,書いたプログラムについて少し見てみよう.

実際に「Hello, World!」という表示を行っている部分は,

```
printf("Hello, World!¥n")
```

の部分だけだ.

現時点では, 以下の点について理解していればよい.

- 処理はまず　int main(void)の後の中括弧の中から始まる.上から順に命令が実行される. 現時点では,printf の部分以外は定型文として覚えてもらってよい.
- 命令の後には必ずセミコロン 「;」 を入れる.
- printf(“〇〇〇”)と入力すれば, 〇〇〇を出力してくれる.
- printf 中にある『¥n』は, 改行を表す文字.
- printf を呼び出すためには, #include <stdio.h>を書かなければならない.
- return 0 でプログラムが終了する.

3. 数値や文字の表示

次のプログラムは,様々な数値や文字を画面に出力するプログラムである.

Sample1_2.c	

```c
#include <stdio.h>

int main(void) {

    printf("37 + 24 = %d¥n", 37 + 24);
    printf("%d, %d, and %d¥n", 64, 3 + 4, 128);
    printf("Pi is %f", 3.14);
    printf("%c, %s", 'A', "Hello");

    return 0;
}
```

実行結果は以下のようになる.

```
37 + 24 = 61
64, 7, 128
3.1400
A Hello
```

ソースコードについて,int main()内部から順にみていこう.

```
printf("37 + 24 = %d¥n", 37 + 24);
```

とは, 「『37 + 24 = %d』を表示しろ.ただし, %d は 37 + 24 を計算した値に読み替えて

くれ.」という意味.
%d というのは，「カンマで区切られた先にある値を整数で表示せよ」という意味.
%d を複数書けば,一つの printf 中に複数の値が表示できる.そのときは,複数のカンマで複数の値を区切る.値はその順番ごとに表示される.

```
printf("%d, %d, and %d¥n", 64, 3 + 4, 128);
```

表示できるのは整数だけではない.%f を用いれば実数が表示される.%c で文字,%s で文字列を表示することができる.

```
printf("Pi is %f", 3.14);
printf("%c, %s", 'A', "Hello");
```

まとめると,%〇と表示値の種類との対応は以下の表のようになる

%d	整数
%f	実数
%c	文字
%s	文字列

[補足]
文字は「一文字」を表し，これを表現するためにはシングルクォーテーションでくくる.
文字列は「文字の集まり」を表し，これを表現するためにはダブルクォーテーションでくくる.

4. 変数の宣言と代入

プログラミングの世界には「変数」という言葉がある.
数学で習った「変数」とは微妙に意味が違う.
プログラミングにおける変数とは，簡単に言うと
　何か値を入れておくための箱
である.箱の中に必要なものを入れておけば，それを後で計算に用いたり，書き換えたりできる.箱には(基本的には)何でも入れていいわけではない.入れるものに適した箱が必要である.
変数においても同じことが言える.変数に入れられる値の種類のことを「型」と呼ぶ.よく使う型として,整数型,実数型,文字型がある.
変数を作ることを,「変数の宣言」と呼ぶ.C 言語において,変数は以下の書式で宣言する.

```
型 変数名;
```

『型』の部分に入れるものについて,現時点では以下の 3 つを覚えておけばよい.

型	種類
int	整数

char	文字
double	実数

『変数名』については，「他の変数と名前が重複しない」，「予約語(C 言語が持っている特別な単語)でない」限りは，どんな名前でもよい.

また，

```
型 変数1，変数2，変数3;
```

のように,カンマで区切って変数を複数宣言できる.

変数に値を入れることを,「変数の代入」という.変数の代入は以下の書式で行う.

```
変数名 = 値;
```

後でも述べるが,ここで使っている=という記号は,数学における=ではないことに注意しよう.あくまで代入するための記号であると覚えておこう.

これらを踏まえて,サンプルを見てみよう.以下は,変数 val1,val2 を宣言した後,val1 には 10 を代入し,val2 には(val1+10)の計算結果を代入して,それらを出力するプログラムである.

Sample1_3.c

```c
#include <stdio.h>

int main(void) {
    int val1, val2;
    val1 = 10;
    val2 = val1 + 35;

    printf("val1 is %d¥n", val1);
    printf("val2 is %d¥n", val2);

    return 0;
}
```

実行すると以下のような結果となる.

```
val1 is 10
val2 is 45
```

順番に見ていこう.まず

```
int val1, val2;
```

によって,変数 val1 と val2 を宣言している.次に

```
val1 = 10;
val2 = val1 + 35;
```

で,val1 に 10 を,val2 に(val1+35)を代入している.val2 の代入の右辺については,

```
val1 + 35 = 10 + 35 = 45
```

と計算されて,最終的には val2 には 45 が代入される.そして

```
printf("val1 is %d¥n", val1);
printf("val2 is %d¥n", val2);
```

によって,val1 と val2 の値を出力する.

5. 変数の初期化

変数の初期化とは,変数の宣言と同時に値を代入することである.以下の書式で書く.

```
型 変数名 = 値;
```

前項のプログラムを変数の初期化を使って書き換えると,以下のようになる.

Sample1_4.c

```
#include <stdio.h>

int main(void) {
    int val1 = 10;
    int val2 = val1 + 35;

    printf("val1 is %d¥n", val1);
    printf("val2 is %d¥n", val2);

    return 0;
}
```

実行結果は前項と同じ.

ちなみに，もし初期化されていない変数を表示すると，どんな値が出るかわからない(この値を「不定値」と呼ぶ).何が入っているかわからないので，初期化または代入を行っていない変数を使ってはいけない.

6. 演算

普段数学などで扱っている四則演算などのことを,算術演算子と呼ぶ.

よく使う算術演算子について,以下に示す.

算術演算子	意味
+	足し算

-	引き算
*	掛け算
/	割り算
%	剰余算

一番最後の「剰余算」とは,余りを計算する演算子である.○%□の形で,「○を□で割った余り」を表す.

これらの 5 つの算術演算子を使ったプログラムを見てみよう.以下は,変数 a を 9,変数 b を 4 で初期化し,a と b に対して算術演算を行った結果を出力するプログラムである.

Sample1_5a.c

```c
#include <stdio.h>

int main(void) {
    int a = 9;
    int b = 4;

    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;
}
```

実行結果は以下のようになる.

```
a + b = 13
a - b = 5
a * b = 36
a / b = 2
a % b = 5
```

計算結果がどんな型になるかは,計算に用いた型による.整数同士の計算なら,計算結果は整数になる.だから上の実行結果について,a/b の結果が整数値として表示されている(後でも述べるが,整数同士の割り算は切り捨てになることに注意).

[補足]

%は printf においては特殊な文字となっているため,単に

```
printf("a % b = %d¥n", a % b);
```

と書いても%は表示できない.これを printf で出力したいときは,%%と書かなければならない.よって,

```
printf("a %% b = %d¥n", a % b);
```

と書けば,%が正しく出力される.

次に,複合代入演算子を紹介する.複合代入演算子とは,代入と演算を同時に行える演算子である.よく使う複合代入演算子について,書式を以下に示す.説明の都合上,複合代入演算子を☆,右辺と左辺をそれぞれ x,a として,『x☆a』の形式で書く.

複合代入演算子	意味
x += a	x に a 加える(x = x + a)
x -= a	x から a 引く(x = x – a)
x *= a	x を a 倍する(x = x * a)
x /= a	x を a で割る(x = x / a)
x %= a	x を a で割った余りを計算(x = x % a)

次に,「インクリメント/デクリメント演算子」を紹介する.これは,変数に 1 を加えたり,変数から 1 引いたりする演算子である.これらの演算子も,代入と計算を同時に行うという点については,複合代入演算子の仲間といえる.以下にそれらの書式を示す.

演算子	意味
x++	インクリメント.x に 1 を加える.
x--	デクリメント.x から 1 を引く.

上に挙げた演算子を使ったプログラムを見てみよう.以下は,変数 x を 0 で初期化した後,インクリメント,デクリメント,+=の計算を行い,そのたびごとに x の値を出力するプログラムである.

Sample1_5b.c

```
#include <stdio.h>

int main(void) {
    int x = 0;
    printf("x: %d¥n", x);

    x++;
    printf("x: %d¥n", x);
```

```
    x += 5;
    printf("x: %d¥n", x);

    x--;
    printf("x: %d¥n", x);

    return 0;
}
```

はじめ x は 0 であったが,

```
x++;
```

によって x は 1 となり,

```
x += 5;
```

によって x は 6 となり,

```
x--;
```

によって x は 5 となる.よって,実行結果は以下のようになる.

```
x: 0
x: 1
x: 6
x: 5
```

7. キーボードから入力

画面に文字や数値を出力するために printf を用いた.対して,キーボードから文字や数値を入力するためには,scanf を用いる.

scanf の使い方について,実際に例を見ながら理解しよう.以下は,キーボードから入力した値を変数 a, b に格納し,それらに対して四則演算した結果を出力するプログラムである.

Sample1_6.c	

```
#include <stdio.h>

int main(void) {
    int a, b;
```

```
    printf("input:");
    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);

    return 0;
}
```

実行結果は以下のようになる(*27 3* は入力例).

```
input: 27 3
a + b = 30
a - b = 24
a * b = 81
a / b = 9
```

キーボードから入力した整数値を変数 a, b に格納する処理は

```
scanf("%d %d", &a, &b);
```

で行われている.

一般に,キーボードから入力した整数値を変数に格納するためには,

```
scanf("%d", &変数名);
```

と書くことを覚えておこう.printf と似ているが, %d は入力した数を整数値として変数に格納することを表す.カンマ後に格納先の変数を指定する.ただし, その変数の先頭に &をつけなくてはならない.

2 つの整数をスペース区切りで入力するには, 次のように書く.

```
scanf("%d %d", &変数名, &変数名);
```

%d 以外の表記についても,以下の表に示す.printf のときとよく似ているが,実数の場合だけ表記が異なるので注意.

%d	整数
%lf (注意)	実数
%c	文字
%s	文字列(&はつけない.詳しくは後述)

8. これで Practice A は解ける？

いままで入出力,変数とそれらの計算について学んだが,実はまだ Practice A は解けない.

なぜなら,文字列の入力について学んでいないからだ.文字列と配列には密接な関係があるため,まずは配列について説明する.

9. 配列

いくつかの変数をまとめて扱いたいとき,配列が有用である.

配列とは,例えるなら

　複数の箱が一直線上に並んだもの

と理解しておけばよい.それぞれの箱には番号が書かれているので,その番号を指定して値を入れたり取り出したりする.

配列の宣言,及び初期化は以下の書式で行う.

[配列の宣言]

```
型 配列名[要素数];
```

[配列の初期化]

```
型 配列名[要素数] = {0 番目の要素, 1 番目の要素, … };
```

配列の宣言も初期化も,変数のそれとよく似ている.異なるのは,配列の宣言,初期化の際には要素数を指定する点である.ここで,配列の初期化の書式の「0 番目の要素」という記述に注目してほしい.配列の番号は 0 番目から始まる.だからもし要素数 4 の配列を宣言したら,配列の番号は 0, 1, 2, 3 と割り振られる.

配列の使い方について,番号を指定する以外は基本的に変数の場合と同様である.

それでは,配列を使った例を見てみよう.以下は,{1, 2, 3, 4, 5}の入った配列 b の要素それぞれに 1 を加えたものを配列 a の要素として代入し,それを表示するプログラムである.

Sample1_7.c

```c
#include <stdio.h>

int main(void) {
    int a[5];
    int b[5] = {1, 2, 3, 4, 5};

    a[0] = b[0] + 1;
    a[1] = b[1] + 1;
    a[2] = b[2] + 1;
    a[3] = b[3] + 1;
```

```
    a[4] = b[4] + 1;

    printf("a[0]: %d¥n", a[0]);
    printf("a[1]: %d¥n", a[1]);
    printf("a[2]: %d¥n", a[2]);
    printf("a[3]: %d¥n", a[3]);
    printf("a[4]: %d¥n", a[4]);

    return 0;
}
```

順にみていこう.まず,

```
int a[5];
```

で,int 型,要素数 5 の配列 a を宣言する.

続いて,

```
int b[5] = {1, 2, 3, 4, 5};
```

で,b の 0 番目の要素を 1, 1 番目の要素を 2, … , 4 番目の要素を 5 として初期化している.

続いて,プログラムでは a[0]～a[4]に値を代入している.配列も変数と同様に,計算したり代入したりできていることが分かるだろう.

最後に,その値を printf で表示している.

今回の例では,配列の要素数が 5 個と少なめだったが,もしもっと要素数の大きな値,例えば 10000 くらいだったらどうだろう.同じような処理を 10000 個も書くのは現実的ではない.そのような問題を解決すべく「繰り返し処理」があるのだが,それはまた別のときに説明する.

10. char 型の配列

さて,文字列が「文字の集まり」なら,文字列は char 型の配列を使って以下のように表現できそうである.

```
char str[6] = {'H', 'e', 'l', 'l', 'o', '¥0'};
```

ここで,とは文字列の終端を表す文字で「ヌル文字」と呼ばれている(NULL とは「何もない」ことを表す言葉).文字列を表すためには,最後にヌル文字を入れることを忘れてはいけない.しかし,いちいちヌル文字を入力するのは面倒であるし,カンマやシングルクォーテーションを入力するのは少し面倒である.そのような問題を解決するために,以下の省略記法が用意されている.

```
char str[6] = "Hello";
```

ヌル文字も省略できたし,ダブルクォーテーション二つで済んだ.しかしまだ問題がある.今回は Hello という文字が 5 文字だから,ヌル文字を含めて要素数 6 の配列を宣言すればよいことが分かる.しかし,

askdjflaksdjflkjsknkadnlskanvl

のような文字の数を手で数えるのは辛い.また,キーボードで文字を入力するとき,何文字の入力があるかわからないときもある.実は,初期化のときは配列の要素数を省略できる.

```
char str[] = "askdjflaksdjflkjsknkadnlskanvl";
```

宣言のときは要素数を省略できないので,多めに要素数を取っておくのが良い.

```
char str[100];
```

もちろんこの配列 str には 100 文字以上の文字を入れてはいけない.もし 100 文字以上入れる可能性があるのなら,もっと大きな要素数で初期化するのが良いだろう.

ではこれを踏まえて,文字列を入力してそれを単に出力するだけのプログラムを書いてみよう.

Sample1_8.c

```
#include <stdio.h>

int main(void) {
    char a[100];

    printf("input: ");
    scanf("%s", a);
    printf("output: %s¥n", a);

    return 0;
}
```

実行結果は次のようになる. 斜体部分は,キーボードからの入力例を示している.

```
input: Hello,Everyone.
output: Hello,Everyone.
```

要素数 100 の配列 a を宣言した後,キーボードの入力を求めて,入力した文字列を a に入れている.

文字列を入れるとき,scanf のダブルクォーテーション内に書くのは%s.

%s のときは,カンマの後の a には&をつけない.

最後に printf で単に a の中身を表示している.文字列は%s で表示することができる.

[補足]

Hello, World のような入力は(scanf では)できない.スペースで区切ると,そこで入力を取りやめてしまうので,a には”Hello,”までしか入らない.

11. これで Practice A は解ける？

解けます.解いてみましょう.

12. 演習問題を解くときの注意点

次に示す演習問題を解くにあたって,いくつか重要な点をここで述べる.

・演算の項でも述べたが,正整数同士の割り算は切り捨てになる.

・余りを求める演算%について,次の視点を持っておくことが望ましい.

　・余りには周期性がある

　・A％B とは「A から B の塊をできるだけ取り除いた余り」

※負整数を含む割り算の場合,切り捨てか切り上げかはコンパイラによって異なる.余りの演算に関しても,答えが負数なのか整数なのかはコンパイラによって異なる.負数を含む割り算や剰余算は基本的に使うべきでない.

13. 演習問題

[入出力の問題]

　Practice A

　ABC068_A

[計算問題]

　ABC001_A

　ABC039_A

　ABC005_A

　ABC043_A

　ABC026_A

　ABC076_A

　ABC055_A

　ABC057_A

　ABC087_A