1.124J Foundations of Software Engineering

Problem Set 4

Due Date: Tuesday 10/17/00

Reference Readings:
From C++ Primer
(in addition to the reference reading from PS1, PS2 & PS3):

- *Chapter 11: I/O*
- *Chapter 12: Templates*
- *chapter 14: File processing*

From algorithms in C++

- *Chapter 9: Quicksort*
- *Chapter 14: Binary search*

Problem 1:[40%]

In this problem you need to develop a program that can handle user-provided data. You need to write the *main()* function and two template functions. The program should be able to read using the one template function monthly data for a number of years. It should be used twice, once to read the number of visitors, let's say to a tourist resort, and the other the income in millions of dollars.

The input data are provided in the following files, in tabular form with each row representing the year and for each row having 12 columns, one for each months

- *visitorsNumber.data*: contains the number of tourist visitors
- *touristIncome.data*: contains the revenue in millions of dollars

e.g. the *touristIncome.data* file looks as follows:

474.33 129.72 232.33 239.43 234.56 244.60 286.72 362.46 283.43 245.26 253.40 98.07

240.34 268.36 245.36 165.36 234.45 242.43 344.25 320.20 447.93 375.26 201.30 94.34

The first element represents the tourist income for the first month of the corresponding year, which in this case assume that it is 1997. The second row has the 12 values for the income for the 1 months of year 1998, and so on.

You are asked to implement the two template functions in the file *ps4_1.h,* and the *main()* function in the file *ps4_1.C*.

main(): The *main()* function should:

- Create an array of *int*s, of a 50 rows by 12 columns size, to hold the provided data. This array should be named *visitors*.
- Invoke the template function *readData* to read in the data. The *readData* function should read until the end of the provided file. Each row has 12 values, which correspond to the values of each month of a given year. The function *readData* should get from the user the initial year. It should also count the number of years for which data are provided in the input file.
- Invoke the function *writeInvertedData* to save in a file, whose name is given by the user, the numbers for visitors, in inverted format. The data should be saved in tabular form with each column representing a year and each row a month. A certain formatting should be used as shown in a sample execution of the program.
- Then, create an array of double, of size 50 rows by 12 columns, to hold the provided data. Name the array *income*.
- Invoke the template function *readData* to read in the data. The *readData* function should read all rows until the end of the provided file. Each row has 12 values, one for each month of the corresponding year. The function *readData* should get from the user the initial year. It should count the number of years for which data are provided in the input file while reading the data.
- Invoke the function *writeInvertedData* to save in a file the income values in tabular form. The name of the file should be given by the user, and the data should be saved in an "inverted format", with each column representing a year and each row a month. In contrast, the data are read in with each row corresponding to a year and each column to a month. The exact reverse of this should be used in this function, i.e. each row should correspond to a month and each column to the year. This formatting is illustrated below in a sample execution of an implemented program.

***readData()* template function:** This template function should:

- prompt the user to give the first year of statistical data, e.g. 1997
- prompt the user to give the name of the input data, e.g. *touristIncome.data*
- open, then, the file with that name
- read until the end of file in the values into a 2-dimensional array that is passed by reference from

main() to hold the corresponding data. It should also count the number of years for which data are provided while reading in the data
- print out the number of years for which data are provided

writeInvertedData() **template function: This template function should:**

- prompt the user to give the name of the file where the data should be stored
- then, save in that file the data, which are provided by the array that is passed as argument to the function, e.g. visitors or income. The data should be printed in an inverted way. A sample execution of the program, below, demonstrates the expected format that should be used while saving the data in a file. Each row should correspond to a month and each column to a year. A specified number of decimals (i.e. precision) should be used to print out the data. Values of *double* (such as those of the *income* array) should be printed with 3 decimal points, while *int* values (such as those of the *visitors* array) should be printed without any decimal point. Any *int* value should be printed as an integer and not in a scientific way, e.g. you should print 3453000 and not as 3.453+06.
- print out the number of years for which data has been saved

Sample execution of the program:

The execution of your program should have an output similar to the following.
The ***bolded and italic*** identify whatever is entered by the user of your program.

Visitors Statistics

First year: ***1997***
File with data: ***visitorsNumber.data***
Data for 3 years have been read

File to store data: ***visitors.out***
Data for 3 years have been stored to file: visitors.out

Income Statistics

First year: ***1997***
File with data: ***touristIncome.data***
Data for 2 years have been read

File to store data: ***touristIncome.out***
Data for 2 years have been stored to file: touristIncome.out

The contents of the input and output files, after the above execution, are presented below:

Input files:

visitorsNumber.data:

475544 1985572 2076432 2165239 2283546 2344460 2635040 2958672 3047626 2843543 2734526 2523400 2412307 878734
2343240 2546836 2436456 1943656 2453425 2754240 3256404 3454645 3698200 4138479 3743756 3398320 983474 923498
2845672 3456566 2341653 2734562 2534400 2346307 943734 2234544 2398636 2345346 1546256 2453435 2345240 3456504

touristIncome.data

474.33 129.72 232.33 239.43 234.56 244.60 286.72 362.46 283.43 245.26 253.40 98.07
240.34 268.36 245.36 165.36 234.45 242.43 344.25 320.20 447.93 375.26 201.30 94.34

Output files:

visitors.out:

Month	1997	1998	1999
1	475544	2412307	3743756
2	1985572	878734	3398320
3	2076432	2343240	983474
4	2165239	2546836	923498
5	2283546	2436456	2845672
6	2344460	1943656	3456566
7	2635040	2453425	2341653
8	2958672	2754240	2734562
9	3047626	3256404	2534400
10	2843543	3454645	2346307
11	2734526	3698200	943734
12	2523400	4138479	2234544

touristIncome.out:

Month	1997	1998
1	474.330	240.340
2	129.720	268.360
3	232.330	245.360

4	239.430	165.360
5	234.560	234.450
6	244.600	242.430
7	286.720	344.250
8	362.460	320.200
9	283.430	447.930
10	245.260	375.260
11	253.400	201.300
12	98.070	94.340

Problem 2:[60%]

In this problem you need to write a program that will read a number of points store them using an array of pointers to objects of the class *Point*, and print the points out, as they are. It should then call a function to sort them using the X-coordinate and print them out sorted. Then, it should invoke the same function to sort them using the Y-coordinate and print them out sorted.

Two files, in which the class *Point* is defined, are provided for you. The one is a header file named *point.h*, and the other is named *point.C*. These files are presented below. You should **NOT** make any changes to these files. However, when submitting electronically your code you need to submit these files as well, as that will be helpful during grading in order to compile and run your code.

// Problem Set#4: [point.h]

```
#ifndef POINT_4_H
#define POINT_4_H


class Point
{
private:
  double x,y;

public:
  Point(double x, double y);

  double getX(void);
  double getY(void);
```

```
  friend ostream& operator << (ostream &i, Point &p);
};

#endif
```

// Problem Set#4: [point.C]

```
#include  <iostream.h>
#include "point.h"


Point::Point(double x, double y)
 {
   this -> x = x ;
   this -> y = y ;
 }


double Point::getX(void)
{
  return x;
}

double Point::getY(void)
{
  return y;
}

ostream& operator << (ostream &o, Point &p)
{
  o << " (x,y) = (" << p.x << " , " << p.y << ")"  ;
  return o;
}
```

The code that you are asked to provide should be written in two files, the ***ps4_2.h*** and the ***ps4_2.C***. In the former you need to provide only the declarations and in the latter the actual definitions of your code.

main()

The *main()* function, which should be provided in the file ***ps4_2.C***, should do the following with this

certain order:

- Define an array of pointers to *Point* objects with the name points should be defined, as well as any other variables that you may need to use.
- In order to be able to use the sorting method to sort in both X and Y direction you can use a pointer to a member function of the class *Point*. Then you will be able to send as an argument the pointer to the member function to the sorting method and use it to once point to the function *getX ()* and sort in the X-direction, and then point to the function *getX()* and sort in the Y-direction. Therefore, you may need to define a pointer to a member function of the class *Point.*
- All the function *readPoints(),* which you also need to provide in this file to read all the points. The user should be prompt how many points are to be given, then the required memory for as many points should be dynamically allocated, and the points should be provided by the user and created dynamically with their address stored in the array of pointers.
- Then, the function *printPoint()* should be invoked to print out the points.
- After assigning the pointer to the function to the *getX* member function, in order to be able to access the X coordinate, the *quickSortPoints()* should be invoked to sort the array of pointers to *Point* objects using the X-coordinates.
- The function *printPoint()* should, then, be invoked again to print out the sorted points.
- After assigning the pointer to the function to the *getY* member function, in order to be able to access the Y coordinate, the *quickSortPoints()* should be invoked to sort the array of pointers to *Point* objects using the Y-coordinates.
- The function *printPoint()* should be invoked again to print out the sorted points
- Finally, the program should release the dynamically allocated memory calling the function *releaseMemory()* and exit.

readPoints()

This function should read in the data. You can use the provided file *dat4_2* and redirection to save some time while debugging your program. You need to read in the number of points and dynamically allocate the pointers to *Point,* and then use one by one the x and y values and dynamically allocate memory for a *Point* object and assign accordingly its address to the corresponding element of the array of pointers.

File ***dat4_2*** has the following structure:

```
5
 65.34  618.34
-365.1   54.92
 324.2   54.07
 72.5   527.34
-193.2    9.37
```

printPoints()

This function should print out the points in a format similarly as to the following:

Points
Point 1: (x,y) = (65.34 , 618.34)
Point 2: (x,y) = (-365.1 , 54.92)
Point 3: (x,y) = (324.2 , 54.07)
Point 4: (x,y) = (72.5 , 527.34)
Point 5: (x,y) = (-193.2 , 9.37)

quickSortPoints()

This function should sort the points, either in X or Y direction. The latter can be specified using a pointer to a *Point* class member function to point to the *getX* and *getY* functions, respectively. You will need to call a partition function, which you also need to provide. Please, name that function *partitionPoints()*.

partitionPoints()

This function should perform the partition that is necessary for quicksort.

releaseMemory()

Finally, this function should release any dynamically allocated memory.

Sample execution:

The output of your program for the following input data set should look as follows:

Number of Points: 5

x = 65.34
y = 618.34

x = -365.1
y = 54.92

x = 324.2
y = 54.07

x = 72.5
y = 527.34

x = -193.2
y = 9.37

Points
Point 1: (x,y) = (65.34 , 618.34)
Point 2: (x,y) = (-365.1 , 54.92)
Point 3: (x,y) = (324.2 , 54.07)
Point 4: (x,y) = (72.5 , 527.34)
Point 5: (x,y) = (-193.2 , 9.37)

Points sorted in X-direction
Point 1: (x,y) = (-365.1 , 54.92)
Point 2: (x,y) = (-193.2 , 9.37)
Point 3: (x,y) = (65.34 , 618.34)
Point 4: (x,y) = (72.5 , 527.34)
Point 5: (x,y) = (324.2 , 54.07)

Points sorted in Y-direction
Point 1: (x,y) = (-193.2 , 9.37)
Point 2: (x,y) = (324.2 , 54.07)
Point 3: (x,y) = (-365.1 , 54.92)
Point 4: (x,y) = (72.5 , 527.34)
Point 5: (x,y) = (65.34 , 618.34)

Releasing all the dynamically allocated memory and exiting....

Note:

Please submit **both** printouts of the source code you have written (preferably using *% enscript -2Gr -Pprinter filename*) and (or screen dumps of) the execution output (using *%xdpr -Pprinter*), with your

name and username clearly written on the first page of the **stapled** submitted problem set. The submitted code must be identical to that electronically turned in (as described above).
