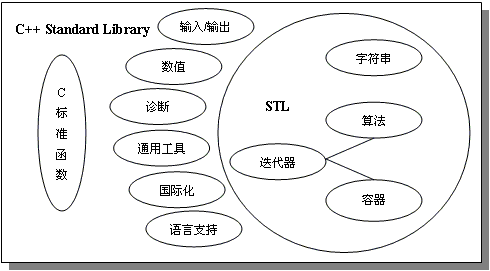
**C++ STL(标准模板库)**

　总体上，在C++标准函数库中，STL主要包含了容器、算法、迭代器。string也可以算做是STL的一部分。



**if(!q.empty())**

**{**

**ans+=q.top();**

**q.pop();**

**}**

**三个操作搭配一起使用是好习惯，能尽量防止出错。**

**1表List:**

头文件 #include<list.h>

|  |  |
| --- | --- |
| List的构造 | |
| list<int> c | 产生一个空表c，其中没有任何元素 |
| list<int> c(n) | 产生一个大小为n的表c |
| list<int> c(n,elem) | 产生一个大小为n的表c，每个元素值都是elem |
| list<int> c(beg, end) | 产生一个表c，以区间[beg, end)做为元素初值 |
| list的非变动性操作 | |
| **c.empty()** | **判断表是否为空,是空就返回true（1），不空返回false（0）** |
| c.size() | 返回表的大小（目前有多少元素） |
| c.max\_size() | 返回容器最大的可以存储的元素（容量上限，由址空间决定） |
| c1 == c2 | 判断c1 是否等于c2 |
| c1 != c2 | 判断c1是否不等于c2 |
| c1 < c2 | 判断c1 是否小于c2 |
| c1 > c2 | 判断c1 是否大于c2 |
| c1 <= c2 | 判断c1是否小于等于c2 |
| c1 >= c2 | 判断c1是否大于等于c2 |
| list的赋值操作 | |
| c1 = c2 | 将c2的全部元素赋值给c1 |
| c.assign(n, elem) | 复制n个elem，复制给c |
| c.assign(beg, end) | 将区间[beg;end）内的元素赋值给c |
| c1.swap(c2) | 将c1和c2元素互换 |
| swap(c1,c2) | 同上，此为全局函数 |
| list元素之间存取 | |
| **c.front()** | **返回第一个元素，不检查元素存在与否** |
| **c.back()** | **返回最后一个元素，不检查元素存在与否** |
| list迭代器相关函数 | |
| c.begin() | 返回一个双向迭代器，指向第一个元素 |
| c.end() | 返回一个双向迭代器，指向最后一个元素的下一个位置 |
| c.begin() | 返回一个逆向迭代器，指向逆向迭代的第一个元素 |
| c.end() | 返回一个逆向迭代器，指向逆向迭代的最后一个元素的下一个位置 |
| list安插、移除操作函数（list<char>::iterator pos=L.begin(); pos++；） | |
| c.insert(pos, elem) | 在pos位置上安插一个elem元素，并返回新元素的位置 |
| c.insert(pos,n,elem) | 在pos位置上插入n个elem元素，无返回值 |
| c.insert(pos,beg,end) | 在pos位置上插入区间[beg,end]内的所有元素，没有返回值 |
| **c.push\_front()** | **在头部添加一个elem元素** |
| **c.push\_back(elem)** | **在尾部添加一个elem元素** |
|  |  |
| c.remove(elem) | 移除所有其值为elem的元素 |
| c.erase(pos) | 移除pos位置上的元素，返回下一个元素的位置 |
| c.erase(beg, end) | 移除[beg, end)区间内的所有元素，返回下一个元素的位置 |
| c.pop\_front() | 移除第一个元素，无返回值 |
| c.pop\_back() | 移除最后一个元素，无返回值 |
| c.clear() | 移除所有元素，将容器清空 |
| List的特殊变动性操作 | |
| c1.splice(pos, c2) | 将c2内的所有元素转移到c1之内pos所指的位置上 |
| c1.splice(pos1,c2,pos2) | 将c2内的pos2所指元素转移到c1之内的pos1所指位置上 (c1,c2可相同) |
| c.sort() | 以operator<为准则，对所有元素排序 |
| c.sort(cmp) | 以cmp()为准则，对所有元素排序 |
| c1.merge(c2) | 假设c1和c2容器都包含已序(相同的排序方式)元素，将c2的全部元素转移到c1，并保证合并后的list还是已序。 |
| c1.merge(c2，op) | 假设c1和c2容器都包含op()原则下的已序(相同的排序方式)元素，将c2的全部元素转移到c1，并保证合并后的list在op（）原则仍是已序。 |
| c.reverse() | 将所有元素反序 |

例如：

#include<cstdio>

#include<list>

using namespace std;

int main( )

{

char a = 'a',b = 'b',c = 'c',d = 'd';

list<char> L;

list<char>::iterator it=L.begin(); it++;

L.insert(it,a);

L.push\_back('b');

L.push\_back(c);

while(!L.empty())

{

printf("%d",L.size());

printf("%c ",L.front());

L.pop\_front();

}

return 0;

}

**2栈**

**c++ stl栈stack的头文件为**:#include <stack>

**c++ stl栈stack的成员函数介绍**

empty() 堆栈为空则返回真

pop() 移除栈顶元素

push() 在栈顶增加元素

size() 返回栈中元素数目

top() 返回栈顶元素

**3队列：**

头文件：#include<queue>

1. 普通队列quque：

queue的构造

queue<elem>q;

queue的使用

q.push( x ) //在队尾增加元素x

q.front( ) //输出队首元素

q.pop( ) //删除队首元素

q.empty( ) //判断队列是否空，空则返回true

q.size( ) //返回队列元素个数

1. 优先队列priority\_queue：

priority\_quque的构造：

priority\_queue<Type, Container, Functional>，模板声明带有三个参数：Type 为数据类型， Container 为保存数据的容器，Functional 为元素比较方式。

Container 必须是用数组实现的容器，比如 vector, deque 但不能用 list. STL里面默认用的是 vector。

第一种，比较方式默认用 operator< ,优先队列就是大顶堆，队头元素最大。

**priority\_quue<int> pq;**

第二种，自己写cmp比较函数。

Struct cmp

{

bool operator () (const int &a, const int &b)

{

return a>b;

}

};

**priority\_queue< int, vector<int>, cmp >pq;**

第三种，重载operator<号。注意 ：重载>号会编译出错，因为标准库默认使用元素类型的<操作符来确定它们之间的优先级关系。另外由于某些编译器的问题要提前申明

**Struct Node;**

**Bool operator <(const int &a, const int &b);**

Struct Node

{

int a,b,c;

friend bool operator<(const int &a, const int &b);

　　　　 }

**prority\_queue<Node>pq；**

prority\_queue的使用

q.push( x ) //在队尾增加元素x

q.top( ) //输出队首元素

q.pop( ) //删除队首元素

q.empty( ) //判断队列是否空，空则返回true

q.size( ) //返回队列元素个数

[注意：queue用front，但是priority\_queue用](http://www.itnose.net/detail/623274.html)top

注意：因为队列是队尾增加，队首输出，所以重载的内容若是“< ”是队首最大，从大到小输出； 若是“>”是队首最小，从小到大输出。