# Solution10

**191220008 陈南曈**

## 概念题

### 1、STL主要包含哪些内容？它们的功能分别是什么？

**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。

### 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？

**输出迭代器（output iterator，记为：OutIt）**

- 可以修改它所指向的容器元素
- 间接访问操作（*）
- ++操作

**输入迭代器（input iterator，记为：InIt）**

- 只能读取它所指向的容器元素
- 间接访问操作（*）和元素成员间接访问（->）
- ++、==、!=操作。

**前向迭代器（forward iterator，记为：FwdIt）**

- 可以读取/修改它所指向的容器元素
- 元素间接访问操作（*）和元素成员间接访问操作（->）
- ++、==、!=操作

**双向迭代器（bidirectional iterator，记为：BidIt）**

- 可以读取/修改它所指向的容器元素
- 元素间接访问操作（*）和元素成员间接访问操作（->）
- ++、--、==、!=操作

**随机访问迭代器（random-access iterator，记为：RanIt）**

- 可以读取/修改它所指向的容器元素
- 元素间接访问操作（*）、元素成员间接访问操作（->）和下标访问元素操作（[]）

- ++、--、+、-、+=、-=、==、!=、<、>、<=、>=操作

对于list容器类，与它关联的迭代器类型为双向迭代器，但不提供随机访问迭代器，因此不能对 list 中的元素使用 sort() 算法。

## 编程题

### 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

```
enum Sex { MALE, FEMALE };
enum Major { COMPUTER, PHYSICS, MATH, PHILOSOPHY };

class Student
{
    int no;
    string name;
    Sex sex;
    int age;
    string birth_place;
    Major major;
public:
    Student() {}
    Student(int no, string name, Sex sex, int age, string birth_place, Major
major) {
        this->no = no; this->name = name; this->sex = sex; this->age = age;
this->birth_place = birth_place; this->major = major;
    }
    int get_no() { return no; }
    string get_name() { return name; }
    int get_sex() { return sex; }
    int get_age() { return age; }
    string get_birth_place() { return birth_place; }
    Major get_major() { return major; }
};

// 升序输出计算机专业男生的姓名
void Fun1(vector<Student>students)
{
    vector<Student>now_students(students.size());
    auto it = copy_if(students.begin(), students.end(), now_students.begin(), []
(Student& st) {return st.get_major() == COMPUTER && st.get_sex() == MALE; });
    now_students.resize(distance(now_students.begin(), it));
    sort(now_students.begin(), now_students.end(), [](Student& st1, Student&
st2) {return st1.get_name() < st2.get_name(); });
    for_each(now_students.begin(), now_students.end(), [](Student& st) {cout <<
st.get_name() << endl; });
}

// 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)
void Fun2(vector<Student>students)
{
    vector<Student>now_students(students.size());
    auto it = copy_if(students.begin(), students.end(), now_students.begin(), []
(Student& st) {return st.get_birth_place()=="Nanjing" &&  (st.get_major() ==
PHILOSOPHY || st.get_major() == MATH); });
    now_students.resize(distance(now_students.begin(), it));
    sort(now_students.begin(), now_students.end(), [](Student& st1, Student&
st2) {return st1.get_age() < st2.get_age(); });
    for_each(now_students.begin(), now_students.end(), [](Student& st) {cout <<
st.get_age() << endl; });
}

// 统计全部女生的平均年龄
double Fun3(vector<Student>students)
{
    vector<Student>now_students(students.size());
    auto it = copy_if(students.begin(), students.end(), now_students.begin(), []
(Student& st) {return st.get_sex() == FEMALE; });
    now_students.resize(distance(now_students.begin(), it));
    return accumulate(now_students.begin(), now_students.end(), 0.0, [](double
partial, Student& st) {return partial + st.get_age(); }) / now_students.size();
}

// 统计出生地是“南京”的计算机专业学生的平均年龄
double Fun4(vector<Student>students)
{
    vector<Student>now_students(students.size());
    auto it = copy_if(students.begin(), students.end(), now_students.begin(), []
(Student& st) {return st.get_birth_place() == "Nanjing" && st.get_major() ==
COMPUTER; });
    now_students.resize(distance(now_students.begin(), it));
    return accumulate(now_students.begin(), now_students.end(), 0.0, [](double
partial, Student& st) {return partial + st.get_age(); }) / now_students.size();
}

// 统计非计算机专业年龄小于20岁的学生的平均年龄
double Fun5(vector<Student>students)
{
    vector<Student>now_students(students.size());
    auto it = copy_if(students.begin(), students.end(), now_students.begin(), []
(Student& st) {return  st.get_major() != COMPUTER && st.get_age() < 20; });
    now_students.resize(distance(now_students.begin(), it));
    return accumulate(now_students.begin(), now_students.end(), 0.0, [](double
partial, Student& st) {return partial + st.get_age(); }) / now_students.size();
}
```

## 2、请基于STL在买main函数完成下面要求的任务。

```
class Point
{
    int x, y;
public:
    Point(){}
    Point(int _x, int _y) : x(_x), y(_y) {}
    int get_x() { return x; }
    int get_y() { return y; }
};

// 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)
void Fun1(vector<Point>& points1, vector<Point>& points2)
{
    sort(points1.begin(), points1.end(), [](Point& point1, Point& point2)
{return point1.get_x() < point2.get_x() || (point1.get_x() == point2.get_x() &&
point1.get_y() < point2.get_y()); });
    for_each(points1.begin(), points1.end(), [](Point& point) {cout << "(" <<
point.get_x() << ", " << point.get_y() << ")" << endl; });
    sort(points2.begin(), points2.end(), [](Point& point1, Point& point2)
{return point1.get_x() < point2.get_x() || (point1.get_x() == point2.get_x() &&
point1.get_y() < point2.get_y()); });
    for_each(points2.begin(), points2.end(), [](Point& point) {cout << "(" <<
point.get_x() << ", " << point.get_y() << ")" << endl; });
}

// 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
void Fun2(vector<Point>points)
{
    vector<Point>now_points(points.size());
    auto it1 = copy_if(points.begin(), points.end(), now_points.begin(), []
(Point& point) {return point.get_x() > 0 && point.get_y() > 0; });
    now_points.resize(distance(now_points.begin(), it1));
    vector<int>dist_points(now_points.size());
    auto it2 = transform(now_points.begin(), now_points.end(),
dist_points.begin(), [](Point& point) {return point.get_x() * point.get_x() +
point.get_y() * point.get_y(); });
    dist_points.resize(distance(dist_points.begin(), it2));
    sort(dist_points.begin(), dist_points.end());
    for_each(dist_points.begin(), dist_points.end(), [](int x) {cout << x <<
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
int Fun3(vector<Point>points)
{
    vector<Point>now_points(points.size());
    auto it1 = copy_if(points.begin(), points.end(), now_points.begin(), []
(Point& point) {return point.get_x() > 0 && point.get_y() > 0; });
    now_points.resize(distance(now_points.begin(), it1));
    vector<int>dist_points(points.size() - 1);
    auto it2 = transform(now_points.begin(), now_points.end() - 1,
now_points.begin() + 1, dist_points.begin(), [](Point& point1, Point& point2)
{return (point1.get_x() - point2.get_x()) * (point1.get_x() - point2.get_x()) +
(point1.get_y() - point2.get_y()) * (point1.get_y() - point2.get_y()); });
    dist_points.resize(distance(dist_points.begin(), it2));
    return accumulate(dist_points.begin(), dist_points.end(), 0);
}

// 4. 计算p中x > 0, y > 0的顶点与(0, 0)的距离的平方和
int Fun4(vector<Point>points)
{
    vector<Point>now_points(points.size());
    auto it = copy_if(points.begin(), points.end(), now_points.begin(), []
(Point& point) {return point.get_x() > 0 && point.get_y() > 0; });
    now_points.resize(distance(now_points.begin(), it));
    return accumulate(now_points.begin(), now_points.end(), 0, [](int partial,
Point& point) {return partial + point.get_x() * point.get_x() + point.get_y() *
point.get_y(); });
}

// 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
int Fun5(vector<Point>points1, vector<Point>points2)
{
    vector<Point>now_points1(points1.size());
    auto it1 = copy_if(points1.begin(), points1.end(), now_points1.begin(), []
(Point& point) {return point.get_x() > 0 && point.get_y() > 0; });
    now_points1.resize(distance(now_points1.begin(), it1));
    vector<Point>now_points2(points2.size());
    auto it2 = copy_if(points2.begin(), points2.end(), now_points2.begin(), []
(Point& point) {return point.get_x() > 0 && point.get_y() > 0; });
    now_points2.resize(distance(now_points2.begin(), it2));
    vector<int>now_points(points1.size());
    auto it = transform(points1.begin(), points1.end(), points2.begin(),
now_points.begin(), [](Point& point1, Point& point2) {return (point1.get_x() -
point2.get_x()) * (point1.get_x() - point2.get_x()) + (point1.get_y() -
point2.get_y()) * (point1.get_y() - point2.get_y()); });
    now_points.resize(distance(now_points.begin(), it));
    return accumulate(now_points.begin(), now_points.end(), 0, [](int partial,
int point) {return partial + point == 2; });
}

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));

    vector<Point>p_copy, q_copy;
    p_copy = p, q_copy = q;

    Fun1(p_copy, q_copy);
    Fun2(p);
    cout << Fun3(p_copy) << endl;
    cout << Fun4(p) << endl;
    cout << Fun5(p_copy, q_copy) << endl;

    return 0;
}
```