一. 什么是STL?

STL（Standard Template Library），即标准模板库，是一个具有工业强度的，高效的C++程序库。它被容纳于C++标准程序库（C++ Standard Library）中，是ANSI/ISO C++标准中最新的也是极具革命性的一部分。该库包含了诸多在计算机科学领域里所常用的基本数据结构和基本算法。为广大C++程序员们提供了一个可扩展的应用框架，高度体现了软件的可复用性。

STL的一个重要特点是数据结构和算法的分离。尽管这是个简单的概念，但这种分离确实使得STL变得非常通用。例如，由于STL的sort()函数是完全通用的，你可以用它来操作几乎任何数据集合，包括链表，容器和数组；

STL另一个重要特性是它不是面向对象的。为了具有足够通用性，STL主要依赖于模板而不是封装，继承和虚函数（多态性）——OOP的三个要素。你在STL中找不到任何明显的类继承关系。这好像是一种倒退，但这正好是使得STL的组件具有广泛通用性的底层特征。另外，由于STL是基于模板，内联函数的使用使得生成的代码短小高效；

从逻辑层次来看，在STL中体现了泛型化程序设计的思想，引入了诸多新的名词，比如像需求（requirements），概念（concept），模型（model），容器（container），算法（algorithmn），迭代子（iterator）等。与OOP（object-oriented programming）中的多态（polymorphism）一样，泛型也是一种软件的复用技术；

从实现层次看，整个STL是以一种类型参数化的方式实现的，这种方式基于一个在早先C++标准中没有出现的语言特性--模板（template）。

二.STL简述

STL是一个具有工业强度的，高效的C++程序库她实现了诸多在计算机科学领域内常用的基本数据结构和基本算法。

STL主要包括了容器、算法、迭代器。

1.容器

是容纳、包含相同类型元素的对象，主要用类模板实现

序列型容器：容器中的元素按照线性结构组织起来，可以逐个读写元素。主要包括vector（向量）、deque（双端队列）、list（双向链表）。

关联型容器：通过键（key）存储和读取元素。主要有map（映射）、set（集合）等。

容器适配器：是对前面提到的某些容器进行再包装，使其变成另一种容器。主要的有stack（栈）、queue（队列）等。

2.迭代器

是用于确定元素位置的数据类型，可以用来遍历容器中的元素

通过迭代器可以读取、修改它指向的元素，它的用法和指针类似。

每一种容器都定义了一种迭代器

定义一个容器类的迭代器的方法可以是

容器类名<元素类型>::iterator 变量名

```
vector::iterator  it;
```

访问一个迭代器指向的元素：

*迭代器变量名

```
*it=5;
```

3.算法

由许多函数模板组合成的集合，实现了大量的通用算法，用于操纵各种容器

STL中提供的算法涉及到：比较、交换、查找、遍历、复制、修改、移除、反转、排序、合并等。大约有70中标准算法

算法通过迭代器来操纵容器中的元素

算法可以处理容器，也可以处理C语言数组

三.STL容器

1. Vector

- 向量，变长数组
- 用于初始数组容量不确定的情况
- 用邻接表实现图

1.使用方式

```
#include <vector>
using namespace std;

vector<typename> name;

```

2. 访问方式

1. 下标访问
2. 使用迭代器**iterator** 访问

```
#include <vector>
using namespace std;

struct node{
  int x;
  int y ;

}node;

vector<node>a;
a[i].x;

int main(){
    Vector<int> v;
  pair<int,int> p;
  v.push();    //向数组中添加元素
    ...
   v.clear();
    //1.使用 下标访问
    for(int i = 0;i<v.size();i++){
        printf("%d",v[i]);
    }

    //2.使用 迭代器访问]

    vector<int> :: itrator it;

    for(vertor(int)::iterator it =
v.begin();it!=v.end();it++){
        printf("%d",*it);
    }

  for(auto p : v){

  }
}
```

```
36
```

3. 常用函数

函数名	函数作用	时间复杂度
push_back()	在容器尾部添加元素	O(1)
pop_back()	在容器尾部删除元素	O(1)
size()	获取数组容量	O(1)
clear()	清空数组	O(N)
insert（it，x）	在迭代器it位置添加x元素	O(N)
erase（it）/ erase（first,last）	删除迭代器it位置的元素 删除[first,last)位置的元素	O(N)

2. queue&&priority_queue

queue

- 先进先出
- 一般用于广度优先搜索（BFS）

一、使用方式

```
#include<queue>
using namespace std;

queue<typename> name;


```

二、访问方式

```
#include <cstdio>
#include <queue>
using namespace std;

int main(int argc, char const *argv[])

```

```
{
    queue<int> q;
    q.push(12);
    q.push(13);
    q.push(15);
  //因为queue是一种先进先出表，所以使用 front() 访问队首元素，back()访问队尾
    printf("%d\n",q.front());
    printf("%d",q.back());
    return 0;
}
```

三、常用函数

函数名	函数功能	时间复杂度
push()	将元素进队列	O(1)
front()/top（）	访问队首元素	O(1)
back()	访问队尾元素	O(1)
size()	获取队列大小	O(1)
pop()	将队首出队列	O(1)
empty()	判断队列是否为空	O(1)

Priority_queue

- 优先队列
- 底层使用堆来实现，每次队首都是优先级最大的元素 什么是堆?

一、使用方式

```
#include<queue>
using namespace std;

priority_queue<typename> q;

```

二、访问方式

二、访问方式

```
#include <cstdio>
#include <queue>
#include <string>
```

```cpp
using namespace std;

int main(int argc, char const *argv[])
{
    priority_queue<int,vector<int>,greater<> > q; //大根堆
    priority_queue<int,vector<int>,less<>> q;//小根堆
    q.push(1);
    q.push(6);
    q.push(3);
  //同队列，有限队列只能使用 top() 来访问队顶首元素
    printf("%d\n",q.top());

    return 0;

}
```

三、调整优先级

基本数据类型

- 在优先队列中，基本数据类型默认优先级为：从大到小（char为字典序从大到小）

```cpp
//以下两种表示无区别

priority_queue<int> q;

priority_queue<int,vector<int>,less<int> >q;//小根堆

priority_queue<int,vector<int>,greater<int>>q;//大根堆
```

- 第二种表达方式的第二个参数 vector 表示承载 堆 的容器
- 第三个参数是第一个参数 int 的比较类，less 表示数字越大优先级越大，可替换参数：greater 表示数字越小优先级越大

例题

队列操作

答案

3.stack

- 栈
- 后进先出
- 用来模拟递归，防止栈内存的限制而导致程序出错

一、使用方式

```
#include<stack>
using namespace std;

stack<typename> name;

```

二、访问方式

```
#include <cstdio>
#include <stack>
using namespace std;


int main(int argc, char const *argv[])
{
    stack<int> s;
    s.push(12);
    s.push(3);
  //由于 栈 是一个先进后出表，所以只能使用 top() 访问队首元素
    printf("%d", s.top());

    return 0;

}


() () ( ( () ) () () () )
```

三、常用函数

函数名	函数功能	时间复杂度
push()	将元素 入栈	O(1)
pop()	将元素栈顶元素 出栈	O(1)
top()	获取栈顶元素	O(1)
size()	获取 栈 的容量	O(1)
empty()	判断栈是否为空	O(1)

例题

二元函数

答案

4. map&&multimap

- 映射
- 将任何类型映射到任何类型,在map中数据以key的字典序排序
- Map中键和值的都是唯一的，如果需要一个键对应多个值可参考multimap

一、使用方式

```
#include<map>
using namespace std;

map<typename,typename> name;

```

二、访问方式

1. 通过下标访问
2. 通过迭代器访问

```
#include <cstdio>
#include <map>
using namespace std;


int main(int argc, char const *argv[])
{
    map<char,int> mp;
  map <string ,int >mp;
  map<int,int>mp;
   mp.clear();
    mp['c'] = 12;
   printf("%d\n",mp['c']);// 12
    mp['d'] = 13;
    mp['a'] = 1;
    mp['f'] = 43;
    mp['g'] = 1;
    //map中key唯一，后添加的会覆盖前面的值
    mp['c'] = 121;

     printf("%d\n",mp['c']); // 121

    for (map<char,int>::iterator it = mp.begin();
it!=mp.end(); it++)
```

```
    {
        //通过it->first访问key的值，通过it->second访问value的值
        printf("%c: %d\t",it->first,it->second);
    }

    return 0;
}
while(~scanf("%d",&a))
while(cin >> a)


```

三、常用函数

函数名	函数功能	时间复杂度
find（key）	返回键为key的映射的迭代器	O(logN)
erase(it) erase(key) erase(first,last)	删除单个元素 删除区间元素	O(1) O(logN) O(last-first)
size()	获取映射容量	O(1)
clear()	清空映射	O(N)

例题

错误票据

答案

5. set

- 集合
- 内部自动有序
- 自动去重

一、使用方式

```
#include<set>
using namespace std;

set<typename> name;
```

二、访问方式

- 只支持 迭代器访问

```
#include<stdio.h>
#include<set>
using namespace std;

int main(){
    set<int> s;
    s.insert(12);    //添加元素
    ...


    for(set<int>:: iterator it = s.begin();it!=s.end();it++){
        printf("%d",*it);
    }
  for(auto p : s)
}
第一行一个数字t代表测试数据数量
 第二行一个数字n代表数据个数
  第三行n个数代表数据
  3
  3
  1 2 3
  4
  3 2 1 5
  5
  1 2 3 4 5

#include<set>
#include<iostream>
using namespace std;
const int N = 1000;
int a[N];

set<int>s;

int main(){
  int t;
  cin >> t;

  while(t--){
   s.clear();
    int n;
    cin >> n;
    for(int i = 1; i <= n; i ++)
    {
      cin >> a[i];
```

```
46          s.insert(a[i]);
47      }
48      for(set<int>:: iterator it = s.begin();it!=s.end();it++){
49          printf("%d",*it);
50      }
51
52    }
53    return 0;
54  }
55
56
```

三、常用函数

函数名	函数功能	时间复杂度
insert(x)	向集合中插入元素	O(N)
find(x)	在集合中查询，返回对应迭代器	O(logN)
size()	获取集合大小	O(1)
clear()	清空集合	O(1)
erase(it) /earse(value)/ erase(first It,last)	删除迭代器所指向的数值 /删除指定的数值/ 删除first It所指向到last地址的元素	O(N)/ O(1)/ O(last-first)

例题

集合运算

答案

四. algorithm库中的算法/函数

algorithm常用函数

函数名	函数功能
min(x,y)	比较两个数的大小，返回较小的
max(x,y)	比较两个数大小，返回较大的

函数名	函数功能
abs()	返回 整数 类型的绝对值
swap(x,y)	交换两个数的值
reverse(p1，p2)	可以将数组指针在 [p1,p2)间的元素反转
next_permutation()	返回一个序列全排列的下一个 序列
fill (first,last,val)	将数组或容器中任意一段区间赋相同值
sort() // log (n)	将可比较元素的区间进行排序
lower_bound(first，last，val)	在一个 有序 的数组(容器)中，返回在 [first,last)区间里第一个值大于等于 **val** 元素的位置（指针）
upper_bound()	在一个 有序 的数组(容器)中，返回在 [first,last)区间里第一个值大于 **val** 元素的位置（指针）

```
int a = 1, b = 2;
cout << min(a,b); // a
int c = min(a,b);
cout << c;

swap(a[i],a[j]);
int a[10];
string s = "abcd";
reverse(s,s+4);
reverse(a,a+10);
reverse(s.begin(),s.end());
cout << s <<endl; //dcba

 int a[]={1, 2, 3, 4};
do{
  .....
}while(next_permutation(a,a+4));

```

sort

- 基本使用方法
  sort()函数可以对给定区间所有元素进行排序。它有三个参数sort(begin, end, cmp)，其中begin为指向待sort()的数组的第一个元素的指针，end为指向待sort()的数组的最后一个元素的下一个位置的指针，cmp参数为排序准则，cmp参数可以不写，如果不写的话，默认从小到大进行排序。如果我们想从大到小排序可以将cmp参数写为greater()就是对int数组进行排序，当然<>中我们也可以写double、long、float等等。如果我们需要按照其他的排序准则，那么就需要我们自己定义一个bool类型的函数来传入。比如我们对一个整型数组进行从大到小排序：

```
#include<iostream>
#include<algorithm>
using namespace std;
bool cmp(int x,int y){
  return x > y;
  return x < y;
}
int main(){
  int num[10] = {6,5,9,1,2,8,7,3,4,0};
  sort(num,num+10,greater<int>());
  sort(num,num+10,cmp);
  for(int i=0;i<10;i++){
    cout<<num[i]<<" ";
  }//输出结果:9 8 7 6 5 4 3 2 1 0


  return 0;

}
```

- 自定义排序准则
  上面我们说到sort()函数可以自定义排序准则，以便满足不同的排序情况。使用sort()我们不仅仅可以从大到小排或者从小到大排，还可以按照一定的准则进行排序。比如说我们按照每个数的个位进行从大到小排序，我们就可以根据自己的需求来写一个函数作为排序的准则传入到sort()中。
  我们可以将这个函数定义为：
- 自定义排序准则
  上面我们说到sort()函数可以自定义排序准则，以便满足不同的排序情况。使用sort()我们不仅仅可以从大到小排或者从小到大排，还可以按照一定的准则进行排序。比如说我们按照每个数的个位进行从大到小排序，我们就可以根据自己的需求来写一个函数作为排序的准则传入到sort()中。
  我们可以将这个函数定义为：

```
bool cmp(int x,int y){
  return x % 10 > y % 10;

```

然后我们将这个cmp函数作为参数传入sort()中即可实现了上述排序需求。

```
#include<iostream>
#include<algorithm>
using namespace std;

bool cmp(int x,int y){
  return x % 10 > y % 10;
}

int main(){
  int num[10] = {65,59,96,13,21,80,72,33,44,99};
  sort(num,num+10,cmp);
  for(int i=0;i<10;i++){
    cout<<num[i]<<" ";
  }//输出结果：59 99 96 65 44 13 33 72 21 80


  return 0;

}
```

- 对结构体进行排序
  sort()也可以对结构体进行排序，比如我们定义一个结构体含有学生的姓名和成绩的结构体Student，然后我们按照每个学生的成绩从高到底进行排序。首先我们将结构体定义为：

```
struct Student{
  string name;
  int score1;
  int score2;
  Student() {}
  Student(string n,int s):name(n),score(s) {}

  bool operator >(Student x){

  }

}a[100];

   运算符重载
```

根据排序要求我们可以将排序准则函数写为：

```
bool cmp_score(Student x,Student y){
  if(x.score1 == y.score1)
  return x.score2 > y.score2;
  else return x.score1 > y.score2;
}
sort(a,a+10,cmp_score);
```

- 对STL容器进行排序

```
//基础版
vector<int>vec;
sort(vec.begin(), vec.end());


//进阶版

typedef pair<int,int>pii;
vector<pii>vec;
bool cmp(pii x,pii y){
  if(x.first != y.first)
  return x.first > y.first;
  else return x.second < y.first;
}
sort(vec.begin(),vec.end(),cmp);

```

运算符重载

```
//pair 写法
#include<iostream>
#include<algorithm>
using namespace std;
const int N = 110;

pair<string ,int>pii[N];

bool cmp(pii x, pii y){
  if(x.second != y.second)
    return x.scond > y.second;
  else return x.first < y.first;
}

int main(){

  itn n;
  cin >> n;
  for(int i = 1; i <= n; i++){
```

```
20      cin >> pii[i].first >> pii[i].second;
21    }
22    sort(pii + 1, pii + 1 + n,cmp);
23
24    for(int i = 1; i <= n; i++){
25      .....
26    }
27    return 0;
28  }
```

例题

成绩排名

答案