＜C＋＋入門＞

目 次

１章　まえがき

C++とは？

　C++は1983年に作られて以降、その圧倒的な速度・メモリ消費の少なさから世界中で使われてきました。速度やメモリを重視しなくてはならない組み込み系との相性も良く、我々が使っているArduinoも内部でC++を動かしています。
なので、**ArduinoとC++は文法がほぼ同じです。**

このテキストのコンセプト

　このテキストではArduinoでのプログラミングがより上達すること、
パソコンで上で動くプログラムが書けるようになることを目指していきます。

注意点

2章 C++の基礎文法

まずは「ＫＨＥＣ！！」と表示するプログラムを書いていきます。

```
#include <iostream>

using namespace std;

int main(){

    cout << "KHEC!!" << endl ;

}
```

```
#include <iostream>
```

ここで、<iostream>というファイルをインクルードしています。

<iostream>は標準ライブラリなのでどこかからダウンロードしなくても

最初から入っています。

<iostream>は入出力に関する機能があります。

```
using namespace std;
```

これはおまじないです。。

名前空間の章で説明するので許してください。。

```
int main(){
    cout << "KHEC!!" << endl ;
}
```

C++のプログラムは "main" という関数で処理を行います。

Arduinoでの setup/loop みたいなものです。

おそらく、3行目が分かりづらいと思います。

cout << 何か << endl

こうやって何かを囲ってやると出力できます。

何か は文字でなくても、変数でも大丈夫です。

たとえば。。

```
int number=5;
cout << number << endl ;
```

このようにすると　　5　　と出力されます。

何がうれしいのでしょうか・・

これは複数の変数を表示するときに便利です。

"私たちは (部活名) です。部員は (部員数) 人います。"

って表示する際、

ArduinoではSerial.print()を何個も書かなくてはいけないと思います。

これを

```
cout << "私たちは" << club_name << "です。"

          << "部員は" << club_member_num << "人います。" << endl;
```

という風に、変数を交えて書けます。

すごい。

今度は入力を試してみましょう。

```
#include <iostream>

using namespace std;

int main()

{

    int number;

    cin >> number;

    cout << number << endl;

}
```

5行目で入力をしています。

cin >> 変数

という風に変数を囲むと処理が途中で止まります。ここでターミナルに数字を打ち込みEnterを押すと、入力が変数に入ります。

このプログラムでは number に入力が入ります。

５って入れたら 5 と返ってきます。

これは cout と同じく、複数の変数を扱えます。

入力された数字2つを足して出力するプログラムを書いてみましょう。

```
#include <iostream>

using namespace std;

int main()

{

    int number1;

    int number2;

    cin >> number1 >> number2 ;

    cout << number1+number2 << endl;

}
```

6行目の

cin >> number1 >> number2;

で、入力を２回、number1 と number2 で受け取っています。

それを７行目で足し算して出力してます。

文字列を出力することもできます。

```
#include <iostream>

using namespace std;

int main()

{

    cout <<"electronics club"<< endl;

}
```

文字列を入力して変数に代入するにはArduinoのようにString型を使うわけですが、C++ の場合は #include <string> を事前にしておかないと使うことができません。またC++ではString str; ではなく string str; のように小文字で書きます

```
#include <iostream>

#include <string>

using namespace std;

int main()

{

   string str;

   cin >> str ;

   cout << str << endl;

}
```

PRINTF

cin や cout は C++の機能の一つですが、Arduinoではこれらに近い構文でSerial通信をしたりすることはできません。しかし、C言語の出力機能である printf はESP32であれば Serial.printf()という形で使うことができ、また printf の姉妹機能である sprintf は Arduino Uno でも使用可能です。

printf(テンプレート文字列, 値1, 値2, ...);

```
#include <iostream>

using namespace std;

int main()

{

   int a = 1;

   int b = 10;

   printf("a= %d, b= %d", a, b);

}
```

出力

```
a= 1, b= 10
```

このように、テンプレート文字列の中にある%dが、その後ろの値に置き換わります。このようにprintfは文字列を表示した上で、更に%から始まる特殊な文字を後ろに書いた値に置き換えます。

置き換えのパターンはこのようになっています。

このパターンはかなり詳細に設定することができるのですべてここに書くことはできません。このサイトを参考にしてください。

https://www.k-cube.co.jp/wakaba/server/format.html

指定子	対応する型
%c	char
%s	char* (文字列)
%d	intなど±のある整数
%u	unsigned intなどマイナスを含まない整数
%x	整数を16進法で出力する
%f	floatのような少数を出力する
%g	少数を自動でいい感じに出力する
%ld	longのようなintより大きい値を取れる整数
%lu	unsigned long

なおstring型の文字列を表示するときには左ではなく右のようにしないと表示されないので、注意してください。

X

```
 string str = "aaa";

printf("str -> %s", str);
```

○

```
string str = "aaa";

printf("str -> %s", str.c_str());
```

問題

1, 入力された数字を2倍して返すプログラムを書いてください。

2, 入力された数字が13で割り切れるか、"YES"か"NO"で出力してください。

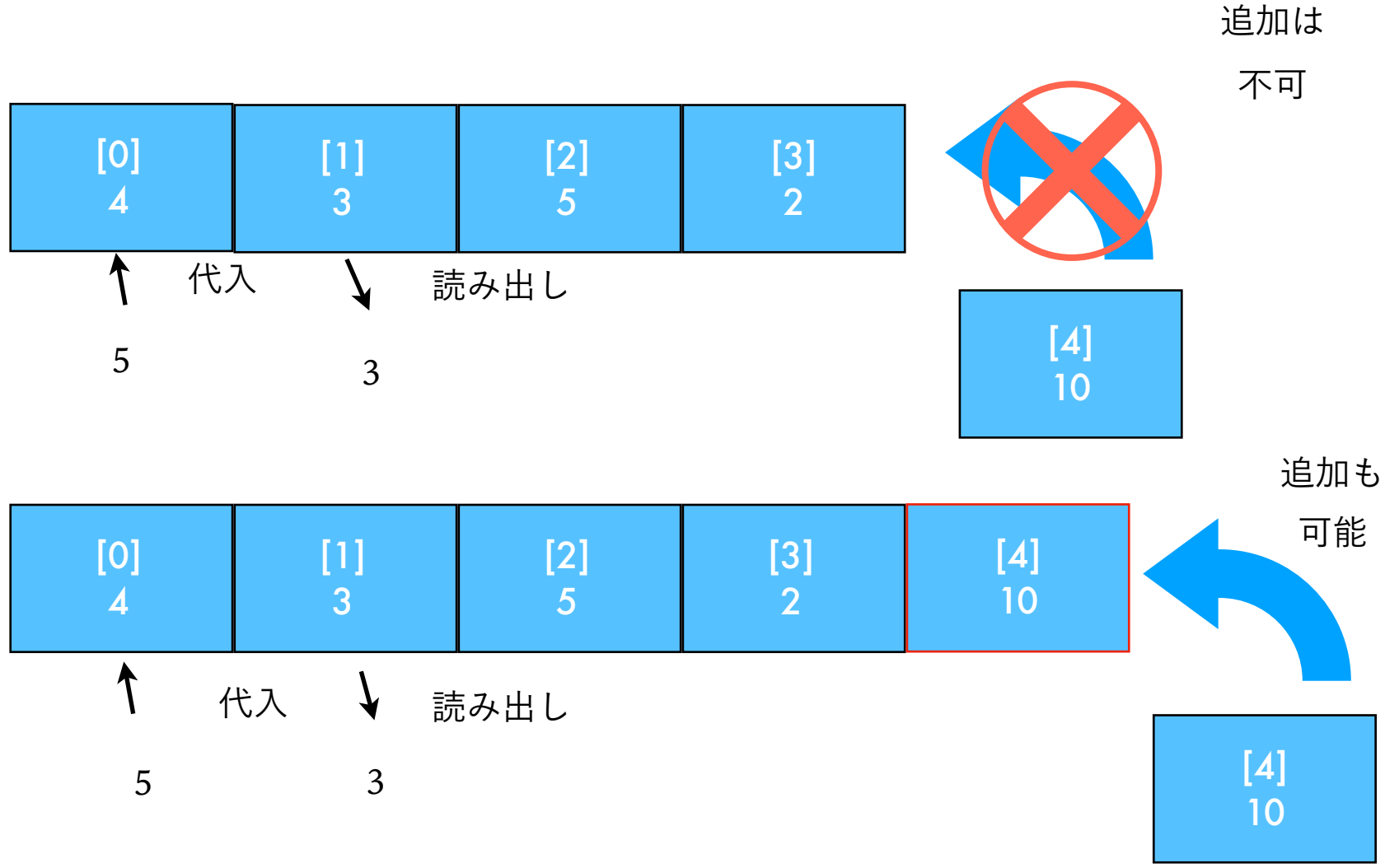
3章 VECTOR・名前空間

1. VECTOR

1.1 VECTORとは

Arduinoで使えた配列の要素数はコンパイル時に決まってしまい、必要に応じて要素を追加していったり、逆に要らなくなった要素を削除したりすることができません。しかしC++の標準ライブラリである"vector"を使えば、プログラムの実行中に要素を増やしたり減らしたりすることができる「可変長配列」を作ることができます。

配列



なおvectorはESP32でも使うことができます。

当然Arduino UNOでは非標準ライブラリを追加しない限り使用できません。

1.2 VECTORの基本的な使い方

コード例

```
#include <vector>
#include <iostream>
using namespace std;
int main(){
    vector<int> array = {2,4,5};
    cout << array[0] << endl;
    array[0] = 3;
    array.push_back(10);
    for(int i = 0; i < array.size(); i++){
        cout << array[i] << endl;
    }
}
```

出力

```
2
3
4
5
10
```

vectorを使用する際には**#include <vector>**でライブラリを読み込む必要があります。

初期化:　vector<中に入れる型名> 配列名;

vector<中に入れる型名> 配列名 = {初期化時に代入する要素};

<>の中にはvectorに収納する値の型を書きます。int型の値を入れるならvector<int>、stringならvector<string>とします。この構文については後の章で説明します。

要素読み出し:　　配列名[インデックス]

要素代入:　　配列名[インデックス] = 値;

要素の値の読み出しや代入は配列と同様です。

要素の追加:　　配列名.push_back(追加したい値);

これによってvector配列の一番後ろに値を付け足します。

要素数の取得:　　配列名.size()

vectorは通常の配列と違って要素数が変わるため、for文で配列の全要素を取得したりするにはfor文を実行する前に要素数を取得する必要があります。配列名.size()という関数を実行すると、その時点での配列の要素数を返します。

1.3 VECTORを返す関数

```
#include <vector>
#include <iostream>
using namespace std;
vector<int> func1(){
    vector<int> a = {2,4,6};
    return a;
}
vector<int> func2(){
    return {1,3,5};
}
int main(){
    vector<int> array1 = func1();
    vector<int> array2 = func2();
}
```

vectorのもう一つの特徴として関数の戻り値になれることがあります。

このように戻り値にvector<要素の型>を指定すればvector配列を返すことができ、またfunc2のように配列を初期化するときに使う{}を直接returnの後ろに書くことで、vectorの初期化まで省略して返すことができます。

問題

1. 数字を入力するとその値が配列に入った後現在の配列内のすべての要素が表示され、また数字を入力できるようになるプログラムを作成せよ。
   入出力例 （入力と出力を区別するため出力の行頭には矢印をつけるが、回答ではいらないこととする。）

```
1
-> {1}
2
-> {1,2}
5
-> {1,2,5}
```

2. 簡易的な単語保存ツールを作る。add "何らかの文字列" と入力するとその文字列を保存してその単語の番号を表示し、 read 番号 と入力するとその番号の文字列を表示するようにする。以下のコードに足りないところを補え。

```
#include <string>
#include <vector>
#include <iostream>
using namespace std;
int main(){
    while(true){
        string command;
        cin >> command;
        if(command == "add"){
            string str;
            cin >> str;
        }else if(command == "read")
            int number;
            cin >> number;
        }
    }
}
```

2．名前空間

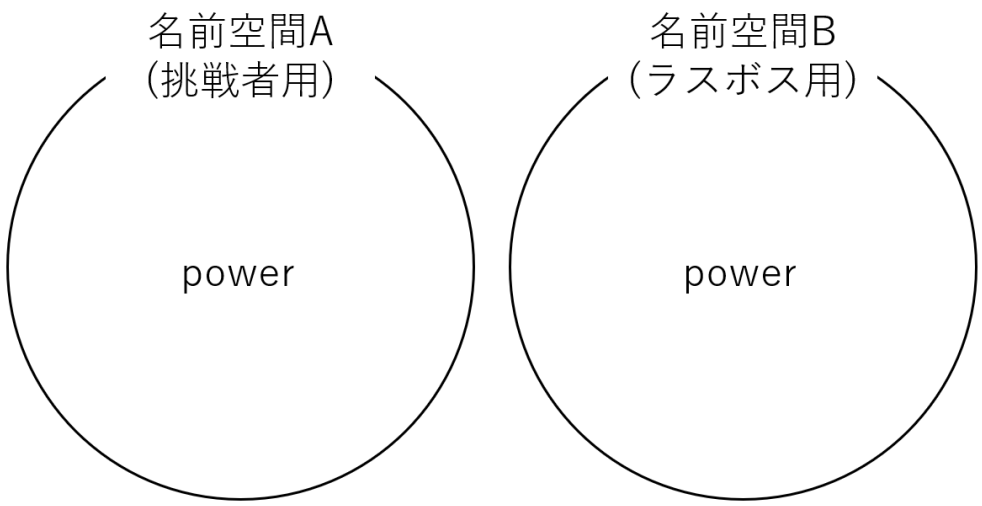
2.1 名前空間とは

プログラミングを行う際、異なる変数として使うつもりだった変数同士が、読み返すと同じ変数名でそこでエラーが発生していたことはありませんか?このような問題は、特に複数人で開発をする際によく発生します。そして、この問題を解決するために、「名前空間」が違えば同じ名前を用いて良いというルールが作られました。

例えば、ゲームのラスボス戦を開発する際、挑戦者の開発をする人とラスボスの開発をする人に分けたとします。そして、それぞれ別の名前空間で開発したとします。この状態であれば、開発者同士が同じ変数名を使ってしまっても問題ありません。ここでは、挑戦者のHPは名前空間Aのpowerという変数名を、ラスボスのHPは名前空間Bの同じくpowerという変数名を用いたとして以下に図示します。

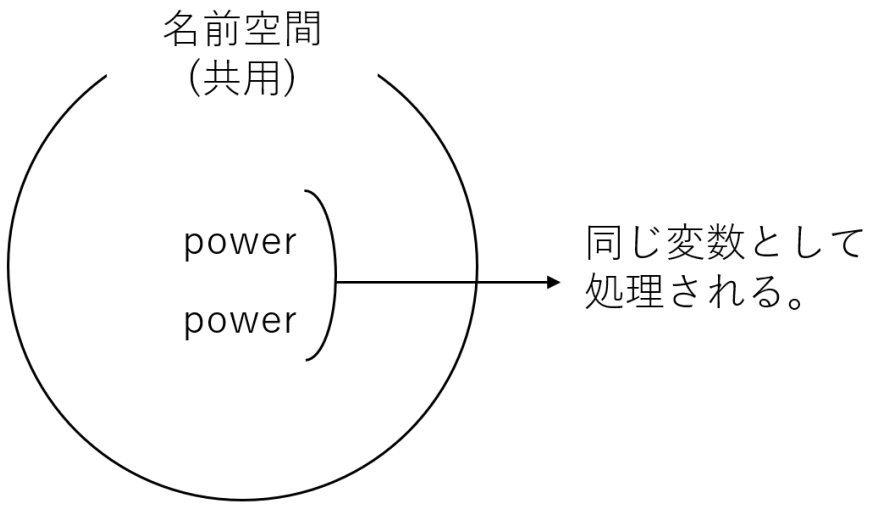
→HPは名前空間Aのpowerという変数に。

→HPは名前空間Bのpowerという変数に。



→名前空間が異なるのでpowerという変数2つはそれぞれ別の変数として問題なく機能する。

対して、同じ名前空間でpowerという変数名を違う変数にするつもりで2回使ってしまった場合を以下に図示します。



→名前空間が同じなのでpowerは同じ変数として処理され、挑戦者のHPとラスボスのHPは連動して減ってしまう。

2.2 名前空間の作り方

名前空間は以下の様な形式で作ります。但し、名前空間を使う前に、グローバルに記述しなければなりません。

```
namespace 名前空間名 {
    // 名前空間に入れたい識別子(注1)
}
```

注1) 識別子とは、変数名、関数名、クラス名等のことです。

名前空間に含むことが出来るのは、変数名、関数名、クラス名等です。vectorもクラスなので使うことが出来ます。

以下に名前空間aと名前空間bを作る場合のコード例を示します。

コード例

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

namespace a {
    void function() { cout << "KHEC_1" << endl; };  //名前空間aの関数
    int number = 1;  // 名前空間aの変数
}
namespace b {
    void function() { cout << "KHEC_2" << endl; };  // 名前空間bの関数
    int number = 2;  // 名前空間bの変数
}
//この後で名前空間を使う。
```

さらに、同じ名前空間であっても以下の様に分割して作ることが出来ます。同じ名前の名前空間同士は1つの名前空間として扱われるので、このソースコードは先程のソースコードとやっていることは変わりません。

コード例

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

namespace a {
    void function() { cout << "KHEC_1" << endl; };
}
namespace a {
    int number = 1;
}
namespace b {
    void function() { cout << "KHEC_2" << endl; };
}
namespace b {
    int number = 2;
}
```

加えて、名前空間は以下の様にネストする(名前空間の中に名前空間を作る)こともできます。使い方については後で2.3で示します。

名前空間aの中に名前空間bを作るコード例

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

namespace a {
    void function() { cout << "KHEC_1" << endl; };
    int number = 1;
    namespace b {
        void function() { cout << "KHEC_2" << endl; };
        int number = 2;
    }
}
```

2.3 名前空間の使い方

2種類の名前空間の使い方を示します。

名前空間の使い方1

以下の様に名前空間名と識別子をセットにして名前空間の中の識別子を使うことが出来ます。「::」を「スコープ解決演算子」といいます。

```
名前空間名::識別子
```

例えば、名前空間aの中のpowerという変数を指定する場合は以下の様に書きます。

```
a::power
```

コード例

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

namespace a {
    void function() { cout << "KHEC" << endl; };
    int number = 1;
}

int main() {
    a::function();
    cout << a::number << endl;  // 名前空間aの中の変数numberを表示
    return 0;
}
```

出力

```
KHEC
```

問題1

名前空間aを作成し、以下の関数と変数を入れてください。ただし、スコープ解決演算子「::」 を用いて下さい。

・出力にint型の引数を表示する関数function

・「1」が代入されているint型の変数number

名前空間a内の関数functionをmain関数内から呼び出し、そらの引数をnumberとすることで以下の様に表示して下さい。

出力例

```
1
```

名前空間の使い方2

スコープ解決演算子を用いて、名前空間内の多くの識別子を使おうとすると名前空間名をいちいち書く必要が有り面倒である。そこで、以下の様に書くと同じスコープ内ではその名前空間の識別子をスコープ解決演算子を無しで使うことができます。これを「using指令」という。

```
using namespace 名前空間名;
```

これをグローバルに記述すると、using指令の後の部分なら、以下の様にのどこであってもその名前空間に属する名前を使うことができます。但し、using指令は名前空間を作った後に行う必要があります。

コード例

以下のコードでは名前空間aの中の変数numberを表示している。

```
#include<iostream>
using namespace std;
namespace a {
    void function() { cout << "KHEC_1" << endl; };
    int number = 1;
}
using namespace a;
int main() {
    function();
    cout << number << endl;
    return 0;
}
```

出力

```
1
```

問題

名前空間aを作成し、以下の関数と変数を入れてください。ただし、スコープ解決演算子を用いないで下さい。

・出力にint型の引数を表示する関数function

・「1」が代入されているint型の変数number1

・「2」が代入されているint型の変数number2

・「3」が代入されているint型の変数number3

名前空間a内の関数functionをmain関数内から3回呼び出し、そらの引数をnumber1、number2、number3とすることで以下の様に表示して下さい。

出力例

```
1
2
3
```

また、using指令が有効なのはそのスコープ内のみです。当然、スコープ内のみなのでインクルードしていない別のファイルでは有効ではありません。

例えば、以下のコード例の様に関数の中でusing指令を行った場合、他の関数にはそれが適用されません。

コード例(これは正しく動作しません)

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

namespace a {
    void function() { cout << "KHEC_1" << endl; };
    int number = 1;
}

int main() {
    using namespace a;
    function();
    cout << number << endl;
    hoge();
    return 0;
}

int hoge() {
    function();
    cout << number << endl;
    return 0;
}
```

→この場合、hoge内で名前空間aの識別子を使うことはできません。

ネストされた名前空間の使い方

ネストされた(名前空間の中にある)名前空間を使う際は、以下の様に指定する。

```
親の名前空間名::子の名前空間名::識別子
```

例えば、名前空間aの中の名前空間bの中にあるpowerという変数を指定する場合は以下の様に指定する。

```
a::b::power
```

コード例

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

namespace a {
    void function() { cout << "KHEC_1" << endl; };
    int number = 1;
    namespace b {
        void function() { cout << "KHEC_2" << endl; };
        int number = 2;
    }
}

void main() {
    a::function();
    cout << a::number << endl;
    a::b::function();  // ネストされた名前空間の関数functionを呼び出す。
    cout << a::b::number << endl;  // ネストされた名前空間の変数numberを表示する。
}
```

出力

```
KHEC_1
1
KHEC_2
2
```

問題

名前空間a内にint型の配列arrayを入れ、その値を{1, 2, 3, 4, 5}として下さい。また、名前空間a内に名前空間bをネストし、その中にarrayという名前のvectorを入れてください。

まず、名前空間a内の配列arrayの値を順番に表示して下さい。そして、名前空間a内の配列arrayを名前空間b内のarrayという名前のvectorにコピーし、vectorの値を順番に表示して下さい。

出力例

```
int型配列>
{1, 2, 3, 4, 5}
vector>
{1, 2, 3, 4, 5}
```

ヒント: 配列の値の表示方法はvectorの問題と同様。

2.4 標準名前空間

以前、以下の様にして入力した内容をそのまま返すプログラムを書きました。

```
#include <iostream>

using namespace std;

int main(){
    int a;
    cin >> a;
    cout << "a=" << a << endl;
    return 0;
}
```

cinやcout、endlを使うには「using namespace std;」が必要でした。これは、stdという名前空間の中にcinやcout、endlといったものがあるからです。しかし、stdという名前空間を私たちが作る必要は無く、それらは元からC++の機能として入っているということです。このstdという名前空間を「標準名前空間」といいます。

これもまた名前空間なので以下の様にusing指令ではなくスコープ解決演算子を用いて書くことも出来ます。

```
#include <iostream>
int main(){
    int a;
    std::cin >> a;
    std::cout << "a=" << a << std::endl;
    return 0;
}
```

このプログラムにおいて、cinは入力を、coutは出力を司るモノ(オブジェクト)です。変数や関数ではありません。とりあえずcinは入力された内容が入っているもの、coutは出力画面と思っておくといいと思います。endlは\nの様に改行を意味する記号です。

そして、<<や>>は「ストリーム」といい、以下の様に開いている方から閉じている方にデータ(数値や文字列を送るという意味があります。

```
cin >> a;  // 入力した内容 --送る--> 変数a

cout << "a=" << a << endl;  // 出力画面 <--送る-- 変数a <--送る--改行
```

※問題はありません