Assignment 10

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？

解：容器：容器用于存储序列化的数据，如：向量、队列、栈、集合等；算法（函数）：算法用于对容器中的数据元素进行一些常用操作，如：排序、统计等；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元素的访问，这样使得算法与容器保持独立，从而提高算法的通用性。

1.2 迭代器有哪几种类型？为什么sort算法不支持对list类型的排序？

解：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、反向迭代器、插入迭代器；sort算法需要随机访问迭代器，list所关联的迭代器为双向迭代器。

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.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
#include <iterator>
#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:
    static Date current_date;
    Date(int y, int m, int d) { year = y; month = m; day = d; }
    int get_year() { return year; }
    int get_month() { return month; }
    int get_day() { return day; }
    void display() { cout << year << '/' << month << "/" << day; }
};
Date Date::current_date(2021, 5, 13);
class Student
{
    int no;
    string name;
    Sex sex;
```

```
    string birth_place;
    Date birth_date;
    Major major_;
  public:
    Student(int no1, char *name1, Sex sex1, Date birth_date1,
            char *birth_place1, Major major1):
        no(no1), name(name1), sex(sex1), birth_date(birth_date1),
        birth_place(birth_place1), major_(major1) {}
    int get_no() { return no; }
    string get_name() { return name; }
    int get_sex() { return sex; }
    Major get_major() { return major_; }
    string get_birth_place() { return birth_place; }
    int get_age() {
        if (Date::current_date.get_month() > birth_date.get_month()
             || Date::current_date.get_month() == birth_date.get_month()
                && Date::current_date.get_day() >= birth_date.get_day()) {
            return Date::current_date.get_year() - birth_date.get_year() - 1;
        } else {
            return Date::current_date.get_year() - birth_date.get_year();
        }
    }
    void display() {
        // cout << name << " ";
    }
};

int main()
{
    vector<Student> students;
    students.push_back(Student(0, "A", FEMALE, Date(2000, 6, 2), "南京",
COMPUTER));
    //...

    // 1. 升序输出计算机专业男生的姓名
    {
    vector<Student> sts;
    vector<string> names;
    copy_if(students.begin(), students.end(), back_inserter(sts),
            [](Student &st) { return st.get_major() == COMPUTER
               && st.get_sex() == MALE; });
    transform(sts.begin(), sts.end(), back_inserter(names),
              [](Student &st) { return st.get_name(); });
    sort(names.begin(), names.end());
    for_each(names.begin(), names.end(), [](string name) { cout << name; });
    cout << endl;
    }

    // 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)
    {
    vector<Student> sts;
    vector<int> ages;
    copy_if(students.begin(), students.end(), back_inserter(sts),
            [](Student &st) { return st.get_birth_place().find("南京")
             != string::npos && ( st.get_major() == PHILOSOPHY ||
              st.get_major() == MATHEMATICS ); });
    transform(sts.begin(), sts.end(), back_inserter(sts),
```

```
            [](Student &st) { return st.get_age(); });
    sort(ages.begin(), ages.end());
    for_each(ages.begin(), ages.end(), [](int age) { cout << age << " "; });
    cout << endl;
    }

    // 3. 统计全部女生的平均年龄
    {
    vector<Student> sts;
    vector<int> ages;
    copy_if(students.begin(), students.end(), back_inserter(sts),
            [](Student &st) { return st.get_sex() == FEMALE; });
    transform(sts.begin(), sts.end(), back_inserter(sts),
            [](Student &st) { return st.get_age(); });
    int total = accumulate(sts.begin(), sts.end(), 0, [](int sum, Student &st)
                           { return sum + st.get_age(); });
    cout << (sts.size() ? total/sts.size() : 0) << endl;
    }

    // 4. 统计出生地是“南京”的计算机专业学生的平均年龄
    {
    vector<Student> sts;
    vector<int> ages;
    copy_if(students.begin(), students.end(), back_inserter(sts),
            [](Student &st) { return st.get_major() == COMPUTER &&
             st.get_birth_place().find("南京") != string::npos; });
    transform(sts.begin(), sts.end(), back_inserter(sts),
            [](Student &st) { return st.get_age(); });
    int total = accumulate(sts.begin(), sts.end(), 0, [](int sum, Student &st)
                           { return sum + st.get_age(); });
    cout << (sts.size() ? total/sts.size() : -1) << endl;
    }

    // 5. 统计非计算机专业年龄小于20岁的学生的平均年龄
    {
    vector<Student> sts;
    vector<int> ages;
    copy_if(students.begin(), students.end(), back_inserter(sts),
            [](Student &st) { return st.get_major() == COMPUTER &&
             st.get_age() < 20; });
    transform(sts.begin(), sts.end(), back_inserter(sts),
            [](Student &st) { return st.get_age(); });
    int total = accumulate(sts.begin(), sts.end(), 0, [](int sum, Student &st)
                           { return sum + st.get_age(); });
    cout << (sts.size() ? total/sts.size() : -1) << endl;
    }
}
```

2.2 请基于STL在买main函数完成下面要求的任务

```
#include <vector>
#include <algorithm>
#include <iostream>
#include <numeric>
```

```
using namespace std;

class Point
{
    int x, y;

public:
    Point(int _x, int _y) : x(_x), y(_y) {}
    int get_x() const { return x; }
    int get_y() const { return 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)
    {
    sort(p.begin(), p.end(), [] (Point &p1, Point &p2)
          { return p1.get_x() < p2.get_x() || p1.get_x() == p2.get_x()
             && p1.get_y() > p2.get_y(); });
    for_each(p.begin(), p.end(), [](const Point &pt)
              { cout << '(' << pt.get_x() << ','
                 << pt.get_y() << ')' << ' '; });
    sort(q.begin(), q.end(), [] (Point &p1, Point &p2)
          { return p1.get_x() < p2.get_x() || p1.get_x() == p2.get_x()
             && p1.get_y() > p2.get_y(); });
    for_each(q.begin(), q.end(), [](const Point &pt)
              { cout << '(' << pt.get_x() << ','
                 << pt.get_y() << ')' << ' '; });
    }

    // 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
    {
    vector<Point> points;
    vector<int> dist;
    copy_if(p.begin(), p.end(), back_inserter(points), [](const Point &pt)
             { return pt.get_x() > 0 && pt.get_y() > 0; });
    transform(points.begin(), points.end(), back_inserter(dist),
               [](Point &pt) { return pt.get_x() * pt.get_x() +
                                pt.get_y() * pt.get_y(); });
    sort(dist.begin(), dist.end());
    for_each(dist.begin(), dist.end(), [](int d) { cout << d << endl; });
    }

    // 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
    {
    vector<Point> points;
    vector<int> dist;
    copy_if(p.begin(), p.end(), back_inserter(points), [](const Point &x)
            { return x.get_x() > 0 && x.get_y() > 0; });
    transform(points.begin(), points.end() - 1, points.begin() + 1,
```

```
                    back_inserter(dist), [](const Point &p1, const Point &p2)
                    { return (p1.get_x()-p2.get_x())*(p1.get_x()-p2.get_x())
                           + (p1.get_y()-p2.get_y())*(p1.get_y()-p2.get_y()); });
    cout << accumulate(dist.begin(), dist.end(), 0) << endl;
    }

    // 4. 计算p中x > 0, y > 0的顶点与(0, 0)的距离的平方和
    {
    vector<Point> points;
    vector<int> dist;
    copy_if(p.begin(), p.end(), back_inserter(points), [](const Point &pt)
             { return pt.get_x() > 0 && pt.get_y() > 0; });
    transform(points.begin(), points.end(), back_inserter(dist),
               [](Point &pt) { return pt.get_x() * pt.get_x() +
                                pt.get_y() * pt.get_y(); });
    cout << accumulate(dist.begin(), dist.end(), 0) << endl;
    }

    // 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
    {
    vector<Point> points1;
    vector<Point> points2;
    vector<int> dist;
    copy_if(p.begin(), p.end(), back_inserter(points1), [](const Point &x)
            { return x.get_x() > 0 && x.get_y() > 0; });
    copy_if(p.begin(), p.end(), back_inserter(points2), [](const Point &x)
            { return x.get_x() > 0 && x.get_y() > 0; });
    transform(points1.begin(), points1.end(), points2.begin(),
                    back_inserter(dist), [](const Point &p1, const Point &p2)
                    { return (p1.get_x()-p2.get_x())*(p1.get_x()-p2.get_x())
                           + (p1.get_y()-p2.get_y())*(p1.get_y()-p2.get_y()); });
    cout << count_if(dist.begin(), dist.end(), [](int d) { return d == 2; }) <<
endl;
    }

    return 0;
}
```