**UEE1303 (1072/1074) S18: Final Examination**

**-- Programming Part --**

**Full Scores:**

100%

**Examination Time:**

150 minutes

**Instructions:**

There are four problems and each is allocated 25 points. You may pick an arbitrary number of problems to solve with a total score up to 100 points.

You are allowed to open any notes or books, but prohibited to browse on the internet to search C/C++ codes as direct answers. Read carefully the statements and requirements of each problem. Once you complete your program for one problem, please raise your hand and TA will come to you and test your program. Please note that points will be given until your program fully fulfills the requirements of each problem.

You should be aware of that some lengthy problems are not as difficult as you think. In contrast, some short problems may not be solved easily. Last but not least, **cheating** is a serious academic demeanor and should be avoided at all most. Once any cheating is caught, all your answers will be void and get 0 point for your midterm. **Good luck!**

**UEE1303(1072/1074) S18: Final Examination**

**-- Programming Part --**

**Full Scores:** 100 points

**ID Number** (學號):

**Full Name** (姓名):

|  |  |  |
| --- | --- | --- |
| 題號 | 助教簽名 | 總分 |
| 01 |  |  |
| 02 |  |
| 03 |  |
| 04 |  |
| 05 |  |

**Problem 01 Routing Problem**

A number of points are given with ID, coordinate x and coordinate y in input\_file1 (t1a\_coordinate.in). Please use a map container to store these points.

|  |
| --- |
| >***cat t1a\_coordinate.in*** ↵  ID x y  1 5 5  2 2 5  3 1 7  4 2 1  5 4 8  6 5 2 |

Requirements:

1. First, sort these points with the increasing order in coordinate x and then sort with the increasing order in coordinate y. After the sorting, show the new order of points. After the sorting, the new order should become 3->4->2->5->6->1.

3(1,7)

2(2,5)

4(2,1)

5(4,8)

1(5,5)

6(5,2)

6.08

4

3.61

6.08

3

1. Then, following the new order, compute the distance among all points (from point ID 3 to point ID 1). In this case, the distance is 6.08+4+3.61+6.08+3=22.77
2. Secondly, read the second input\_file2 (t1a\_weight.in) with weight information of each point. Note that the order of points in the weight file is random, you should use map function to match each weight to each point. Then delete the points with low weight among all points with the same coordinate x. For example, the coordinate x of point 1 and point 6 are both 5, and the weight of point 6 is 1 which is lower than the weight of point 1. Therefore, the point 6 should be deleted from the list. Show the new order after deletion as 3->4->5->1

|  |
| --- |
| >***cat t1a\_weight.in*** ↵  ID weight  3 2  5 2  1 3  2 3  6 1  4 10 |

1. After the deletion, recomputed the distance though the new path. In addition, the distance between two points should time their weight. In this case, the distance from point ID 3 to point ID 1 is (2\*10\*6.08)+(10\*2\*7.28)+(2\*3\*3.16)=286.16

3(1,7)

2(2,5)

4(2,1)

5(4,8)

1(5,5)

6(5,2)

w=2

w=3

w=10

w=2

w=3

w=1

deletion

3(1,7)

4(2,1)

5(4,8)

1(5,5)

w=2

w=10

w=2

w=3

6.08

7.28

3.16

The main function is given as follows:

|  |
| --- |
| class Point2D  {  private:  int x;  int y;  public:  …  };  class WeightedPoint: public Point2D  {  private:  int weight;  public:  …  };  int main(int argc,char\* argv[])  {  //read file 1 (t1a\_coordinate.in)  //define the map container here  map<int,Point2D> Point\_map;  //sort these points with x increasing order then with y increasing order  sort(…,…,…);  //show the new order after sorting  cout<<”after sorting, the new order is : ”<<endl;  //compute the distance through the path after sorting  double distance;  distance = Distance\_computation1();  cout<<”after sorting, the distance is : ”<<distance<<endl;  //read file 2 (t1a\_weight.in)  //delete the points with lower weight among the points in the same coordinate x.  … … …  //show the new order after deletion  cout<<”after deletion, the new order is : ”<<endl;  //compute the distance through the path after deletion  distance = Distance\_computation2();  cout<<”after deletion, the distance is : ”<<distance<<endl;  return 0;  } |

The command-line usage of the program is shown below:

>./pg01***input\_file1 input\_file2***

* Sample execution is shown below.

|  |
| --- |
| >***./pg01 t1a\_coordinate.in t1a\_weight.in*** ↵  after sorting, the new order is :  3->4->2->5->6->1  after sorting, the distance is : 22.77  after deletion, the new order is :  3->4->5->1  after deletion, the distance is :  286.16 |

***(Test Case 1)***

|  |
| --- |
| >***cat t1b\_coordinate.in*** ↵  ID x y  1 1 1  2 2 2  3 5 4  4 6 3  5 6 0  6 5 2  7 5 5  8 4 3  9 4 1  10 1 5  11 3 4  12 3 5  13 3 1  14 1 3  15 2 3  >***cat t1b\_weitght.in*** ↵  ID weight  6 11  13 3  14 18  4 1  12 1  9 4  7 15  1 3  2 8  3 2  5 2  10 8  15 12  8 3  11 1  >***./pg01 t1b\_coordinate.in t1b\_weight.in*** ↵  after sorting, the new order is :  1->14->10->2->15->13->11->12->9->8->6->3->7->5->4  after sorting, the distance is : 33.03  after deletion, the new order is :  14->15->13->9->7->5  after deletion, the distance is :  708.86 |

**Problem 02**

Please refer to the inheritance figure shown as below:

Student

Graduated

Undergraduate

Complete the code with proper data members and member functions.

1. Please define an abstract base class named Student where one pure virtual function score() is declared publicly and two data member id and class are declared privately.
2. Please define two classes named Graduated and Undergraduate which are both derived from Student. Properly redefine the above virtual functions for computing the average score among class of each student. For the graduated student, the average score is (0.5\*research+0.25\*math+0.25\*English). If the average score is lower than 70, the student should be changed to class B, otherwise changed to class A. For undergraduate student, the average score is (0.5\*math+0.5\*English). If the average score is lower than 60, the student should be changed to class B, otherwise changed to class A.

In the main function:

* First, you should print out all students in their original class.
* Then perform the average score computation and reassignment for students to suitable class. After reassignment, show the students with their average score in each class **with descending order of average score** (graduated A, graduated B, undergraduate A and undergraduate B).
* Finally show the reassignment record with **increasing order of student ID**

The main function is given as follows:

|  |
| --- |
| #include <iostream>  #include <map>  using namespace std;  class Student  {  private:  string id;  char cls; //class  double score;  public:  //…  virtual void average\_score();  };  class Graduated: public Student  {  private:  double Research;  double Math;  double English;  public:  //…  };  class Undergraduate: public Student  {  private:  double Math;  double English;  public:  //…  };  int main(int argc,char\* argv[])  {  //read file  //show the original class of graduated students  cout<<"class A in graduated school : ";  cout<<"class B in graduated school : ";    //show the original class of undergraduate students  cout<<"class A in undergraduate : ";  //…  cout<<"class B in undergraduate : ";  //…  // compute the average score of each student and then reassign their class.  //…  // show the class and average score of graduated students after reassignment with descending order of average score.  cout<<"---------------------------------"<<endl;  cout<<"class A in graduated school : "<<endl;  //…  cout<<"class B in graduated school : "<<endl;  //…  // show the class and average score of undergraduate students after reassignment with descending order of average score.  cout<<"class A in undergraduate : "<<endl;  //…  cout<<"class B in undergraduate : "<<endl;  //…  //Finally, show the reassignment record with increasing order of student ID  cout<<"---------------------------------"<<endl;  // … … …  return 0;  } |

Test1 is shown below.

|  |
| --- |
| >***cat t2a.in*** ↵  Graduated  9983412  A  Research 80  Math 90  English 70  Graduated  9983400  B  Research 90  Math 100  English 80  Graduated  9983333  A  Research 50  Math 50  English 60  Undergraduate  9913222  B  Math 90  English 70  Undergraduate  9913111  A  Math 50  English 20  Undergraduate  9988292  A  Math 90  English 90 |

The command-line usage of the program is shown below:

>./pg02***input\_file***

* Sample execution is shown below.

|  |
| --- |
| >***pg02 t2a.in*** ↵  class A in graduated school : **9983412 9983333**  class B in graduated school : **9983400**  class A in undergraduate : **9913111 9988292**  class B in undergraduate : **9913222**  class A in graduated school :  student **9983400** score **90**  student **9983412** score **80**  class B in graduated school :  student **9983333** score **52.5**  class A in undergraduated school :  student **9988292** score **90**  student **9913222** score **80**  class B in undergraduated school :  student **9913111** score **35**  ---------------------------------  Student **9913111** move from class **undergraduate A** to **B**  Student **9913222** move from class **undergraduate B** to **A**  Student **9983333** move from class **graduated A** to **B**  Student **9983400** move from class **graduated B** to **A** |

**Problem 03** **QUADRUPLE TEMPLATE**

Please write a C/C++ program to implement a template class named quadruple which can store four kinds of data and please also implement a template function named make\_quadruple to generate and return the quadruple type class.

The main function is given as follows:

|  |
| --- |
| #include <iostream>  using namespace std;  //add quadruple class  int main()  {  quadruple<double,int,string,int> t1;  quadruple<int,string,double,int> t2(1,"t2",0.01,3);  t1 = make\_quadruple<double ,int ,string, int>(0.03,3,"t1",4);  cout << t1.first << " " << t1.second << " " << t1.third << " " << t1.fourth << endl;  cout << t2.first << " " << t2.second << " " << t2.third << " " << t2.fourth << endl;  return 0;  } |

* The output is shown below.

|  |
| --- |
| 0.03 3 t1 4  1 t2 0.01 3 |

**Problem 04** **Find the order**

Please write a program to find the order of the score list.

**Pg04a (5%)**

The input file is shown as follow:

|  |
| --- |
| >***pg04 t4a.in*** ↵  5 2 //line 0  Kobe 42 //line 1  Kobe 36 //line 2  MJ 63 //line 3  Allen 22 //line 4  LBJ 44 //line 5  2 Kobe  2 MJ |

If you can read the file and print out on the screen, you can get 5%.

|  |
| --- |
| Kobe 1 42  Kobe 2 36  MJ 3 63  Allen 4 22  LBJ 5 44 |

In the first line, “Kobe” is the name of player, “1” means the Line 1 and “42” means his scores.

**pg04b (15%)**

Given a score list, your task is to find the k-th occurrence (from first line to last line) of a string name (name of player). There are m queries that you'll have to answer.

Input:

The first line contains two integers n, m (1<=n, m<=100), the number of elements in the score list, and the number of queries. Each of the following n lines contains the name of player and his score. Each of the following m lines contains two variables k (1<=k<=n) and name (the name of player).

Output:

For each query, print which line of the k-th occurrence of the name (name of player) and the score of player. If there is no such element, output 0 instead.

The input file is shown as follow:

|  |
| --- |
| >***cat t4a.in*** ↵  5 2 //line 0  Kobe 42 //line 1  Kobe 36 //line 2  MJ 63 //line 3  Allen 22 //line 4  LBJ 44 //line 5  2 Kobe // k = 2, name = kobe  2 MJ // k = 2, name = MG |

The output file is shown as follow:

|  |
| --- |
| >***cat t4a.out*** ↵  2 36  0 |

The 2-th occurrence of Kobe is Line 2. Then the output is 2 and score 36.

The 2-th occurrence of MJ didn’t appear. Then the output is 0.

The main function is given as follows:

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <cstring>  #include <map>  #include <vector>  using namespace std;  class Pair  {  public:  Pair();  Pair(int count, int score){  //...  };  int getc(){ return c;}  int gets(){ return s;}  friend ostream& operator<<(ostream& out, Pair &p);  private:  int c;  int s;  };  int main(int argc, char \*argv[])  {  // ......  return 0;  } |

**Hint: map<string, vector< Pair > > a;**

**pg04c (5%)**

Please compute the sum and average of each player.

The example is shown as follow:

The input file is shown as follow:

|  |
| --- |
| >***cat t4a.in*** ↵  5 2 //line 0  Kobe 42 //line 1  Kobe 36 //line 2  MJ 63 //line 3  Allen 22 //line 4  LBJ 44 //line 5  2 Kobe // k = 2, name = kobe  2 MJ // k = 2, name = MG |

The command-line usage of the program is shown below:

>./pg04***t4a.in t4a.out***

|  |
| --- |
| >***cat t4a.out*** ↵  2 36  0  --Allen--  sum : 22  avg : 22  --Kobe--  sum : 78  avg : 39  --LBJ--  sum : 44  avg : 44  --MJ--  sum : 63  avg : 63 |

**Note that : The names of players must be alphabetical order.**

(Case1)

|  |
| --- |
| >***./pg04 t4a.in t4a.out*** ↵  Kobe 1 42  Kobe 2 36  MJ 3 63  Allen 4 22  LBJ 5 44 |

|  |
| --- |
| >***cat t4a.out*** ↵  2 36  0  --Allen--  sum : 22  avg : 22  --Kobe--  sum : 78  avg : 39  --LBJ--  sum : 44  avg : 44  --MJ--  sum : 63  avg : 63 |

(Case2)

|  |
| --- |
| >***./pg04 t4b.in t4b.out*** ↵  Kobe 1 42  MJ 2 54  Kobe 3 36  Allen 4 15  Allen 5 21  Kobe 6 81  MJ 7 66  Kobe 8 35  MJ 9 63  Kobe 10 23  MJ 11 54  Allen 12 32  Kobe 13 33  MJ 14 42  Allen 15 22 |

|  |
| --- |
| >***cat t4b.out*** ↵  6 81  15 22  2 54  --Allen--  sum : 90  avg : 22.5  --Kobe--  sum : 250  avg : 41.6667  --MJ--  sum : 279  avg : 55.8 |

(Case3)

|  |
| --- |
| >***./pg04 t4c.in t4c.out*** ↵  Kobe 1 42  MJ 2 54  Nash 3 33  Kobe 4 36  Allen 5 15  Allen 6 21  Durant 7 99  Lin 8 10000  Kobe 9 81  MJ 10 66  LBJ 11 44  Wade 12 23  LBJ 13 55  Durant 14 100  Nash 15 22 |

|  |
| --- |
| >***cat t4c.out*** ↵  9 81  15 22  8 10000  0  14 100  --Allen--  sum : 36  avg : 18  --Durant--  sum : 199  avg : 99.5  --Kobe--  sum : 159  avg : 53  --LBJ--  sum : 99  avg : 49.5  --Lin--  sum : 10000  avg : 10000  --MJ--  sum : 120  avg : 60  --Nash--  sum : 55  avg : 27.5  --Wade--  sum : 23  avg : 23 |

**Problem 05 employees system**

You are an employer of a company. You should design a payroll system for paying your employees. There are four types of employees: **salaried employees** are paid a fixed weekly salary regardless of the number of hours worked, **hourly employees** are paid by the hour and receive overtime pay (i.e., 1.5 times their hourly salary rate) of all hours worked in excess of 40 hours, **commission employees** are paid a percentage of their sales and **base-salaried commission employees** receive a base salary plus a percentage of their sales.:

There is a class structure of the payroll system.

Please refer to the inheritance figure shown as below:

Empolyee

SalariedEmployee

CommissionEmployee

HourlyEmployee

BasePlusCommissionEmployee

Class Employee is an abstract superclass. There are two pure virtual function of class Employee, **double earnings()** and **void display()**.

Please complete the following requirement:

1. Please define **the constructor** of each subclass.
2. Please define an **earnings() function** in each subclass to calculate correct earning money and return it’s value.
3. The way to construct your input stream to a pointer array called **Employee \* employee[n]**.

The following table is described the way to calculate an **earnings()** function of different types of employees.

|  |  |
| --- | --- |
|  | earnings() |
| Employee | abstract |
| SalariedEmployee | weeklySalary |
| HourlyEmployee | if(hours <= 40)  hours\*wage  else  40\*wage+(hours-40)\*wage\*1.5 |
| CommissionEmployee | grossSales\* CommissionRate |
| BasePlusCommissionEmployee | (grossSales\* CommissionRate) + baseSalary |

The class definition is given as follows:

|  |
| --- |
| class Employee{  public:  string name;  string IDnumber;  Employee(string n, string id):name(n), IDnumber(id){}  virtual double earnings() = 0;  virtual void display() = 0;  };  class SalariedEmployee: public Employee{  public:  double weeklySalary;  // implement here  };  class HourlyEmployee: public Employee{  double wage;  double hours;  // implement here  };  class CommissionEmployee: public Employee{  double grossSales;  double CommissionRate;  // implement here  };  class BasePlusCommissionEmployee: public CommissionEmployee{  double baseSalary;  // implement here  }; |

The main function is given as follows:

|  |
| --- |
| int main(int argc,char \*argv[]){  ifstream fin;  fin.open(argv[1]);  int n;  fin>>n;  Employee \*employees[n];    // Construct your input stream to the pointer array  for(int i=0;i<n; i++){  employees[i]->display();  }  return 0;  } |

Input Definition:

* The first line is an integer and it represents the total number of the employees.
* After the second line is the data of employees. The first character is the type of classes. ‘S’ means ***Salaried Employee***, ‘H’ means ***Hourly Employee***, ‘C’ means ***Commission Employee,*** and ‘B’ means ***Based-Salaried Commission Employee***. The second term is employee’s name. The third term is the employee’s ID number and the remaining terms are the numbers that each type of employees required. For example, if it’s Salaried Employee, it means weekly salary. If it’s Hourly Employee, they mean wage and hours (in order). If it’s Commission Employee, they mean gross sales and commission rate. If it’s Based-Salaried Commission Employee, they mean gross sales, commission rate, and base salary.

|  |
| --- |
| 4 // the total number of employees  S Ryan N101 100 // type name ID weeklySalary  H Louis N102 10 60 // type name ID wage hours  C Chris N103 400 0.5 // type name ID grossSales commissionRate  B Oliver N104 400 0.5 200 // type name ID grossSales commissionRate baseSalary |

The command-line usage of the program is shown below:

>./pg05***input\_file***

Sample execution is shown below:

|  |
| --- |
| >***cat t5a.in*** ↵  4  S Ryan N101 100  H Louis N102 10 60  C Chris N103 400 0.5  B Oliver N104 400 0.5 200  >***./pg05 t5a.in*** ↵  Salaried Employee: Ryan  ID number: N101  earned: 100  ------------------  Hourly Employee: Louis  ID number: N102  earned: 700  ------------------  Commission Employee: Christina  ID number: N103  earned: 200  ------------------  Based-Salaried Commission Employee: Oliver  ID number: N104  earned: 400  ------------------ |