rigibun / **kyoupro_on_cpp.md**

Last active 23 days ago

kyoupro_on_cpp.md

# C++初心者がC++を使って競技プログラミングするための備忘録のようなもの

この記事は、C++ (fork) Advent Calendar 2013の12日目の記事です。

## はじめに

記事を書く人が居ないみたいなので、C++初心者ですが箸休め的な記事を書こうと思い立ち、いざ書き上げてみたら思いの外長くなりました。

この記事は、C++初心者な著者が、C++を用いて競技プログラミングをするために、調べたことや試した事などのまとめです。 記事中に誤り、問題点やご指摘、ご質問等ありましたら、@rigibunまでご連絡下さい(特にpush_bach)

githubのmarkdownを使いたかったことと、変更履歴が見られることからgistで書きました。

## 免責事項

著者はこの記事を利用することによって生じたいかなる損害等に対して責任を負いません。

この記事は競技プログラミングに必要な部分に特化して書かれているので、全編を通していわゆる"BetterC"なコードになっていることをご了承下さい。

競技プログラミングにおいてはC++11を使えないコンテストもあるので、この記事はC++03を前提として書かれています。全てのコンテストがC++11で書けるようになってほしい。

この記事では、C++におけるプログラミングにおいて、一般的に推奨されない記述がありますので、一般的なC++プログラミングの作法等に関しては別な記事をご覧ください。また、推奨されない記述が何の注釈も無しに書かれている点があればご指摘下さい。

この記事には、C++-erにとっては当たり前の事しか書かれておらず、既にC++を活用されている方にとっては、新たな発見は無いと思います。

## 基本

C++はmain関数がエントリーポイントです。TopCoderなどのクラスとメソッドを実装する形式のコンテスト を除けば、基本的にどのコンテストでもmain関数を書くことになるでしょう。まずは基本のHello, Worldから

```
#include <iostream>
using namespace std;
int main() {
        cout << "Hello, World" << endl;
        return 0;
}
```

`#include` によってライブラリを利用するためにヘッダファイルをインクルードします。この場合は `cout` と `endl` を利用するために `#include <iostream>` としてiostreamをインクルードします。

続く `using namespace std;` について。C++は名前空間を持っており、基本的に標準ライブラリはstd名前空間に含まれます。 `cout` や `endl` もこの名前空間に含まれるので、本来は `std::cout << "Hello, World" << std::endl;` と書かなければなりません。競技プログラミングではこのように記述するのは冗長で時間の無駄なので、 `using namespace std;` と書くことによってこれを避けています。

ここで注意すべきなのは、一般的にC++のプログラミングにおいて、このように `using namespace std;` と書くことは避けるべきである。という点です。この辺りの議論に関してはgoogle先生などに任せます。

次に、main関数です。C++では関数は **返り値 関数名(引数) {手続き}** という形で関数を定義します。 mainの返り値はint型とします。

また、main関数に関しては、 `return 0` の記述は必要ないそうです。

## I/O編

それでは、早速C++で簡単な問題を解いて見ましょう。 AOJ 10001 X Cubic

入出力が出来れば何の問題もなく解ける問題です。

```
#include <iostream>
using namespace std;
int main() {
        int x;
        cin >> x;
        cout << x * x * x << endl;
}
```

普通に書けばこのようになると思いますが、入出力に関しては、幾つか問題があります。

### iostreamが遅い

iostream(cin,cout,endl)などは遅い、というのが競技プログラミング界隈での常識のようです。

### 出力指定が面倒

coutを使うとどうにも出力指定が面倒です。(出力指定に関しては、C++ の iostream フォーマット指定早見表などを参照して下さい。)

### 解決策

一つ目には、素直にstdioを使うというのがあります。

```
#include <cstdio>
using namespace std;
int main() {
        int x;
        scanf("%d", &x);
        printf("%d\n", x * x * x);
}
```

C++でもC言語のstdioを使用することができます。

もうひとつは、cinやcoutを遅くしている原因を取り除くことです。

```
#include <iostream>
using namespace std;
int main() {
        cin.tie(0);
        ios::sync_with_stdio(false);

        int x;
        cin >> x;
        cout << x * x * x << "\n";
}
```

`cin.tie(0);` ではcinとcoutの結び付きを解除しているようです。というのも、cinは呼び出しの度にcoutをフラッシュしているらしいです。

`ios::sync_with_stdio(false);` では、stdioとの同期を解除しています。

おまじないのような二行ですが、これを書くだけで、入力が50万行あるようなケースでは速度が見違えるほど速くなります。

また、 `cout << x * x * x << "\n";` のように、endlではなく改行文字 `\n` を書いています。endlは改行と同時にバッファを全て吐き出すために遅くなるようです。これも出力が多い時に速度差が出るようです。

## 基本データ型

ここでは競技プログラミングでよく使う型を紹介します。

### int

まずはお馴染みのint型。整数を格納するために使います。恐らく32bitと捉えていいでしょう。

### short, long

同じく整数型。shortは16bit、longは64bitと想定して問題ないでしょう。

### double, float

浮動小数点数です。floatは32bit、doubleは64bitと捉えていいと思います。基本的にはfloatを使うことは少なく、大抵doubleを使って書きます。

### char

文字型です。競プロではマルチバイト文字を使用することはほぼないので、ASCII文字を格納するために使います。一応数値数値型として利用することも出来ますが……

### bool

忘れては行けないbool型。true, falseを格納するために使います。フラグなどにはbool型を使いましょう。

## 配列とコンテナ、その他の型

C++では、そのまま競技プログラミングに使える便利なデータ構造が標準で用意されています。

### 配列

まずはC++の配列を見てみましょう。

```
int main() {
        int array[10];
        for(int i = 0; i < 10; i++)
                array[i] = i * i;
}
```

この辺りは他の言語と比べて大きな違いはない部分でしょうか。

```
#include <iostream>
int main() {
        int n;
        std::cin >> n;
        int arr[n];
}
```

コンパイラの独自拡張で、動的に確保出来たりします。コンテストで使用する処理系によっては動かないこともあります。AOJでは動きました。

### vector

可変長配列です。データの大きさに合わせて配列のサイズが変わります。基本的には配列でも問題なく書けるんですが、vectorを使う時もあります。

```
#include <vector> // vectorを使うためにインクルードする
#include <iostream>
using namespace std;
int main() {
        vector<int> vc; // <T>でT型のvector
        int n;
        cin >> n;
        while(n--) {
                int x;
                cin >> x;
                vc.push_back(x);
        }

        n = vc.size() / 2;
        if(vc[n] == vc.at(n))
                cout << n << '\n';
}
```

`push_back()` で配列の最後に要素を追加します。 `size()` はvectorの大きさを返してくれます。

また、vectorの要素へのアクセスは、配列のように `[]` と書く方法と、 `at()` を使う方法があります。後者は範囲外のインデックスを与えると例外を出すため前者より安全です。が、競プロでは安全とか考えずに前者を使います。

### list

双方向リストです。要素の挿入、削除が多い時に。

### set

集合です。

### map

連想配列です。キーと値を関連付けて保持します。

```
#include <map>
#include <iostream>
using namespace std;
int main() {
        map<char, int> mp; // <key, value> と書くとkey型をキーとし、value型を値とする。
        mp['a'] = 1;
        mp['b'] = 2;
        cout << mp['a'] << ' ' << mp['b'] << '\n';

        mp.insert(map<char, int>::value_type('c', 3)); // 長い
        if(mp.find('c') != mp.end())
                cout << mp['c'] << '\n';
}
```

`[]` 演算子が使えるので使います。findはイテレータ(後述)を返します。

setとmapは、内部的には木構造で実装されているようで、値は順序付けられて格納されています。

### stack, queue, priority_queue

スタック、キュー、優先度付きキューです。いずれも競プロでは頻出です。stackは、まさしくスタックそのものをシミュレートするような問題にそのまま使えます。 queueは、幅優先探索などをするときに利用することが多いです。priority_queueは貪欲なアルゴリズムを実装する場合などに使います。

### string

文字列型です。

```
#include <string>
#include <iostream>
```

```
using namespace std;
int main() {
        string str;
        str = "this is a string";       // 代入できます
        str[2] = 'a';   // []演算子でアクセスできます。
        str[3] = 't';
        if(str.length() >= 5) // lengthで長さを取得できます
                cout << str << '\n';
        if(str.find("is") != string::npos)      // 文字列を検索できます。見つかればインデックス、見つからなければstring::r
                cout << str.find("is") << '\n';
}
```

char[]でも出来ないことはないんですが、stringを使うほうが便利なのでstringを使います。

### pair

二つの値を一組にします。

```
#include <utility>      // pairはutilityヘッダ
#include <vector>
#include <iostream>
using namespace std;
int main() {
        vector< pair<int, int> > vc;
        int n;
        cin >> n;
        while(n--) {
                int x, y;
                cin >> x >> y;
                vc.push_back(pair<int, int>(x, y));
        }
        pair<int, int> p = vc.front();
        cout << p.first << ' ' << p.second << '\n';     // 各要素はfirst, secondでアクセス可能
}
```

### ※イテレータ is 何

コンテナの全要素に順番にアクセスする際に使います。

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
        int n;
        cin >> n;
        vector<int> vc;
        while(n--) {
                int x;
                cin >> x;
                vc.push_back(x);
        }
        // vector<T>のイテレータはvector<T>::iterator型
        vector<int>::iterator it = vc.begin();  // begin()はvectorの最初の要素を指すイテレータを返す。
        vector<int>::iterator endIt = vc.end(); // end()はvectorの終端位置を指すイテレータを返す。
        while(it != endIt) {
                cout << *it << '\n';    // イテレータの指す要素は、ポインタのように*演算子でアクセスします。
                it++;   // インクリメントすることで次の要素を指します。
        }
}
```

このようにして、イテレータを持つコンテナの要素に順にアクセスできます。vectorの場合、逆順にアクセスするには、`rbegin()` と `rend()` が提供されています。 イテレータには幾つか種類があります。詳しくはgoogle先生などに聞いてみるといいでしょう。

## その他のSTL

STLにはコンテナ以外にも競技プログラミングで有用なものが多くあります。

### 値の交換をいちいち書くの面倒

→swapを使いましょう。

```
#include <algorithm>    // swapはalgorithmヘッダ
using namespace std;
int main() {
        int a = 1;
        int b = 5;
        swap(a, b);     // -> a = 5, b = 1
}
```

## ソート書くの面倒・クイックソート書けない

→ `sort()` 、あります。競プロではものすごく使います。

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main() {
        int n;
        cin >> n;
        vector<int> vc;
        while(n--) {
                int x;
                cin >> x;
                vc.push_back(x);
        }

        sort(vc.begin(), vc.end());     // sortにはソートする範囲のイテレータを渡す

        for(int i = 0; i < vc.size(); i++)
                cout << vc[i] << endl;
}
```

`sort()` は配列にも適用可能です。その場合、

```
int arr[N];
sort(arr, arr + N);     // 先頭のポインタ, 先頭のポインタ+要素数
```

でソートします。ソートはデフォルトで昇順です。降順にソートするには、

```
#include <algorithm>
#include <functional>   // greater<int> はfunctionalヘッダ
using namespace std;
int main() {
        int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
        sort(arr, arr+9, greater<int>());
}
```

のように書きます。functionalヘッダにはこの他にも、STLに渡すための関数が幾つか定義されています。

また、vectorの場合は単に `sort(vc.rbegin(), vc.rend())` のように書くこともできます。

## 大きい値・小さい値を代入したい

→min, maxあります。

```
#include <algorithm>
using namespace std;
int main() {
        int a = max(1, 5);      // 5
        int b = min(1, 5);      // 1
}
```

DP(動的計画法)で解く問題をはじめとして、min, maxは頻出です。

## 二分探索ってどう書くの?

→あります。C++には何でもあります。

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main(void) {
        int n;
        cin >> n;
        vector<int> vc;
        while(n--) {
                int x;
                cin >> x;
                vc.push_back(x);
        }
        sort(vc.begin(), vc.end());     // 二分探索なので必ずソートしておく

        int x;
        cin >> x;
        if(binary_search(vc.begin(), vc.end(), x))      //binary_searchはtrue, falseを返す
                cout << "I found " << x << '\n';
        vector<int>::iterator lit = lower_bound(vc.begin(), vc.end(), x);       // lower_boundは指定された値"以
        vector<int>::iterator uit = upper_bound(vc.begin(), vc.end(), x);       // upper_boundは指定された値を
        if(lit != vc.end() && lit != uit)
                cout << uit - lit << '\n';      // この場合, ult - lit は vc 中の x の個数になる
}
```

`lower_bound()` と `upper_bound()` を組み合わせると配列内で特定の値が存在する範囲を得ることができます。

## その他

### (テンプレート等)変数の宣言が長い

typedefしましょう。

```
#include <utility>
#include <vector>
#include <iostream>
using namespace std;
typedef unsigned long ul;
typedef pair<ul, ul> P;

int main() {
        vector<P> vc;
        int n;
        cin >> n;
        while(n--) {
                ul x, y;
                cin >> x >> y;
                vc.push_back(P(x, y));
        }
}
```

### for文が長い

defineマクロ使いましょう

```
#include <iostream>
#define REP(i,n) for(int i=0;i<n;i++)
using namespace std;
int main() {
        REP(i, 10)
                cout << i << '\n';
}
```

## まとめ

競技プログラミングでC++を使う利点として、単に他の言語(スクリプト言語、C#やJava)より高速という点の他に、ライブラリに競プロの道具が揃っているという点があります。 C++の機能を使いこなし、コンテスト中の脳のリソースを最大限問題に割けるようにしましょう！

## 参考文献

Wikipedia

C/C++ リファレンス

競技プログラミングで使えそうなSTLまとめ (Competitive Programming Advent Calendar) - プログラミング関係のメモとか

競技プログラミングのためのC++入門

paiza - 新人女子の書いたコードを直したけど架空の新人女子だったことに気づいた - Qiita [キータ]

PaizaとはI/Oゲーである - 名古屋３１３の日記