WHUST 2017 Div.2 Day 3.5
C++ 标准模版库

WHU ACM / ICPC 集训队
郭松 <gs199704@gmail.com>

July 20, 2017

算法竞赛中，需要用到的 C++ 知识

- 标准命名空间 `std`.

算法竞赛中，需要用到的 C++ 知识

- 标准命名空间 `std`.
- 模版函数.

算法竞赛中，需要用到的 C++ 知识

- 标准命名空间 `std`.
- 模版函数.
- 模版类.

命名空间

- C++ 中引入了命名空间的概念.

命名空间

- C++ 中引入了命名空间的概念.
- C++ 的标准函数都在标准命名空间 `std` 中.

命名空间

- C++ 中引入了命名空间的概念.
- C++ 的标准函数都在标准命名空间 `std` 中.
- 对命名空间的了解只需要做到知道有这东西即可.

命名空间

- C++ 中引入了命名空间的概念.
- C++ 的标准函数都在标准命名空间 `std` 中.
- 对命名空间的了解只需要做到知道有这东西即可.
- 在 `#include` 语句之后加上 `using namespace std;` 即可

代码框架

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  int main()
4  {
5      return 0;
6  }
```

代码解释

- 在 `bits/stdc++.h` 头文件包含了所有的头文件，在算法竞赛中你可以随意使用，但请不要在算法竞赛之外的场合使用

代码解释

- 在 `bits/stdc++.h` 头文件包含了所有的头文件，在算法竞赛中你可以随意使用，但请不要在算法竞赛之外的场合使用
- 在算法竞赛中，我们推荐使用 `scanf`, `printf` 作为输入输出语句，它们有更高的效率

代码解释

- 在 `bits/stdc++.h` 头文件包含了所有的头文件，在算法竞赛中你可以随意使用，但请不要在算法竞赛之外的场合使用
- 在算法竞赛中，我们推荐使用 `scanf`, `printf` 作为输入输出语句，它们有更高的效率
- C++11 标准新增了很多新的语法，在这里暂时不介绍，在后面遇到的时候结合实际的代码介绍

C++ 代码中引入了实参的概念，这意味着你可以不用那么多指针。

比如你可以这样在函数中交换两个外部变量

```
1  void swap (int& a, int& b)
2  {
3      int t;
4      t = a; a = b; b = t;
5  }
6
7  int main()
8  {
9      int a = 3, b = 4;
10     swap(a, b); // a = 4, b = 3
11 }
```

使用方法很简单，在想使用实参的参数前面加上 & 即可

C++11 标准中，引入了 auto 类型，用来让编译器自动猜测数据类型，在不方便写数据类型，或数据类型很长的时候很管用。

```
1 auto a = 1; // a is an "int"
2 auto b = a; // b is an "int"
3 auto c;     // 错误，无法猜测类型
```

注意

auto 类型猜测规则可能与你想的不一样，请在可以完全确定变量的数据类型的时候使用它

模版函数

我们使用 C++ 的一大原因就是其提供了丰富的模版函数，这些函数大多包含于 algorithm 头文件中，如果要在别的地方使用，只需要 `#include<algorithm>`
C++ 的模版函数很多，我们只介绍算法竞赛中常用的几种

swap

交换两个元素

定义

```
1 void swap (T& a, T& b);
```

注意

参数表里的 T 表示任意类型，同一个字母表示同一种类型
即这里要求两个元素类型相同

用法

```
1  int a = 1, b = 2;
2  swap(a, b); // a = 2, b = 1
3
4  int c = 1; double d = 2.;
5  swap(c, d); // 编译错误, c d 类型不同
6
7  int e = 1;
8  swap(e, 2); // 编译错误, 常数不能转为实参类型
```

sort

对数组／ vector 进行排序，效率为 $O(n \log n)$

定义

```
1 void sort (RandomAccessIterator first,
  ↪  RandomAccessIterator last);
2 void sort (RandomAccessIterator first,
  ↪  RandomAccessIterator last, Compare comp);
```

注意

RandomAccessIterator 随机访问迭代器，你可以理解为指针
Compare 比较器，你可以理解为函数指针
sort 如果不定义比较器，则按照从小到大的顺序排序
sort 的排序范围是左闭右开的

用法 1

```
int a[10] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};
sort(&a[0], &a[10]);
// a: 0 1 2 3 4 5 6 7 8 9
```

用法 2

```
bool cmp(int a, int b) {
    return a > b;
}

int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
sort(&a[0], &a[10], cmp);
// a: 9 8 7 6 5 4 3 2 1 0
```

min / max

返回二者中较大的一个

定义

```
1 T& min (const T& a, const T& b);
2 T& max (const T& a, const T& b);
```

用法

```
1 cout << min(1,2) << '\n'; // 1
2 cout << min(2,1) << '\n'; // 1
3 cout << min('a','z') << '\n'; // 'a'
4 cout << min(3.14,2.72) << '\n'; // 2.72
5
6 min(1, 2ll); // 编译错误，类型不符
```

模版类

我们使用 C++ 的另一大原因即是其提供了大量的基本数据结构，对于解决简单的题目它们十分有用，这些数据结构有变长数组、队列、栈、堆、集合、映射等等

map - 映射

map 定义了这样的一种操作，即储存键-值对，并支持在 $\log(n)$ 时间内返回给定的键对应的值的功能

声明

```
map< 键类型，值类型 > XXX;
```

使用 1

```
1  map<char, string> mymap;
2
3  mymap['a']="AAA";
4  mymap['b']="BBB";
5  mymap['c']=mymap['b'];
6
7  cout << mymap['a'] << '\n'; // AAA
8  cout << mymap['b'] << '\n'; // BBB
9  cout << mymap['c'] << '\n'; // BBB
10 cout << mymap['d'] << '\n'; //  (空行)
```

使用 2

```
map<char, string> mymap;

mymap['a']="AAA";

map<char, string>::iterator x = mymap.find('a');
cout << x->first << ' ' << x->second << '\n'; // a AAA

map<char, string>::iterator y = mymap.find('b');
cout << (y == mymay.end()) << '\n'; // 1
```

注意

这样查找，如碰到空元素不会创建

queue - 队列

queue 提供了与基本数据结构的队列功能一致的队列。

声明

```
queue< 值类型 > XXX;
```

使用

```
queue<int> myqueue;

myqueue.push(5);
myqueue.push(6);

myqueue.front(); // 5
myqueue.size();  // 2

while (!myqueue.empty()) {
    cout << myqueue.front() << ' ';
    myqueue.pop();
}

// 5 6
```

set - 集合

set 提供了与数学中的集合功能一致的集合，自带排序与去重。

声明

```
set< 值类型 > XXX;
```

遍历

```
1  set<int> myset;
2  for (int i=5; i >= 1; i--)
3      myset.insert(i*10);
4
5  for (set<int>::iterator it = myset.begin();
6       it != myset.end();
7       it++)
8      cout << *it << " ";
9
10 // 10 20 30 40 50
```

时间复杂度

遍历整个集合的复杂度是 $O(n)$

查找删除

```
1  // 假设有 set, 内容为 10 20 30 40 50
2
3  set<int>::iterator it;
4
5  it = set.lower_bound(20); // 20
6  it = set.lower_bound(21); // 30
7  it = set.upper_bound(20); // 30
8
9  it = set.find(35); // it == set.end()
10 it = set.find(30); // *it = 30
11
12 set.erase(it); // set: 10 20 40 50
13 set.erase(10); // set: 20 30 40 50
14
15 set.clear();   // set: <empty>
```

时间复杂度

查找的复杂度是 $O(\log n)$
使用第一种方法删除一个元素的复杂度是 $O(1)$
使用第二种方法删除一个元素复杂度是 $O(\log n)$
清空的复杂度是 $O(n)$.

stack - 栈

set 提供了与基本数据结构功能一致的栈。

声明

```
stack< 值类型 > XXX;
```

使用

```
1  stack<int> mystack;
2
3  for (int i=0; i<5; ++i) mystack.push(i);
4
5  cout << "Popping out elements...";
6  while (!mystack.empty()) {
7      cout << ' ' << mystack.top();
8      mystack.pop();
9  }
10 std::cout << '\n';
11
12 // Popping out elements... 4 3 2 1 0
```

vector - 向量，变长数组

vector 提供了与数组操作类似，但长度可变的数组。

声明

```
vector< 值类型 > XXX;
```

使用

```
1  vector<int> myvector;
2
3  for (int i = 0; i < 10; i++) myvector.push_back(10
   ↪  - i);
4
5  myvector.size(); // 10
6  myvector[3]; // 7
7
8  sort(myvector.begin(), myvector.end());
9  for (int i = 0; i < 10; i++) cout << myvector[i] <<
   ↪  ' ';
10 // 1 2 3 4 5 6 7 8 9 10
```

- http://www.cplusplus.com/reference/

- http://www.cplusplus.com/reference/
- 别人的代码