Assignment 10

191300051 [1]

汪福运 [1]

一、概念题

1.1 STL主要包含哪些内容？它们的功能分别是什么？

STL中包含有

- 容器：容器用于存储序列化的数据，如：向量、队列、栈、集合等。
    - vector
    - list
    - deque
    - stack
    - queue
    - priority_queue
    - map
    - multimap
    - set
    - multiset
    - basic_string<字符类型>
    - 每个容器都提供了一些成员函数来实现基本的操作
    - 如果一个容器的元素类型是一个类，那么针对该类很有可能需要，自定义拷贝构造函数，重载小于函数。
- 算法：算法用于对容器中的数据元素进行一些常用操作，如：排序、统计。
- 迭代器：
    - 迭代器实现了抽象的指针功能，它们指向容器中的数据元素，用于对容器中的数据元素进行遍历和访问。
    - 迭代器是容器和算法之间的桥梁：传给算法的不是容器，而是指向容器中元素的迭代器，算法通过迭代器实现对容器中数据的访问。这样使得算法与容器保持独立，从而提高算法的通用性。
    - 输出迭代器
    - 输入迭代器

- 前向迭代器
- 双向迭代器
- 随机访问迭代器

1.2 迭代器有哪几种类型？为什么sort 算法不支持对list类的排序

- 输入迭代器

提供对数据的只读访问

- 输出迭代器

提供对数据的只写访问

- 前向迭代器

提供读写操作，并能向前推进迭代器

- 双向迭代器

提供读写操作，并能向前向后操作

- 随机访问迭代器

提供读写操作，并能以条约的方式访问容器的任意数据，是功能最强的迭代器。

sort能够接受的是随机访问迭代器！而与list相关联的迭代器类型为双向迭代器！

但是实际上list容器内部实现了排序的算法

```
List<int> l;
l.sort();
```

二、编程题

2.1 请围绕PPT中的最后的学生信息统计应用示例，基于STL实现下面的功能.

1. 升序输出计算机专业男生的姓名
2. 升序输出出生地是“南京”、专业为哲学或数学的学生的年龄(年龄计算可以以代码实现时的日期为准)
3. 统计全部女生的平均年龄
4. 统计出生地是“南京”的计算机专业学生的平均年龄
5. 统计非计算机专业年龄小于20岁的学生的平均年龄

```
// 2.1和2.2你还可能用到:
//从指定范围内复制使cond为真的所有元素到另一个范围内
OutIt copy_if(InIt src_first, InIt src_last, OutIt dst_first,
Pred cond);
//将操作f应用于指定范围内的所有元素并存储到另一个范围内
OutIt transform(InIt src_first, InIt src_last, OutIt dst_first,
Op f);
OutIt transform(InIt1 src_first1, InIt src_last1, InIt2
src_first2, OutIt
dst_first, Op f);
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
#include <numeric>
using namespace std;
enum Sex
{
    MALE,
    FEMALE
};
enum Major
{
    MATHEMATICS,
    PHYSICS,
    CHEMISTRY,
    COMPUTER,
    GEOGRAPHY,
    ASTRONOMY,
    ENGLISH,
    CHINESE,
    PHILOSOPHY
```

```
};
class Date
{
    int year;
    int month;
    int day;

public:
    Date(int y, int m, int d)
    {
        year = y;
        month = m;
        day = d;
    }
    int get_year() const { return year; }
    int get_month() const { return month; }
    int get_day() const { return day; }
    void display()
    {
        cout << year << '/' << month << '/' << day;
    }
};
class Student
{
    int no;
    string name;
    Sex sex;
    Date birth_date;
    string birth_place;
    Major major;

public:
    Student(int no1, char *name1, Sex sex1,
            Date birth_date1, char *birth_place1, Major major1)
: name(name1), birth_date(birth_date1),
birth_place(birth_place1)
    {
        no = no1;
        sex = sex1;
        major = major1;
    }
    int get_no() const { return no; }
    string get_name() const { return name; }
```

```
64      Sex get_sex() const { return sex; }
65      Major get_major() const { return major; }
66      string get_birth_place() const { return birth_place; }
67      int get_age(const Date &dt) const //const 会约束不能够改变对象
    的数据成员!
68      {
69          int age = dt.get_year() - birth_date.get_year();
70          if (dt.get_month() < birth_date.get_month())
71          {
72              age -= 1;
73          }
74          else if (dt.get_month() == birth_date.get_month() &&
    dt.get_day() < birth_date.get_day())
75          {
76              age -= 1;
77          }
78          return age;
79      }
80      void display()
81      {
82          cout << no << ", " << name << ", "
83               << (sex == MALE ? "male" : "female") << ", ";
84          birth_date.display();
85          cout << ", " << birth_place << ", ";
86          switch (major)
87          {
88          case MATHEMATICS:
89              cout << "Mathematics";
90              break;
91          case PHYSICS:
92              cout << "Physics";
93              break;
94          case CHEMISTRY:
95              cout << "Chemistry ";
96              break;
97          case COMPUTER:
98              cout << "Computer";
99              break;
100         case GEOGRAPHY:
101             cout << "Geography";
102             break;
103         case ASTRONOMY:
104             cout << "Astronomy";
```

```
            break;
        case ENGLISH:
            cout << "English";
            break;
        case CHINESE:
            cout << "Chinese";
            break;
        case PHILOSOPHY:
            cout << "Philosophy";
            break;
        }
    }
};
bool match_major_and_sex(Student &st) //判断st是否为计算机专业的女生?
{
    return st.get_major() == COMPUTER && st.get_sex() == FEMALE;
}
bool match_birth_place(Student &st) //判断st的出生地是否为"南京"
{
    return (st.get_birth_place()).find("南京") != string::npos;
}
bool compare_no(Student &st1, Student &st2) //比较st1和st2的的学号
{
    return st1.get_no() < st2.get_no();
}
void display_student_info(Student &st) //显示st的信息?
{
    st.display();
    cout << endl;
}

class MatchMajorAndSex
{
    Major major;
    Sex sex;

public:
    MatchMajorAndSex(Major m, Sex s)
    {
        major = m;
        sex = s;
```

```
    }
    bool operator()(Student &st)
    {
        return st.get_major() == major && st.get_sex() == sex;
    }
};
class MatchBirthPlace
{
    string birth_place;

public:
    MatchBirthPlace(char *bp) { birth_place = bp; }
    bool operator()(Student &st) //判断st的出生地是否"南京"
    {
        return (st.get_birth_place()).find(birth_place) !=
string::npos;
    }
};
int main()
{
    vector<Student> students; //创建存放学生信息的容器students

    //获得所有学生的数据，存储在容器students中，
    students.push_back(Student(2, "zhang", FEMALE, Date(1991,
10, 1),
                               "江苏南京", COMPUTER));
    students.push_back(Student(5, "li", MALE, Date(1992, 10,
1),
                               "北京", PHILOSOPHY));
    students.push_back(Student(1, "wang", FEMALE, Date(1993,
10, 1),
                               "南京", COMPUTER));
    students.push_back(Student(6, "sun1", MALE, Date(1994, 12,
30), "江苏省南京市", PHILOSOPHY));
    students.push_back(Student(7, "sun2", MALE, Date(1995, 12,
30), "江苏省南京市", MATHEMATICS));
    students.push_back(Student(4, "qian", MALE, Date(1996, 10,
1),
                               "上海", PHILOSOPHY));
    students.push_back(Student(3, "zhao", MALE, Date(1997, 10,
1),
                               "江苏南京", COMPUTER));
```

```
180     students.push_back(Student(3, "zhao", MALE, Date(2010, 10,
    1),
181                                "江苏南京",PHYSICS));
182
183     //......
184
185     //统计计算机专业女生的人数
186     cout << "计算机专业女生的人数是："
187          << count_if(students.begin(), students.end(),
    match_major_and_sex)
188          << endl;
189
190     //统计出生地为"南京"的学生人数?
191     cout << "出生地为\"南京\"的学生人数是:"
192          << count_if(students.begin(), students.end(),
    match_birth_place)
193          << endl;
194
195     //按照学号进行排序了！！
196     //按学号由小到大对students的元素进行排序?
197     sort(students.begin(), students.end(), compare_no);
198
199     //按学号由小到大输出所有学生的信息
200     cout << "按学号排序后的学生信息：\n";
201     for_each(students.begin(), students.end(),
    display_student_info);
202
203     //编译命令要加上-std=c++11 or -std=gnu++11?
204     //升序输出计算机专业男生的姓名
205     vector<Student> s1;
206     //这里必须要用inserter！因为这些算法不会自动的扩容！
207     copy_if(students.begin(), students.end(), inserter(s1,
    s1.begin()), MatchMajorAndSex(COMPUTER, MALE));
208     //
    copy_if(students.begin(),students.end(),back_inserter(s1),[]
    (const Student &st)->bool{return st.get_sex()==MALE &&
    st.get_major()==COMPUTER;});
209     for_each(s1.begin(), s1.end(), [](const Student &st) ->
    void { cout << st.get_name() << endl; });
210     //升序输出出生地是“南京”、专业为哲学或数学的学生的年龄(年龄计算可以以代
    码实现时的日期为准)
```

```
    for_each(students.begin(), students.end(), [](const Student &st) -> void {if ((st.get_birth_place()).find("南京") != string::npos  && (st.get_major()==PHILOSOPHY || st.get_major()==MATHEMATICS)){cout<<st.get_no()<<":"<<st.get_name()<<":"<<"age:"<<st.get_age(Date(2021,5,15))<<endl; } });
    //统计全部女生的平均年龄
    s1.clear();
    copy_if(students.begin(), students.end(), inserter(s1, s1.begin()), [](const Student &st) -> bool { return st.get_sex() == FEMALE; });
    cout << accumulate(s1.begin(), s1.end(), 0, [](double partial, const Student &st) -> double { return partial + st.get_age(Date(2021, 5, 15)); }) / s1.size() << endl;
    //统计出生地是“南京”的计算机专业学生的平均年龄
    s1.clear();
    copy_if(students.begin(), students.end(), inserter(s1, s1.begin()), [](const Student &st) -> bool { return (st.get_birth_place()).find("南京") != string::npos && st.get_major() == COMPUTER; });
    cout << (double)accumulate(s1.begin(), s1.end(), 0, [](double partial, const Student &st) -> double { return partial + st.get_age(Date(2021, 5, 15)); }) / s1.size() << endl;
    //统计非计算机专业年龄小于20岁的学生的平均年龄
    s1.clear();
    copy_if(students.begin(), students.end(), inserter(s1, s1.begin()), [](const Student &st) -> bool { return st.get_age(Date(2021, 5, 15)) <= 20 && st.get_major() != COMPUTER; });
    cout << (double)accumulate(s1.begin(), s1.end(), 0, [](double partial, const Student &st) -> double { return partial + st.get_age(Date(2021, 5, 15)); }) / s1.size() << endl;
    return 0;
}

```

输出结果

```
计算机专业女生的人数是：2
出生地为"南京"的学生人数是:6
按学号排序后的学生信息：
1, wang, female, 1993/10/1, 南京, Computer
```

```
2, zhang, female, 1991/10/1, 江苏南京, Computer
3, zhao, male, 1997/10/1, 江苏南京, Computer
3, zhao, male, 2010/10/1, 江苏南京, Physics
4, qian, male, 1996/10/1, 上海, Philosophy
5, li, male, 1992/10/1, 北京, Philosophy
6, sun1, male, 1994/12/30, 江苏省南京市, Philosophy
7, sun2, male, 1995/12/30, 江苏省南京市, Mathematics
zhao
6:sun1:age:26
7:sun2:age:25
28
26.3333
10
```

2.2 请基于STL在买main函数完成下面要求的任务

```
#include <vector>
#include <algorithm>
using namespace std;
class Point
{
int x, y;
public:
Point(int _x, int _y) : x(_x), y(_y) {}
};
int main()
{
vector<Point> p, q;
p.push_back(Point(-1, -1));
p.push_back(Point(2, 2));
q.push_back(Point(1, 1));
q.push_back(Point(-2, -2))...
// 1. 对p、q中顶点"(x, y)" 升序排序并输出(要求按照x大小，若x相等则按y的大小)
// 2. 升序输出p中满足x > 0, y > 0的所有顶点与(0, 0)的距离的平方
// 3. 根据1排序后p中顶点的顺序计算满足x > 0, y > 0相邻两个顶点的距离的平方和
// 4. 计算p中x > 0, y > 0的顶点与(0, 0)的距离的平方和
```

```
// 5. p、q在每个象限顶点数目相同，统计在1排序后p、q中满足x < 0, y < 0的顶点中按顺序所对
应顶点距离的平方为2的数目
```

```
#include <vector>
#include <algorithm>
#include <numeric>
#include <iostream>
using namespace std;
class Point
{
int x, y;
public:
Point(int _x=0, int _y=0) : x(_x), y(_y) {}
int getNormSqure() const
{
    return x*x+y*y;
}
int getX()const
{
    return x;
}
int getY()const
{
    return y;
}
void display()const
{
    cout <<"x:"<< x <<" y:"<<y<<endl;
}
};
int main()
{
vector<Point> p, q;
p.push_back(Point(-1, -1));
p.push_back(Point(-2,-2));
p.push_back(Point(2, 2));
p.push_back(Point(2,100));
p.push_back(Point(3, 3));
p.push_back(Point(4, 4));
p.push_back(Point(5, 5));
p.push_back(Point(6, 6));
```

```
q.push_back(Point(1, 1));
q.push_back(Point(2, 2));
q.push_back(Point(2,100));
q.push_back(Point(3, 3));
q.push_back(Point(4, 4));
q.push_back(Point(5, 5));
q.push_back(Point(6, 6));
q.push_back(Point(-2, -2));
q.push_back(Point(-3,-3));
// 1. 对p、q中顶点"(x, y)" 升序排序并输出(要求按照x大小，若x相等则按y的大小)
// 2. 升序输出p中满足x > 0, y > 0的所有顶点与(0, 0)的距离的平方
// 3. 根据1排序后p中顶点的顺序计算满足x > 0, y > 0相邻两个顶点的距离的平方和
// 4. 计算p中x > 0, y > 0的顶点与(0, 0)的距离的平方和
// 5. p、q在每个象限顶点数目相同，统计在1排序后p、q中满足x < 0, y < 0的顶点中按顺序所对
//应顶点距离的平方为2的数目


//1
sort(p.begin(),p.end(),[](const Point &p1,const Point &p2)->bool {return p1.getX()<p2.getX()|| (p1.getX()==p2.getX()&& p1.getY()<p2.getY());});
sort(q.begin(),q.end(),[](const Point &p1,const Point &p2)->bool {return p1.getX()<p2.getX()|| (p1.getX()==p2.getX()&& p1.getY()<p2.getY());});
cout<<"task1\n";
for_each(p.begin(),p.end(),[](const Point& p)->void{p.display();});
for_each(q.begin(),q.end(),[](const Point& p)->void{p.display();});

//2
cout<<"task2\n";
vector<Point> lst;
copy_if(p.begin(),p.end(),back_inserter(lst),[](const Point &p) {return p.getX()>0 && p.getY()>0;});
sort(lst.begin(),lst.end(),[](const Point &p1,const Point &p2)->bool {return p1.getNormSqure()<p2.getNormSqure();});
for_each(lst.begin(),lst.end(),[](const Point p1)->void {p1.display();});

```

```
70
71  cout<<"task3\n";
72  lst.clear();
73  copy_if(p.begin(),p.end(),back_inserter(lst),[](const Point &p)
    {return p.getX()>0 && p.getY()>0;});
74  vector<Point> lsttmp;
75  transform(lst.begin()+1,lst.end(),lst.begin(),back_inserter(lstt
    mp),[](const Point &p1,const Point& p2)->Point{return
    Point(p1.getX()-p2.getX(),p1.getY()-p2.getY());});
76  for_each(lst.begin(),lst.end(),[](const Point p1)->void
    {p1.display();});
77  for_each(lsttmp.begin(),lsttmp.end(),[](const Point p1)->void
    {p1.display();});
78  cout << accumulate(lsttmp.begin(),lsttmp.end(),0,[](int
    partial,const Point& p)->int{return partial +
    p.getNormSqure();})<<endl;
79
80  cout <<"task4\n";
81  cout << accumulate(lst.begin(),lst.end(),0,[](int partial,const
    Point& p)->int{return partial + p.getNormSqure();})<<endl;
82
83
84  cout <<"task5\n";
85  vector<Point> lst1;
86  vector<Point> lst2;
87  vector<Point> lst3;
88  copy_if(p.begin(),p.end(),back_inserter(lst1),[](const Point &p)
    {return p.getX()<0 && p.getY()<0;});
89  copy_if(q.begin(),q.end(),back_inserter(lst2),[](const Point &p)
    {return p.getX()<0 && p.getY()<0;});
90  transform(lst1.begin(),lst1.end(),lst2.begin(),back_inserter(lst
    3),[](const Point &p1,const Point& p2)->Point{return
    Point(p1.getX()-p2.getX(),p1.getY()-p2.getY());});
91  cout<<count_if(lst3.begin(),lst3.end(),[](const Point &p)-
    >bool{return p.getNormSqure()==2;})<<endl;
92  }
```

输出结果

```
1  task1
2  x:-2 y:-2
3  x:-1 y:-1
```

```
x:2 y:2
x:2 y:100
x:3 y:3
x:4 y:4
x:5 y:5
x:6 y:6
x:-3 y:-3
x:-2 y:-2
x:1 y:1
x:2 y:2
x:2 y:100
x:3 y:3
x:4 y:4
x:5 y:5
x:6 y:6
task2
x:2 y:2
x:3 y:3
x:4 y:4
x:5 y:5
x:6 y:6
x:2 y:100
task3
x:2 y:2
x:2 y:100
x:3 y:3
x:4 y:4
x:5 y:5
x:6 y:6
x:0 y:98
x:1 y:-97
x:1 y:1
x:1 y:1
x:1 y:1
19020
task4
10184
task5
2
```