# STL初步

## 不定长数组 vector

头文件 #include <vector>

vector.size() 读取它的大小

vector.resize() 改变它的大小

vector.push\_back() 向尾部添加元素

vector.pop\_back() 删除最后一个元素

vector.clear() 清空

以UVa101为例：

代码：

#include <cstdio>

#include <cmath>

#include <ctype.h>

#include <iostream>

#include <algorithm>

#include <cstring>

#include <cstdlib>

#include <typeinfo>

#include <vector>

#include <iomanip>

#include <map>

#include <queue>

#include <set>

#include <stack>

using namespace std;

vector<int>pile[25];

int n;

void find\_num(int a,int &p,int &h)

{

for(p=0;p<n;p++)

{

for(h=0;h<pile[p].size();h++)

{

if(pile[p][h]==a) return;

}

}

}

void clear\_above(int p,int h)

{

for(int i=h+1;i<pile[p].size();i++)

{

int x=pile[p][i];

pile[x].push\_back(x);

}

pile[p].resize(h+1);

}

void pile\_onto(int p,int h,int p2)

{

for(int i=h;i<pile[p].size();i++)

{

pile[p2].push\_back(pile[p][i]);

}

pile[p].resize(h);

}

void print()

{

for(int i=0;i<n;i++)

{

printf("%d:",i);

for(int j=0;j<pile[i].size();j++) printf(" %d",pile[i][j]);

printf("\n");

}

}

int main()

{

int a,b;

char s1[10],s2[10];

while(scanf("%d",&n)!=EOF)

{

for(int i=0;i<n;i++) pile[i].push\_back(i);

while(scanf("%s",s1)&&s1[0]!='q')

{

scanf("%d%s%d",&a,s2,&b);

int pa,pb,ha,hb;

find\_num(a,pa,ha);

find\_num(b,pb,hb);

if(pa==pb) continue;

if(s2[1]=='n') clear\_above(pb,hb);

if(s1[0]=='m') clear\_above(pa,ha);

pile\_onto(pa,ha,pb);

}

print();

}

return 0;

}

# 映射(关联数组) map

PS：pair数据是两个数据的整合 比如：pair<int,string>p p.first()=int p.second()=string

map就是从键（key）到值（value）的映射

头文件 #include <map>

数据的插入：

用insert()函数：

map.insert(pair< , >( , ))

map.insert(map< , >::value\_type())

用数组赋值

map[int]=string

区别： 用insert函数不能赋值给已有的数据，用数组赋值会覆盖

比如： insert(1,”abc”) insert(1,”def”) map[1]=”abc”

而 map[1]=”abc” map[1]=”def” map[1]=”def”

map.size() 得到map的大小

map< , >::iterator 迭代器

map< , >::reverse\_iterator 反向迭代器

map.count() 判断数据是否出现，出现返回1，否者返回0

map.find() 返回数据位置，返回为一个迭代器，如没有该数据返回迭代器等于end函数迭代器

lower\_bound函数用法，这个函数用来返回要查找关键字的下界(是一个迭代器)

upper\_bound函数用法，这个函数用来返回要查找关键字的上界(是一个迭代器)

例如：map中已经插入了1，2，3，4的话，如果lower\_bound(2)的话，返回的2，而upper\_bound（2）的话，返回的就是3

map.clear() 清空数据

map.empty() 判断是否为空，为空返回true

map.erase() 数据删除

//如果要删除1,用迭代器删除

map<int, string>::iterator iter;

iter = mapStudent.find(1);

mapStudent.erase(iter);

//如果要删除1，用关键字删除

int n = mapStudent.erase(1);//如果删除了会返回1，否则返回0

//用迭代器，成片的删除

//一下代码把整个map清空

mapStudent.earse(mapStudent.begin(), mapStudent.end());

//成片删除要注意的是，也是STL的特性，删除区间是一个前闭后开的集合

以Uva156为例：

代码：

#include <cstdio>

#include <cmath>

#include <ctype.h>

#include <iostream>

#include <algorithm>

#include <cstring>

#include <cstdlib>

#include <typeinfo>

#include <vector>

#include <iomanip>

#include <map>

#include <queue>

using namespace std;

vector<string>word;

string tmp;

map<string, int>ans;

string strtolower(string s)

{

for (int i=0;i<s.length();i++)

s[i] = tolower(s[i]);

sort(s.begin(),s.end());

return s;

}

int main()

{

ans.clear();

while (cin >> tmp)

{

if (tmp == "#") break;

word.push\_back(tmp);

tmp = strtolower(tmp);

if (!ans.count(tmp)) ans[tmp] = 1;

else ans[tmp]++;

}

sort(word.begin(),word.end());

for (int i = 0; i < word.size(); i++)

if (ans[strtolower(word[i])] == 1)

cout << word[i] << "\n";

return 0;

}

# 栈

所谓栈就是符合“后进先出”（Last In First Out，LIFO）的数据结构。

头文件 #include <stack>

s.empty() 如果栈为空返回true，否则返回false

s.size() 返回栈中元素的个数

s.pop() 删除栈顶元素但不返回其值

s.top() 返回栈顶的元素，但不删除该元素

s.push() 在栈顶压入新元素

以UVa12096为例：

代码：

#include <cstdio>

#include <cstdlib>

#include <iostream>

#include <string>

#include <stack>

#include <set>

#include <map>

#include <algorithm>

#include <vector>

using namespace std;

typedef set<int> Set;

map<Set,int>IDcache;

vector<Set>Setcache;

#define ALL(x) x.begin(),x.end()

#define INS(x) inserter(x,x.begin())

int ID(Set x)

{

if(IDcache.count(x)) return IDcache[x];

Setcache.push\_back(x);

return IDcache[x]=Setcache.size()-1;

}

int main()

{

int T,n;

string op;

scanf("%d",&T);

while(T--)

{

stack<int>s;

scanf("%d",&n);

for(int i=1;i<=n;i++)

{

cin>>op;

if(op[0]=='P') s.push(ID(Set()));

else if(op[0]=='D') s.push(s.top());

else

{

Set x1 = Setcache[s.top()];s.pop();

Set x2 = Setcache[s.top()];s.pop();

Set x;

if(op[0]=='U') set\_union(ALL(x1),ALL(x2),INS(x));

if(op[0]=='I') set\_intersection(ALL(x1),ALL(x2),INS(x));

if(op[0]=='A') {x=x2;x.insert(ID(x1));}

s.push(ID(x));

}

cout<<Setcache[s.top()].size()<<endl;

}

cout<<"\*\*\*"<<endl;

}

return 0;

}

# 集合 set

自定义类型可以定义set，但必须定义“小于”运算符（因为set中元素以从小到大排列好）

set< >::iterator 迭代器，类似于指针

以UVa10815为例：

代码：

#include <cstdio>

#include <cstdlib>

#include <iostream>

#include <string>

#include <set>

#include <sstream>

using namespace std;

set<string>word;

int main()

{

string s,tmp;

while(cin>>s)

{

for(int i=0;i<s.length();i++)

{

if(isalpha(s[i])) s[i]=tolower(s[i]);

else s[i]=' ';

}

stringstream ss(s);

while(ss >> tmp) word.insert(tmp);

}

for(set<string>::iterator it=word.begin();it!=word.end();it++) cout<< \*it <<endl;

return 0;

}

# 队列 queue

队列是符合“先进先出”（First In First Out，FIFO）的数据类型

头文件：#include <queue>

queue.push() 元素入队

queue.pop() 元素出队

queue.front() 返回队首元素但不删除

queue.back() 返回最后一个元素

queue.empty() 为空返回真

queue.size() 元素个数

以UVa540 为例：

代码：

#include <cstdio>

#include <cmath>

#include <ctype.h>

#include <iostream>

#include <algorithm>

#include <cstring>

#include <cstdlib>

#include <typeinfo>

#include <vector>

#include <iomanip>

#include <map>

#include <queue>

#include <set>

#include <stack>

using namespace std;

const int mmax = 10005;

int main()

{

int n,m,t,k=1;

while(scanf("%d",&t)&&t)

{

printf("Scenario #%d\n",k++);

map<int,int> team;

for(int i=1;i<=t;i++)

{

scanf("%d",&n);

while(n--)

{

scanf("%d",&m);

team[m]=i;

}

}

queue<int> bq,q[mmax];

while(1)

{

string s;

cin>>s;

if(s[0]=='S') break;

else if(s[0]=='E')

{

scanf("%d",&m);

n = team[m];

if(q[n].empty()) bq.push(n);

q[n].push(m);

}

else if(s[0]=='D')

{

m=bq.front();

printf("%d\n",q[m].front());

q[m].pop();

if(q[m].empty()) bq.pop();

}

}

printf("\n");

}

return 0;

}

# 优先队列 priority\_queue

优先队列是一种抽象数据类型（Abstract Data Type，ADT）行为有些像队列，但先出队列的元素不是先进队列的元素，而是队列中优先级最高的元素，这样就可以允许类似于“急诊病人插队”的事情发生。

queue.empty() 如果优先队列为空，则返回真

queue.pop() 删除第一个元素

queue.push() 加入一个元素

queue.size() 返回优先队列中拥有的元素的个数

queue.top() 返回优先队列中有最高优先级的元素

对于一般基本数据类型默认“数值越大优先级越大”

如要改为“数值越大优先级越小”

例：priority\_queue<int,vector<int>,greater<int>>pq;

自定义类型要重载“小于”

例：（个位数大优先级小）

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

struct cmp

{

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

{

return a % 10 > b % 10;

}

}

以UVa136为例：

代码：

#include <cstdio>

#include <cstdlib>

#include <iostream>

#include <algorithm>

#include <queue>

#include <set>

using namespace std;

typedef long long LL;

const int sz[3]={2,3,5};

int main()

{

priority\_queue<LL,vector<LL>,greater<LL> >pq;

set<LL>Unum;

pq.push(1);

Unum.insert(1);

for(int i=1;;i++)

{

LL x=pq.top();pq.pop();

if(i==1500)

{

printf("The 1500'th ugly number is %lld.\n",x);

break;

}

for(int j=0;j<3;j++)

{

LL x2=x\*sz[j];

if(!Unum.count(x2))

{

pq.push(x2);

Unum.insert(x2);

}

}

}

return 0;

}