

ACM寒假训练Day3

暨南大学 徐锬浩





基础数据结构+STL

- 基础数据结构：堆栈，队列，链表
- 常用头文件以及函数：

<algorithm>

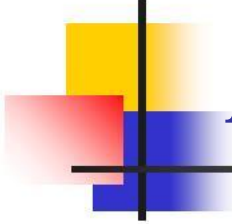
<stack>

<queue>

<map>

<set>

<vector>



Algorithm中的常用函数

`int a = 30; int b = 25;`

最大值max:

`Int c = max(a, b) ; c = 30;`

最小值min:

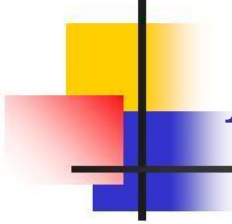
`Int c = min(a, b); c = 25;`

交换函数swap:

`Swap(a, b) a = 25, b = 30;`

最小公倍数gcd:

`Int c = __gcd(a, b); c = 5`



Algorithm中的常用函数

- 常用取整函数

1.floor()（翻译：地板）向下取整，找到一个最大的比该数小的整数

2.ceil()（翻译：天花板）向上取整，找到一个最小的比该数大的整数

3.round()（翻译：XXX）四舍五入取整

```
cout<<floor(3.6)<<endl;  
cout<<ceil(3.6)<<endl;  
cout<<round(3.6)<<endl;  
cout<<round(3.4)<<endl;
```

```
3  
4  
4  
3
```



一、堆栈

- 特点:

- 先进后出

- 入栈 $O(1)$, 出栈 $O(1)$

- 不能随机访问中间的元素

- 实现方法:

- 链表

- 数值

- STL



一、堆栈

•STL使用方法:

```
#include<stack>
```

```
using namespace std;
```

```
stack<int> S;
```

<u>empty</u> ↵	Tests if the stack is empty. ↵
<u>pop</u> ↵	Removes the element from the top of the stack. ↵
<u>push</u> ↵	Adds an element to the top of the stack. ↵
<u>size</u> ↵	Returns the number of elements in the stack. ↵
<u>stack</u> ↵	Constructs a stack that is empty or that is a copy of a base container object. ↵
<u>top</u> ↵	Returns a reference to an element at the top of the stack. ↵



一、堆栈

```
9 = int main(){
10     stack<int> s;
11     cout<<s.empty()<<endl;
12 =   for(int i=1; i<=10; i++){
13         s.push(i);
14     }
15     s.pop();
16     cout<<s.top()<<endl;
17     cout<<s.size()<<endl;
18     cout<<s.empty()<<endl;
19 }
```

```
1
9
9
0
```



二、队列

- 特点:

- 先进先出FIFO

- 入队 $O(1)$, 出队 $O(1)$

- 不能随机访问中间的元素

- 实现方法:

- 链表

- 数值

- STL



二、队列

•STL使用方法:

```
#include<queue>
```

```
using namespace std;
```

```
queue<int> Q;
```

<u>back</u> ↵	Returns a reference to the last and most recently added element at the back of the queue. ↵
<u>empty</u> ↵	Tests if the queue is empty. ↵
<u>front</u> ↵	Returns a reference to the first element at the front of the queue. ↵
<u>pop</u> ↵	Removes an element from the front of the queue. ↵
<u>push</u> ↵	Adds an element to the back of the queue. ↵
<u>queue</u> ↵	Constructs a queue that is empty or that is a copy of a base container object. ↵
<u>size</u> ↵	Returns the number of elements in the queue. ↵

二、队列

```
10 int main(){
11     queue<int> q;
12     cout<<q.empty()<<endl;
13     for(int i=1; i<=10; i++){
14         q.push(i);
15     }
16     q.pop();
17     cout<<q.front()<<endl;
18     cout<<q.back()<<endl;
19     cout<<q.empty()<<endl;
20 }
```

```
1
2
10
0
```



三、排序

- 排序：

- 冒泡，插入，选择 $O(n^2)$

- 归并 $O(n \cdot \log(n))$

- 快排 $O(n \cdot \log(n))$

- 桶排序 $O(m)$



三、排序

STL:

```
1  #include<algorithm>
2  using namespace std;
3  int a[maxn];
4
5  sort(a, a+n);
6  sort(a, a+n, cmp);
7
8  bool cmp(int x, int y){
9      return x > y;
10 }
```



三、排序

```
1  #include<algorithm>
2  #include<iostream>
3  using namespace std;
4  int a[10];
5  bool cmp(int x, int y){
6      return x > y;
7  }
8  int main(){
9      a[0] = 52; a[1] = 26; a[2] = 593;
10     a[3] = 160; a[4] = 1;
11     sort(a, a+5);
12     for(int i=0; i<5; i++){
13         cout<<a[i]<<" ";
14     }cout<<endl;
15     sort(a, a+5, cmp);
16     for(int i=0; i<5; i++){
17         cout<<a[i]<<" ";
18     }
19 }
```



四、STL之pair

类模板： `template <class T1, class T2> struct pair`

参数： **T1**是第一个值的数据类型， **T2**是第二个值的数据类型。

功能： **pair**将一对值组合成一个值，这一对值可以具有不同的数据类型（**T1**和**T2**），两个值可以分别用**pair**的两个公有函数**first**和**second**访问。



四、STL之pair

具体用法：

1、定义（构造）：

```
1 pair<int, double> p1; //使用默认构造函数
2 pair<int, double> p2(1, 2.4); //用给定值初始化
3 pair<int, double> p3(p2); //拷贝构造
```

2、访问两个元素（first和second）：

```
1 pair<int, double> p1; //使用默认构造函数
2 p1.first = 1;
3 p1.second = 2.5;
4 cout << p1.first << ' ' << p1.second << endl;
```



四、STL之pair

具体用法：

3、赋值：

(1)利用make_pair:

```
1  pair<int, double> p1;  
2  p1 = make_pair(1, 1.2);  
3
```

(2)变量间赋值:

```
1  pair<int, double> p1(1, 1.2);  
2  pair<int, double> p2 = p1;  
3
```


四、STL之pair

```
1 #include <iostream>
2 #include <utility>
3 #include <string>
4 using namespace std;
5 int main () {
6     pair <string,double> product1 ("tomatoes",3.25);
7     pair <string,double> product2;
8     pair <string,double> product3;
9     product2.first = "lightbulbs"; // type of first is string
10    product2.second = 0.99; // type of second is double
11    product3 = make_pair ("shoes",20.0);
12    cout << "The price of " << product1.first << " is $" << product1.second << "\n";
13    cout << "The price of " << product2.first << " is $" << product2.second << "\n";
14    cout << "The price of " << product3.first << " is $" << product3.second << "\n";
15    return 0;
16 }
```

```
The price of tomatoes is $3.25
The price of lightbulbs is $0.99
The price of shoes is $20
```



五、STL之vector

介绍

vector是表示可变大小数组的序列容器。

就像数组一样，**vector**也采用的连续存储空间来存储元素。也就意味着可以采用下标对**vector**的元素进行访问，和数组一样高效。但是又不像数组，它的大小是可以动态改变的，而且它的大小会被容器自动处理。

使用**vector**是需要使用头文件`#include<vector>`



五、STL之vector

vector的基本操作

(1). 容量

向量大小: `vec.size();`

向量最大容量: `vec.max_size();`

更改向量大小: `vec.resize();`

向量真实大小: `vec.capacity();`

向量判空: `vec.empty();`

减少向量大小到满足元素所占存储空间的大小:

`vec.shrink_to_fit();`



五、STL之vector

vector的基本操作

(2). 修改

多个元素赋值: `vec.assign();`

末尾添加元素: `vec.push_back();`

末尾删除元素: `vec.pop_back();`

任意位置插入元素: `vec.insert();`

任意位置删除元素: `vec.erase();`

交换两个向量的元素: `vec.swap();`

清空向量元素: `vec.clear();`



五、STL之vector

vector的基本操作

(3)迭代器

开始指针: `vec.begin();`

末尾指针: `vec.end();` //指向最后一个元素的下一个位置

(4)元素的访问

下标访问: `vec[1];` //并不会检查是否越界

at方法访问: `vec.at(1);`

访问第一个元素: `vec.front();`

访问最后一个元素: `vec.back();`



五、STL之vector

vector的基本操作

(4) 算法
遍历元素

```
1  vector<int>::iterator it;
2  for (it = vec.begin(); it != vec.end(); it++)
3      cout << *it << endl;
4  //或者
5  = for (size_t i = 0; i < vec.size(); i++) {
6      cout << vec.at(i) << endl;
7  }
```



五、STL之vector

vector的基本操作

(4)算法
元素排序

```
1  #include <algorithm>
2  sort(vec.begin(), vec.end()); //采用的是从小到大的排序
3  //如果想从大到小排序, 可以采用上面反转函数, 也可以采用下面方法:
4  bool Comp(const int& a, const int& b) {
5      return a > b;
6  }
7  sort(vec.begin(), vec.end(), Comp);
```



六、STL之string

概念

string是STL的字符串类型，通常用来表示字符串。而在使用string之前，字符串通常是用char*表示的。

string和char*的区别：

string是一个类, char*是一个指向字符的指针。

string封装了char*，管理这个字符串，是一个char*型的容器。也就是说string是一个容器，里面元素的数据类型是char*。

string不用考虑内存释放和越界。

string管理char*所分配的内存。每一次string的复制，取值都由string类负责维护，不用担心复制越界和取值越界等。

string提供了一系列的字符串操作函数

查找find，拷贝copy，删除erase，替换replace，插入insert

具体学习参考：

https://blog.csdn.net/sinat_20265495/article/details/52502315



七、STL之map

简介

map是一类关联式容器。它的特点是增加和删除节点对迭代器的影响很小，除了那个操作节点，对其他的节点都没有什么影响。
对于迭代器来说，可以修改实值，而不能修改key。

map的功能

自动建立Key — value的对应。key 和 value可以是任意你需要的类型。
根据key值快速查找记录，查找的复杂度基本是Log(N)，如果有1000个记录，最多查找10次，1,000,000个记录，最多查找20次。
快速插入Key -Value 记录。
快速删除记录
根据Key 修改value记录。
遍历所有记录。



七、STL之map

使用

```
#include<map>
using namespace std;
map<int, int> mp;
```

数据插入

(1)用insert函数插入pair数据

```
1  map<int, string> mapStudent;
2  mapStudent.insert(pair<int, string>(1, "student_one"));
3  mapStudent.insert(pair<int, string>(2, "student_two"));
4  mapStudent.insert(pair<int, string>(3, "student_three"));
5
```



七、STL之map

数据插入

(2)用insert函数插入value_type数据

```
1  map<int, string> mapStudent;  
2  mapStudent.insert(map<int, string>::value_type (1, "student_one"));  
3  mapStudent.insert(map<int, string>::value_type (2, "student_two"));  
4  mapStudent.insert(map<int, string>::value_type (3, "student_three"));
```

(3)用数组方式插入数据

```
1  map<int, string> mapStudent;  
2  mapStudent[1] = "student_one";  
3  mapStudent[2] = "student_two";  
4  mapStudent[3] = "student_three";  
5
```



七、STL之map

map的大小

直接使用size函数

```
1  map<int, string> mapStudent;  
2  cout<<mapStudent.size()<<endl;  
3
```

map的遍历

使用迭代器

```
1  map<int, string>::iterator iter;  
2  for(iter = mapStudent.begin(); iter != mapStudent.end(); iter++){  
3      cout<<iter->first<<' '<<iter->second<<endl;  
4  }
```



七、STL之map

map的查找

用find函数来定位数据出现位置，它返回的一个迭代器，当数据出现时，它返回数据所在位置的迭代器，如果map中没有要查找的数据，它返回的迭代器等于end函数返回的迭代器。

```
8      map<char, int> mp;
9      mp.insert(pair<char, int>('a', 5));
10     mp.insert(pair<char, int>('b', 6));
11     mp.insert(pair<char, int>('c', 7));
12     mp.insert(pair<char, int>('d', 8));
13     map<char, int>::iterator iter;
14     iter = mp.find('a');
15     if(iter != mp.end())
16         cout<<"Find, the value is "<<iter->second<<endl;
17     else
18         cout<<"Do not Find"<<endl;
19 }
```



八、STL之set

简介

set，顾名思义，就是数学上的集合——每个元素最多只出现一次，并且**set**中的元素已经从小到大排好序。

头文件：

```
#include<set>
```

```
using namespace std;
```

```
set<int> st;
```



八、STL之set

常用操作:

begin()	返回set容器的第一个元素的地址
end()	返回set容器的最后一个元素地址
clear()	删除set容器中的所有的元素
empty()	判断set容器是否为空
size()	返回当前set容器中的元素个数
erase(it)	删除某个元素, 参数是一个元素值或者迭代器
insert(a)	插入某个元素

参考:

<http://www.cnblogs.com/yaoyueduzhen/p/4536929.html>



九、STL之priority_queue

优先队列

与queue队列不同的地方在于它具有优先权，既每次push一个元素进去以后都会把优先级最高的放在最前面。

常用函数：

empty()	如果队列为空返回真
pop()	删除队顶元素
push()	加入一个元素
size()	返回优先队列中拥有的元素
top()	返回优先队列队顶元素



九、STL之priority_queue

`priority_queue<Type, Container, Functional>`

Type为数据类型， Container为保存数据的容器， Functional为元素比较方式。

如果不写后两个参数，那么容器默认用的是vector，比较方式默认用operator<，也就是优先队列是大顶堆，队头元素最大。

`priority_queue<int> p; //大的先出队`

`priority_queue<int, vector<int>, greater<int> > p; //小的先出队`

自定义比较函数来决定优先级



训练

训练网址：

<https://vjudge.net/contest/280397>

JNU_STL入门