プログラミング入門
#1 入出力 変数 繰り返し 配列

高校 3 年 赤澤侑

2021/05/12

1　こんにちは、世界！

みなさんこんにちは[1]、高校 3 年の赤澤です。本シリーズ「プログラミング入門」では中学 1 年生に向けて C++ によるプログラムの書き方を説明していきます。厳密さよりも寧ろ、わかりやすさと、幅広い話題に触れ、皆さんの興味のある分野を発掘することの 2 点に重きを置きました。かなり発展的な内容を含みますが、全てを理解する必要はありません。楽しいプログラミングライフを送りましょう！

さて、人間にとって挨拶は重要です。本稿のはじめで私も挨拶をしたとおり、初対面の方には必ず挨拶をする必要があります[2]。それではまずはコンピュータに挨拶をしてみましょう！以下のソースコードを実行してください[3]。

Source Code (1): Hello, World!

```cpp
#include <iostream>
using namespace std;

int main(){
    cout << "Hello, World！" << endl;
    return 0;
}
```

このコードを実行してみると次のような出力が得られます。

Input/OutPut (1): Hello, World! - **実行結果**

```
Hello, World！
```

これは `Hello, World!` という文字列を出力するプログラムです[4]。

では、プログラムの内容を簡単に説明しましょう。早速、1 行目から解説……といきたいところですが、以下に記す部分は「おまじない」[5]だと思ってあまり意味を考えずに記述するようにしてください。最初のプログラムの解説としては少し難しいからです。

---

[1] ここでは世間一般に通じている挨拶を用いましたが、マイコン部では挨拶として「グッドモルﾆｯﾋ！ﾆﾋﾋﾋﾋ！」と言うことが慣習となっています。

[2] プログラムの書き方なんかよりよっぽど重要なことです。挨拶、ダイジ。

[3] 私の tex 環境上、半角の exclamation mark をソースコード上に出力できませんでした。実際にコードを書く際には半角で入力するほうが好ましいです。

[4] Hello, World というのは初学者が最初に書くコードとして非常に有名なものです。C 言語を解説した伝説的名著である、「プログラミング言語 C」という書籍が初出とされています。

[5] よくわからないけどこれを書いておかないとプログラムが動かないもの、という意味です。

Source Code (2): **プログラムの雛形**

```
1 #include <iostream>
2 using namespace std;
3
4 int main(){
5     //処理内容
6     return 0;
7 }
```

`//処理内容`と書いてある部分にプログラムの内容を記述します。この雛形についてはコラムや後の章で詳しく説明しますが、今はそういうものだと思っておいてください。しかし、次のことは極めて重要なので覚えておいてください。すなわち、プログラムは`int main()`と書いてある次の行からスタートし、`int main()`の後にある`{}`[6]の中に記述された内容が実行されます。

では処理内容を詳しく見ていきましょう。処理内容は次のようになっています。

```
1 cout << "Hello, World!" << endl;
```

まず、`cout`という部分に注目します。これはよく「シーアウト」と発音されるものでなにかを出力するときに使います[7]。ではこの cout は何者か、といえば、「もらったものを順番に出力するマン」ということができるでしょう。

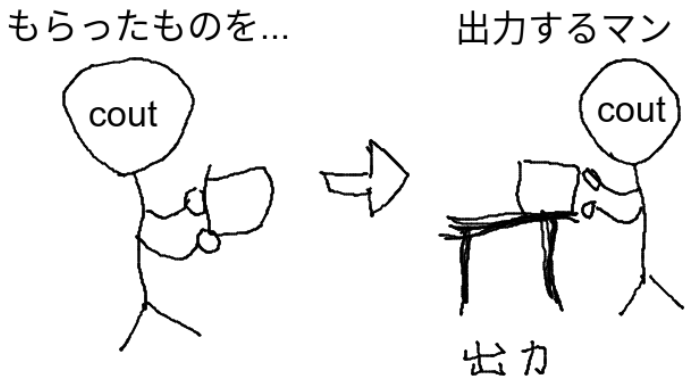

図 1　もらったものを順番に出力するマン

この出力するマンに出力してほしい物を渡すための演算子[8]こそが`<<`です。このソースコードでは cout に"Hello, World!"と endl を順番に渡しているのです。先程説明したとおり、cout は渡されたものを順番に出力してくれるのでソースコードのように<<をいくつもつなげることができます。たとえば、処理内容を次のように書き換えてもソースコード 1 と同じ出力が得られます。

```
1 cout << "Hello" << ", " << "World" << "!" << endl;
```

ここで、出力したい文字は`""`[9]によって囲まれていることに気づくと思います。このように、出力したい文

[6] 開きカッコ`{`は 4 行目の最後、閉じカッコ`}`は 7 行目にあります。

[7] cout は console out の略とされます。console とはコンピュータとやり取りするための入出力装置 (キーボードとかディスプレイとか) のことを指しますが、ここではコマンドライン (文字がいっぱい出ているかっこいいウィンドウのやつ) を意味します。環境によっては入力、出力などと書かれていることもあります。

[8] 挿入子といいます。

[9] "はダブルクオーテーションと読みます。似た文字に’がありますが、こちらはシングルクオーテーションあるいはシングルクオー

字列は`""`でくくってやらねばなりません。試しに""を外してみると、エラーが出てしまうでしょう。

では一番最後の`endl`とは何でしょうか？これは改行を意味するものです。これを cout に渡すことによって改行することができます。プログラム中での改行は C++ においては動作に影響を与えない[10]ので注意しましょう。また、endl の部分を`"\n"`とすることもできます。すなわち、"\n"を改行文字といい、\n 自体が改行を意味する特殊な文字[11]です。以下のソースコードが最初のものと同じ出力を与えます。

```
1 cout << "Hello, Wolrd!\n";
```

最後に、5 行目の一番最後の文字である、`;`[12]についてです。これは文の終わりを意味する文字です。ちょうど、日本語において文の終わりに「。」を、英語において文の終わりに “.”を置くのと同じように C++ においては; を置く必要があります。

以上で第 1 章第 1 節は終了です。出力はプログラムの基本なのでしっかりとおさえましょう。しかし、慣れていけば自然と書けるようになるので、無理に「覚えよう！」と頑張る必要はありません。気楽にやりましょう。

2 変数

次のような場合を考えてみましょう。

> **Problem 1 : 配送手数料**
>
> マイコン通販社では商品を配送する際に商品代金に追加で配送手数料 300 円をお客さんに払ってもらっている。商品代金が X のとき、お客さんが払う金額を求めるプログラムを作成せよ。

計算自体は簡単です。ある料金 X に 300 を足してあげればいいのです。これを $X + 300$ と表現します[13]。ではこれをどのようにプログラム上で表現すればよいのでしょうか？そもそもプログラムというのは何かしらの極めて抽象化された処理を実行するものです。この例題では商品代金という抽象的な値に対する演算を考えています。これを実現する機能が変数なのです。

2.1 変数と変数型

まずは変数について説明します。変数とは**何かしらの情報を保持しておくメモリ上の領域**のことです。では一つずつ詳しく説明します。はじめに、何かしらの情報とはなにを指すのでしょう。それは数字であったり文字列であったり画像であったり音声であったり様々です[14]。ここでは変数が保持する情報を数字である、

トと読みます。一般には引用を意味する記号です。また英語におけるアポストロフィも同じ記号を用いて表現しますが、意味は違うので注意しましょう。

[10] これは、意味のある語 (int、main、cout、endl など) の途中を除いてプログラム中では任意の場所で改行をして良い、ということを意味します。見やすくなるように適宜改行しましょう。一方で、改行がプログラムの動作に影響を与える言語もあります。Python などがその一例です。厳密には Python においてはインデントが動作に影響を与えますが、こういった言語では改行の仕方も言語仕様に従いましょう。

[11] 制御文字という。

[12] セミコロンと読みます。コロン:と間違えないようにしてください。

[13] このような文字を含んだ式のことを文字式といいます。詳しくは代数の教科書を参照してください。

[14] コンピュータ上ではこれらは全て数字に置き換えて扱うので究極的には「変数は数字を保持している」ということができるでしょう。ただし、人間にとってこれらのデータを数字に置き換えることは極めて煩雑な処理なので人間がそれを意識しなくて済むように様々な型の変数があるのです。

と仮定して話を進めていきます。

では、メモリとは何でしょうか？メモリはコンピュータが行っている処理に必要なデータを保存しておく場所です[15]。

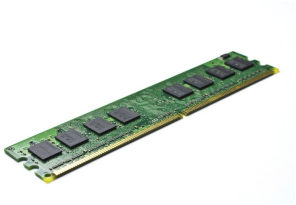

図 2　メモリ (物理)

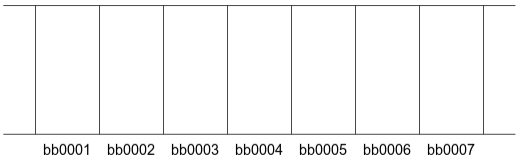

図 3　メモリ内部の模式図

このメモリ上にある黒いチップがメモリチップで実際にデータを保持しています。メモリチップの中をさらにさらに細かく見ていくと図 3 のような小さな区画がたくさん詰まっています。この小さな区画がメモリ上でのデータの基本単位です[16]。

変数とはこの区画をいくつかひとまとめにして名前をつけたものだと考えてください。

いま、整数を保持する変数 X を作ったとしましょう。するとコンピュータは例えばメモリ上の 4 つの区画を占拠して X という名前をつけます。図 4 中の bb0002 から bb0005 の 4 つの区画をひとまとめにしたものが変数 X です。

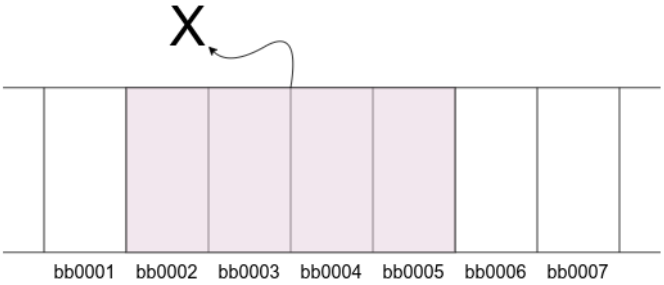

図 4　変数を作る

ところで、この変数 X には入れられる数の大きさの上限があります。具体的には 2,147,483,647 です。これは各区画が表すことのできる数に上限があるからです。詳細は注 15 を確認してください。では、これより大きな数を表すことはできないのでしょうか？実は変数には「型」があり、それによってどれくらいの数が扱えるかが変わってきます。2,147,483,647 より大きな数を扱うことのできる型も存在します。以下の表を見てください。用途によって様々な変数[17]があるということがわかると思います。具体的にこれらをどう扱うかは後述しますので、今は「いろんな型があるんだなぁ〜」と思ってくだされば結構です。

[15] メモリには DRAM と SRAM とがあります。前者は主記憶装置 (メインメモリ) として PC に搭載しますが、後者は CPU 内部にレジスタやキャッシュメモリとして組み込まれています。後者のほうがデータにアクセスするための時間が極めて短い一方で容量が少なく、価格もとても高いです。DRAM では読み書きの速度が遅く、CPU の処理性能を持て余してしまうので、メインメモリからキャッシュメモリにデータを読み出しておいて、それに CPU がアクセスすることで高速なアクセスを実現します。本文ではデータはメインメモリにあるという体で記述されていますが、当然キャッシュ上にあるということも考えられます。なお、レジスタは演算器にくっついており、演算に使うデータを保持して演算を高速化します。

[16] この区画ひとつ分の情報量が 1Byte(バイト) です。メモリを操作する上ではこの Byte が基本的な単位になりますが、情報の基本単位は bit(binary digit; ビット) です。bit は 0 か 1 かのどちらかの状態を持っており、8bit で 1Byte です。いま、8 つの bit が並んでいるとすると、これはちょうど 8 桁の 2 進数があるものと考えられます。つまり 1bit は 2 進数における 1 桁分ということもできます。よって 1Byte は $2^8 = 256$ 通りの数値を表しうるというわけです。

[17] ここに挙げた以外にも `short` や `long`、`float` など様々なタイプの変数があります。また表中の変数は全て負の値を入れることができますが、負の値を認めない `unsigned` というキーワードをつけた変数も存在します。

表 1　さまざまな変数型

型名	種類	区画の個数	上限など
int	整数	4 つ	-2,147,483,648 ～ 2,147,483,647
long long	整数	8 つ	-9,223,372,036,854,775,808 ～ 9,223,372,036,854,775,807
double	小数	8 つ	2.22507e-308 ～ 1.79769e+308
char	文字	1 つ	-128 ～ 127

2.2　変数の作成

では、具体的に、プログラム中で変数を作るにはどうしたら良いのでしょうか？次のコードを実行してみてください。

Source Code (3):　**変数の作成**

```
1  #include <iostream>
2  using namespace std;
3
4  int main(){
5      int a;
6      a = 10;
7      cout << a << endl;
8
9      int b = 100;
10     b = b + 3;
11     cout << b << endl;
12
13     cout << a + b << endl;
14     return 0;
15 }
```

プログラムを解説しながら変数の作り方について説明します。雛形の部分については説明を省きます。まず、5 行目に注目してください。`int a` という部分が変数を作成している箇所です。C++ において変数を作るときにはプログラムに「a という変数を作ってね～～」という命令をしなくてはいけません。このことを「変数を宣言する」といいます。変数を宣言するための書式は以下のとおりです。

```
1  （型名）（変数名）;
```

すなわち、宣言したい変数の型名をまず記し、その後にスペースを開けて[18]変数に自由に設定できる名前であるところの変数名を書きます。このように作成した変数には値を代入したり、出力したりすることができます。この処理が 6 行目と 7 行目です。7 行目を詳しく見てください。文字列を出力するときには"で囲みましたが、変数の中身を出力するときには"をつけず、ただ変数名を書きます。もし"で囲んでしまうと変数名それ自体が出力されてしまいます[19]。

9 行目に注目してください。`int b = 10` とあります。このように変数を宣言したときに初期値を設定することができます。この処理を初期化といいます。変数を宣言するときには入力を受け取ることが確定している場合などを除き、必ず初期化するようにしましょう[20]。

[18] スペースがないとコンピュータにとっては変数の型名だと認識できません。型名のようにそれ自体がプログラムを制御する意味を持つ言葉で変数名などにしてはいけない言葉を予約語あるいはキーワードといいます。キーワードを変数名などに使ってしまうと、制御用の言葉なのか、変数などであるのかコンピュータにとって理解できないからです。

[19] つまり `cout << "a" << endl;` とした場合には `a` という出力が得られるということです。

[20] 変数にもいくつかの種類があり、そのうちのいくつかはプログラマがなにも書かなくても初期化が行われます。また、そうでないものも、処理系によっては気を利かせて初期化してくれる場合が多いです。しかしながら、C++ の言語仕様として規定された挙動ではないので初期化することが望ましいです。

変数に対して演算を行うこともできます。10 行目を見てください。`b = b + 3` という記述はなかなか奇妙に思えるかもしれませんが、C++ において=は等式のように左辺と右辺が等価であることを**意味しません**。イコール記号は代入[21]を意味します。すなわち、「b に 3 足したものを b に代入してくれ」という意味です。

ところで C++ において算術演算を行うための演算子には以下のようなものがあります。

表 2　さまざまな演算子

演算子	仕様例	意味
+	a + b	a と b を足した値を計算する
-	a - b	a から b を引いた値を計算する
*	a * b	a に b を掛けた値を計算する
/	a / b	a を b で割った商を計算する
%	a % b	a を b で割ったあまりを計算する
++	++a または a++	a に 1 加算する
--	--a または a--	a に 1 減算する
+=	a += b	a に a + b を代入する (a = a + b)
-=	a -= b	a に a - b を代入する (a = a - b)
*=	a *= b	a に a * b を代入する (a = a * b)
/=	a /= b	a に a / b を代入する (a = a / b)
%=	a %= b	a に a % b を代入する (a = a % b)

また、出力する際に計算を挟むこともできます。13 行目を確認してください。

以上で、第 2 節第 2 項は終了です。本当ならば各変数の特徴などに迫りたいところですが、それらはコラムにおいて記述します。

2.3　入力受け取り

さて、今回の課題は抽象的な、代金という値に 300 を足して出力しよう、というものでした。抽象的な値そのものである変数を作ることができたので、今度は実行時に値を受け取る方法をご紹介しましょう。使うのは `cin` というものです。次のソースコードを見てください。

Source Code (4):　cin の使い方

```
1  #include <iostream>
2  using namespace std;
3
4  int main(){
5      int x;
6      cin >> x;
7      cout << x << endl;
8
9      int a, b;
10     cin >> a >> b;
11     cout << a + b << endl;
12     return 0;
13 }
```

さて、6 行目に cin が出てきました。cin は「シーイン」とよく発音され、ちょうど cout と対を成すものです。すなわち cin は console に入力されたものを順番に変数に代入していきます。このとき、呼ばれてい

[21] あるいは初期化を意味します。int などの整数型ではあまり効果を実感することはありませんが、代入と初期化は全く異なる処理だ、ということはしっかりと思えておきましょう。

る回数分だけの入力が完全に終わるまでプログラムは入力を待ち続けます。

10 行目に注目してください。cin はいくつもの変数に連続して代入することができます。この場合、2 つの数が入力される[22]までプログラムは 10 行目で止まり続けます。なお、9 行目では `int a, b` と 2 つの変数が同時に宣言されていますが、同じ型の変数を複数個宣言する場合にはこのような記法が可能です。

2.4 Problem1 の回答例

では Problem1 配送手数料の回答例を提示します。

Source Code (5): Problem1 の回答例

```
1 #include <iostream>
2 using namespace std;
3
4 int main(){
5     int price;
6     cin >> price;
7     cout << price + 300 << endl;
8     return 0;
9 }
```

今までに説明したことを素直に実装すれば正しい答えを返すプログラムを書けることでしょう[23]。

3 繰り返し

3.1 for ループ

次に扱うのは繰り返しです。例題として次のプログラムを考えてみましょう。

Problem 2 : 平均値太郎

平均値太郎は平均値が大好きだ。N 個の値が与えられるのでそれらの値の平均値を求めよ。

制約

$2 \leq N \leq 100$

N は整数である。N 個の値はそれぞれ正整数で 1000 以下である。

さて、平均値は次のように計算される値である[24]というのは皆様のよく知るところでしょう。

$$(\text{平均値}) = \frac{(\text{データの和})}{(\text{データの個数})}$$

プログラムでこれを計算する場合は**繰り返し処理**をする必要があります。C++ では様々な繰り返し処理が

[22] 入力は一般にスペースあるいは改行区切りで与えることが多いです。

[23] 今までの説明では全て変数名は一文字でした。しかしながらキーワードでなければ基本的にどれほど長い文字列でも変数名になりえます。ただし、次のことに注意する必要があります。すなわち (1) 大文字小文字の英字、数字、アンダースコアからなる、(2) 数字から始めてはいけない、(3) アンダースコア、大文字の順につなげてはいけない、などです。

[24] 数式で表すと次のようになります。

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N}$$

なお $\bar{x}$ は平均値、N はデータの個数、x_i は i 個めのデータを意味します。$\sum$ というのは総和を意味する記号でこの例で言えば「i を 1 から始めて N まで進める。このときの各 i についての x_i の総和を計算する」というイメージです。

利用できますが、今回はその中でも最も基本的な **for ループ**[25]を使ってみましょう。

Source Code (6): Problem2 の回答例

```cpp
#include <iostream>
using namespace std;

int main(){
    int N;
    cin >> N;

    double sum = 0;
    int x;
    for (int i = 0; i < N; ++i) {
        cin >> x;
        sum += x;
    }
    cout << x / N << endl;
    return 0;
}
```

10 行目を見てください。for という文字を見ることができます。この直後の丸括弧の中でループの設定を行いその後の中括弧内にループの処理を書きます。設定に指定するべき内容は 3 つ、順に変数の宣言[26]、ループ条件[27]、加算処理[28]とします。

```cpp
for（変数の宣言；ループ条件；加算処理）{
    ループの中身
}
```

これが for 文の一般的な書き方ですが当面の間は以下のような「0 から $N-1$ まで N 回繰り返すループ」が書けるようになればよいです。すなわち変数の宣言が `int i = 0`、ループ条件が `i < N`、加算処理が `++i` であるようなループです。

```cpp
for (int i = 0; i < N; ++i) {
    ループの中身
}
```

ではこのようなループが実行するステップを順に見てみます。コンピュータは上記に 3 行に対して次のような動作をしています。

1. まず `int i = 0;` を実行してループカウンタ[29]を作成する。
2. `i < N` を評価する。条件が真であればステップ 3 に進む。条件が偽であればステップ 5 に進む。
3. ループの中身を実行する。
4. ループカウンタ `i` に 1 加算して、ステップ 2 に戻る。
5. 終了する。

[25] for 文とも言います。
[26] この 1 つ目の設定は初期化ともいい、ここで宣言される変数をループカウンタ、カウンター変数、ループ変数あるいは単にカウンタなどといいます。ループを実行するとき、最初に 1 回だけ実行されます。省略可能です。
[27] ループを実行する条件に使用する変数はループカウンタである必要はありません。ループの中身を実行する直前に評価され、真であれば中身が実行されます。省略可能です。これを省略するとループの中身で `break` や `return` しない限りループし続けます。俗に言う無限ループです。
[28] 加算である必要はなく四則演算や、シフト演算を行うことができます。また演算に使う変数はループカウンタである必要はありません。省略可能です。
[29] 注 26 を参照。

ちょっとわかりにくいので図にしてみました。

図 5　ループのイメージ

例えば次のようなコードを書いてみれば for ループの動きがよくわかるでしょう。

Source Code (7): for ループの動き

```
#include <iostream>
using namespace std;

int main(){
    int N = 10;

    cout << "#1 : " << endl;
    for (int i = 100; i < N; ++i) {
        cout << ' ' << i;
    }
    cout << endl;

    cout << "#2 : " << endl;
    for (int i = 0; i < N; ++i) {
        cout << ' ' << i;
    }
    cout << endl;

    cout << "#3 : " << endl;
    for (int i = 0; i < N; i += 2) {
        cout << ' ' << i;
    }
    cout << endl;

    cout << "#4 : " << endl;
    for (int i = 5; i >= 0; --i) {
        cout << ' ' << i;
    }
    cout << endl;

    return 0;
}
```

Input/OutPut (2): for ループの動き：出力

```
#1 :

#2 :
 0 1 2 3 4 5 6 7 8 9
#3 :
 0 2 4 6 8
#4 :
 5 4 3 2 1 0
```

3.2 ループの制御〜break, continue〜

さて、先の節では基本的な for ループの使い方について説明しました。この節ではループを中断したり、スキップしたりする方法を説明します。

まずは以下のプログラムを確認してください。

Source Code (8): break と continue

```
1  #include <iostream>
2  using namespace std;
3
4  int main(){
5      int N = 10;
6
7      for (int i = 0; i < N; ++i) {
8          if (i == 3) {
9              break;
10         }
11         cout << ' ' << i;
12     }
13     cout << endl;
14
15     for (int i = 0; i < N; ++i) {
16         if (i % 2 == 1) {
17             continue;
18         }
19         cout << ' ' << i;
20     }
21     cout << endl;
22
23     return 0;
24 }
```

Input/OutPut (3): break と continue：出力

```
 0 1 2
 0 2 4 6 8
```

break という文を実行するとその場でループから抜けることができます。8，9，10 行目を見てください。これは「i == 3 という条件を満たしたときにループから抜ける」ということを意味しています。実際出力の 1 行目を見てみると 0，1，2 と出力した後、改行されておりループから抜けていることがわかります。

continue という文を実行すると continue 文以降に書いてある「ループの中身」を実行せずに、ループカウンタの更新へとスキップします。16 行目から 18 行目をみると i が奇数であるときには出力処理を行わず、i を更新するように命令していることがわかります。出力の 2 行目を確認すると、正しくこのように動いてい

ることがわかります。

`break` 文や `continue` 文は条件分岐などで代用できることもありますが、「枝刈り」という計算量削減テクニックを用いる場合などには非常に重要であり、プログラムのネストを小さくし、簡潔な記述を可能にしてくれます。余裕があれば使ってみましょう。