# UoA CP Lecture 1&2
# Shifting to C++(or C with STL)

Qingchuan(Sam) Zhang

April 10, 2020

## Contents

## 1 Background

In competitive programming, no expertise in C++/Java/Python is required. We only need to a small portion of the language(s) to convert our ideas to codes. Here I'll introduce the basic stuff needed for CP. Though there are more advanced stuff might be useful, the ones here can fulfill our needs for now.

The documentation of C++ is on cppreference. Use the docs to find things you don't know how to use.

Let's start!

## 2 Basic Framework

```cpp
#include <iostream>
#include <string>
#include <algorithm>
#include <vector>
#include <queue>
// or #include<bits/stdc++.h> if supported
using namespace std;

int main() {

    return 0;
}
```

**`#include`** This is like `import` in Python, you need to include the libraries before using built-in functionalities.

**`using namespace std`** This line means you want to use the functions/containers residing in the `std` namespace as if they were written by yourself. If you don't put this line, you will need to add `std::` before the names. For example, `sort` versus `std::sort`

**`main`** This is where your program starts to run. The return type should always be `int` and it always return 0.

## 3 Basic Data Types

- `bool`: either `true` or `false`.
- `char`: an ASCII character, usually used to store `'a'-'z'` or `'A'-'Z'` or `'0'-'9'`.
- `int`: integers range from $-2^{31}$ to $2^{31}-1$, or just memorize $\pm 10^9$
- `long long`: integers range from $-2^{63}$ to $2^{63}-1$, or just memorize $\pm 10^{18}$
- `double`: decimal numbers of 15 digit precision.
- `std::string`: stores a sequence of characters like `"abcdefg"` or`"from1to2"`

## 4 Input/Output

### 4.1 `C++` style(recommended)

```
#include<iostream>
using namespace std;
int main(){
    int a;
    long long b;
    double c;
    cin >> a >> b >>c;
    cout << a << " " << b << " " << c << endl;//endl = end of line
    return 0;
}
```

### 4.2 `C` style(not recommended)

```
#include<cstdio>
int main(){
    int a;
    long long b;
    double c;
    scanf("%d %lld %lf",&a,&b,&c);
    printf("%d %lld %f",a,b,c);
    return 0;
}
```

## 5 if/for/while

The following three structures all the most common ones used by almost every single c++ program.

### 5.1 if

```cpp
//...
int main(){
    int value;
    cin>>value;
    if(value == 0){
        cout<<"the input is zero"<<endl;
    }else{
        cout<<"the input is not zero"<<endl;
    }
    return 0;
}
```

### 5.2 for

```cpp
//...
int main(){
    int n;
    cin>>n;
    int sum = 0;
    for(int i = 1;i<=n;i++){
        sum = sum + i;
        //sum += n;
    }
    cout<<"The sum from"<<1<<" to n is "<<sum<<endl;
    return 0;
}
```

### 5.3 while

```cpp
//...
int main(){
    int value;
    cin>>value;
    int step = 0;
    //collatz conjecture
    while(value != 1){
        if(value % 2 == 0){
            value /= 2;
        }else{
            value = value * 3 + 1;
        }
        step++;
        //step += 1;
    }
    cout<<"The number of steps is "<<step<<endl;
    return 0;
}
```

# 6 Array

## 6.1 std::vector<Type>

```cpp
//..
#include<vector>
int main(){
    vector<int> a(5,1);//{1,1,1,1,1}
    a.insert(a.begin()+1,2);//{1,2,1,1,1,1}
    //0-indexed
    for(int i = 1;i<a.size();i++){
        a[i] = a[i-1];
    }
    int last = a.back();
    bool is_empty = a.empty();


    return 0;
}
```

## 6.2 Type a[]

```cpp
//...

int a[100]; //{0,0,0,...}
const int maxn = 200;
int b[maxn];
int n;
int c[n];//error
int main(){
    int d[100];// random values
    int value = d[50];
    cout<<value<<endl;
    return 0;
}
```

## 7 Example Problem: Increasing Array

```cpp
#include<bits/stdc++.h>
using namespace std;
const int MAXN=200000+5;
int n;
int nums[MAXN];

int main()
{
    cin>>n;
    for(int i=0; i<n; i++)
        cin>>nums[i];
    long long ans=0;
    for(int i=1; i<n; i++)
    {
        if(nums[i]>=nums[i-1])
            continue;
        else
        {
            ans+=nums[i-1]-nums[i];
            nums[i]=nums[i-1];
        }
    }
    cout<<ans<<endl;
    return 0;
}
```

# 8 String

## 8.1 std::string

```
//..
#include<string>
int main(){
    string a = "abcdefg";// set to literal value
    string b; // set to ""
    string c;
    cin>>c; // input
    b = "xyz";
    c = c + a + b;// concatenation
    cout<<c<<" "<<c[0]<<endl;
    return 0;
}
```

## 8.2 char s[]

```
//...
int main(){
    char s[5] = "abcd";//"abcd\0"
    char a[10];
    scanf("%s",a);
    printf("%s",a);
    return 0;
}
```

## 9 Struct keyword

### 9.1 Definition

```cpp
struct Point{
    int x,y;
    Point(int xx,int yy){
        x = xx;
        y = yy;
    }
    double disFromOrigin(){
        return sqrt(x*x + y*y);
    }
};

int main(){
    Point p(10,20);
    cout<< p.disFromOrigin() << endl;
    return 0;
}
```

## 9.2 Example: Codeforces 479 C

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MAXN = 5000 + 10;
int n;
struct exam_time
{
    int atime, btime;
    exam_time(){};
    exam_time(int a, int b)
    {
        this->atime = a;
        this->btime = b;
    }
    bool operator<(const exam_time &exam)
    {
        return atime < exam.atime || (atime == exam.atime && btime < exam.btime);
    }
} exam[MAXN];
int main()
{
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        int x, y;
        cin >> x >> y;
        exam[i] = exam_time(x, y);
    }
    sort(exam, exam + n);
    int cur = 0;
    for (int i = 0; i < n; i++)
    {
        if (exam[i].btime < cur)
        {
            cur = exam[i].atime;
        }
        else
        {
            cur = exam[i].btime;
        }
    }
    cout << cur << endl;
    return 0;
}
```

## 10 Other Containers

Most, if not all, containers in the Standard Template Library(STL) support `.size()` and `.empty()` operations. The containers listed below are just like the structs you can define which support various operations. The advantage of using them is that they are reliable and encapsulated.

### 10.1 `std::queue`

The data structure *Queue* is a first-in-first-out (FIFO) structure. The three core operations are: `push`,`pop`,`front`.

```
//..
#include<queue>
int main(){
    std::queue<int>q;//{}
    q.push(1);//{1}
    q.push(2);//{1,2}
    q.push(3);//{1,2,3}
    q.front();// 1
    q.pop();//{2,3}
    q.front();// 2
    q.pop();//{3}
    q.pop();//{}
    q.pop();//Error: cannot pop an empty queue
    return 0;
}
```

### 10.2 `std::stack`

The data structure *Stack* is a first-in-last-out (FILO) structure. The three core operations are: `push`,`pop`,`top`.

```
//..
#include<stack>
int main(){
    std::stack<int>s;//{}
    s.push(1);//{1}
    s.push(2);//{1,2}
    s.push(3);//{1,2,3}
    s.top();// 3
    s.pop();//{1,2}
    s.top();// 2
    s.pop();//{1}
    s.pop();//{}
    return 0;
}
```

## 10.3 std::priority_queue

The data structure *Priority Queue* is a first-in-highest-priority-out (FILO) structure. The three core operations are: `push`,`pop`,`top`.

```cpp
//..
#include<queue>
int main(){
    std::priority_queue<int>pq;//{}
    pq.push(1);//{1}
    pq.push(3);//{1,2}
    pq.push(2);//{1,2,3}
    int t = pq.top();//3
    pq.pop();//{1,2}

    //Want a priority queue that pops the smallest element? There are two ways:
    //    1. Make everything negative
    //    2. priority_queue<T,vector<T>,greater<T>> pq; //just memorize the formula, you do

    priority_queue<int,vector<int>,greater<int> >small_pq;
    small_pq.push(1);
    small_pq.push(3);
    small_pq.push(2);
    int t = small_pq.top();//1


    return 0;
}
```

## 10.4 std::set<Type>

A *Set* is used to store a set of elements which can be **ordered**, e.g. `int`. Since it's a set, inserting a same element multiple times would only make one copy in it. The most common operations on a set are: `insert`,`erase`,`count`,`lower_bound`.

```cpp
//..
#include<set>
int main(){
    std::set<int>s;//{}
    s.insert(1);
    s.insert(5);
    s.insert(3);
    s.insert(7);//{1,3,5,7}
    int cnt;
    cnt = s.count(1);//1
    cnt = s.count(2);//0

    //lower_bound returns the address/iterator pointing to
    //the first element large or equal to a given element
    //
    //upper_bound on the other hand returns the iterator of
    //the first strictly larger one.
    set<int>::iterator nxt_ptr = s.lower_bound(3);//returns an iterator pointing to 3
    nxt_ptr = s.lower_bound(4);//pointing to 5 now
    return 0;
}
```

## 10.5 std::map<Key,Value>

A *Map* is used to store a bunch of key-value pairs. It can be thought of as an array but the range of the index could be much larger. For example, `map<int,string>` can have any integer as the index/key.

```
//..
#include<map>
int main(){
    std::map<int,string>mp;
    mp[1] = "Hello";
    mp[1919] = "Bye";
    mp[100000000] = "Nice day";
    cout<<mp[1919]<<endl;// "Bye"
    cout<<mp[2020]<<endl;// "", the default value is "" for string and 0 for numbers.

    return 0;
}
```

## 11 Function

Defining a function and calling it can save your time a lot. When you have to run same procedure on different inputs, you wouldn't want to copy the same code multiple times, which is unorganized, likely to cause problems and hard to modify. Instead, we can define a function to suit our needs.

```
/*
ReturnType FuncName(Parameter1Type p1, Parameter2Type p2,...){
    //do something
}
*/
int sum(int n){
    int result = 0;
    for(int i = 1;i<=n;i++){
        result += i;
    }
    return result;
}

int main(){
    int s1 = 0;for(int i = 1;i<=1;i++)s1+=i;
    int s2 = 0;for(int i = 1;i<=2;i++)s2+=i;
    int s3 = 0;for(int i = 1;i<=3;i++)s3+=i;

    sum(1);
    sum(2);
    sum(3);
    return 0;
}
```

### 11.1 Pointer

A pointer is just the address of some object/variable. When the object we want to pass into the function is very large and we don't want to copy the entire thing, we can use pointer.

```
struct LargeArray{
    int a[10000];
};

LargeArray slow_update(LargeArray arr,int index,int delta){
    arr.a[index] += delta;
}

void fast_update(LargeArray* arr,int index,int delta){
    (*arr).a[index] += delta;
    //or
    (*arr)->a[index] += delta;
}



int main(){
    //pointer of int
    {
        int x = 0;
        int* ptr = &x;//int* means it's a pointer pointing to an int
        (*ptr) += 1
    }
    //pointer of a container
    {
        vector<int>v;
        vector<int>*ptr = &v;
        (*ptr).push_back(1);
        ptr->push_back(2);
    }
    //pointer of array
    {
        int a[10];
        int* ptr = a;
        int* ptr2 = &a[1];
        int* ptr3 = ptr+1; //ptr2 == ptr3
    }

    LargeArray arr;
    arr = slow_update(arr,1,100);

    fast_update(&arr,1,100);
    return 0;
}
```

## 11.2 Reference

Some times we would like to create an alias of some existing variable, *reference* should be used. This is usually clearer then pointer, although pointer is the more powerful one.

```
struct LargeArray{
    int a[10000];
};

LargeArray slow_update(LargeArray arr,int index,int delta){
```

```cpp
    arr.a[index] += delta;
}

void fast_update(LargeArray* arr,int index,int delta){
    (*arr).a[index] += delta;
    //or
    (*arr)->a[index] += delta;
}
void reference_update(LargeArray& arr,int index,int delta){
    arr.a[index] += delta;
}



int main(){
    {
        int a = 1;
        int& b = a;
        //b and a are completely same
        //apart from having a different name/identifer.
        b++;//now a = b = 2
    }

    LargeArray arr;
    reference_update(arr,1,2);

    return 0;
}
```

# 12 Useful Library Functions

## 12.1 std::max/min

```cpp
#include<bits/stdc++.h>
using namespace std;

int main(){
    int a = 1,b = 2;
    int c = max(a,b);//2
    int d = min(a,b);//1
    int e = max(a,b,c)//error: two inputs only
    int f = max({a,b,c})// works on newer standards
    int g = max(1,1ll);//error: has to be exactly the same type: int != long long

    return 0;
}
```

## 12.2 std::sort

```cpp
#include<bits/stdc++.h>
using namespace std;

int main(){
    int arr[] = {1,4,5,3,2};
    sort(arr,arr+5);//{1,2,3,4,5}
    sort(arr,arr+5,greater<int>());//{5,4,3,2,1}
    //or
    reverse(arr,arr+5);


    vector<int>v({1,4,5,3,2});
    sort(begin(v),end(v)); // or sort(v.begin(),v.end());

    return 0;
}
```

## 12.3 std::lower_bound/upper_bound

```cpp
#include<bits/stdc++.h>
using namespace std;

int main(){
    int a[] = {1,3,7,5,9};
    sort(a,a+5);//{1,3,5,7,9}
    int* ptr = lower_bound(a,a+5,2);// &a[1]
    ptr = lower_bound(a,a+5,3);//still &a[1]
    ptr = lower_bound(a,a+5,4);//&a[2]
    ptr = lower_bound(a,a+5,100000);// a+5

    ptr = upper_bound(a,a+5,2);// &a[1]
    ptr = upper_bound(a,a+5,3);// &a[2]
    ptr = upper_bound(a,a+5,10000);// a+5
    return 0;
}
```

## 13 Other useful stuff

### 13.1 Scope

```cpp
#include<bits/stdc++.h>
using namespace std;

int main(){
    vector<int>v({1,3,4,2});
    {
        //calc min
        int ans = INT_MAX;
        for(int i = 0;i<v.size();i++){
            ans = min(ans,v[i]);
        }
        cout<<ans<<endl;
    }
    {
        //calc max
        int ans = INT_MIN;
        for(int i = 0;i<v.size();i++){
            ans = max(ans,v[i]);
        }
        cout<<ans<<endl;
    }

    double result = 2;
    {
        double tmp = result + 1;
        result = tmp * tmp;
    }
    return 0;
}
```

### 13.2 range-based for loop

```cpp
#include<bits/stdc++.h>
using namespace std;

int main(){
    vector<int>v({1,3,4,2});
    int sum = 0;
    fot(int i:v)sum += i;

    //reference
    for(int i:v)i+=1;//doesn't work
    for(int &i:v)i+=1;
    return 0;
}
```

## 13.3 auto

```cpp
#include<bits/stdc++.h>
using namespace std;

int main(){
    vector <int> v({1,2,3,4});
    auto ptr1 = lower_bound(v.begin(),v.end(),3);
    vector<int>::iterator ptr2 = lower_bound(v.begin(),v.end(),3);
    return 0;
}
```

## 13.4 const

```cpp
#include<bits/stdc++.h>
using namespace std;


const int mod = 10007;// recommended
//#define mod 10007 //not recommended

const int N = 100000,Q = 1000;
int a[N], op[N], ans[Q];
int main(){
    int a = rand();
    int b = a % 10007;//bad
    int c = a % mod;
    return 0;
}
```

## 13.5 namespace

```cpp
#include<bits/stdc++.h>
using namespace std;

namespace A{
    int foo(int a,int b){
        return a + b;
    }
}
namespace B{
    int foo(int a,int b){
        return a - b;
    }
}

int main(){
    int a = A::foo(1,2);
    int b = B::foo(1,2);
    return 0;
}
```

### 13.6 pair

```cpp
#include<bits/stdc++.h>
using namespace std;

int main(){
    int a = 1;
    double b = 2;
    pair<int,double> c = make_pair(a,b);
    int d = c.first;
    double f = c.second;


    map<int,string> mp;
    mp[1] = "A";
    mp[3] = "B";
    mp[11111] = "C";
    for(auto pr:mp){
        cout<<pr.first<<":"<<pr.second<<endl;
    }
    return 0;
}
```