HBM803E - Object Oriented Programming Techniques

HOMEWORK II (due to Saturday 09/01/2021 23:59)

Questions

1. **(30pts)** Suppose you want to describe the position of a point on the screen or a location on a map relative to some origin. One way is to state the horizontal offset and the vertical offset of the point from the origin. Traditionally, mathematicians use the symbol **x** to represent the horizontal offset and **y** to represent the vertical offset. Together, x and y constitute ***rectangular coordinates*** (see Fig. 1).

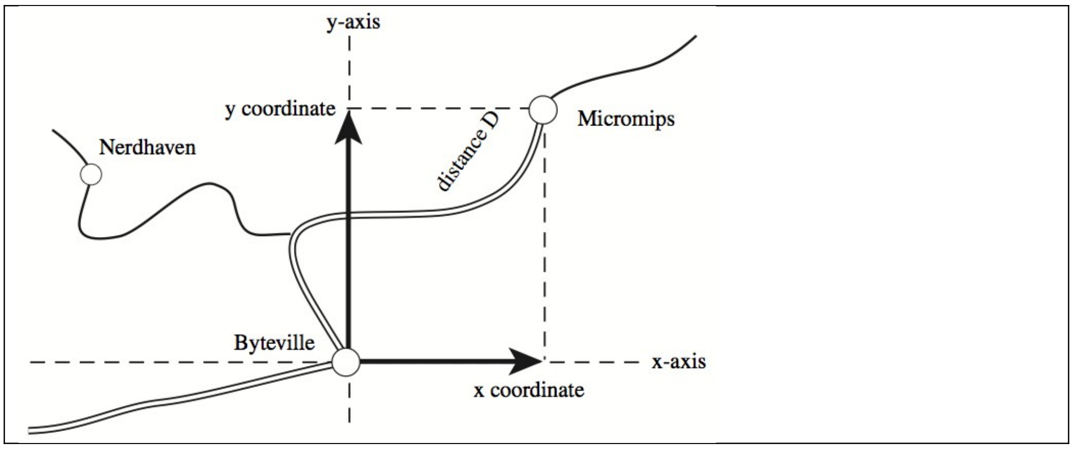

Figure 1 Rectangular coordinates of Micromips relative to Byteville.

A second way to describe the position of a point is to state how far it is from the origin and in what direction it is (for example, 40 degrees north of east). Traditionally, mathematicians have measured the **angle** counterclockwise from the positive horizontal axis. (see Fig. 2) The **distance** and **angle** together constitute polar coordinates. You can define a second structure to represent this view of a position:

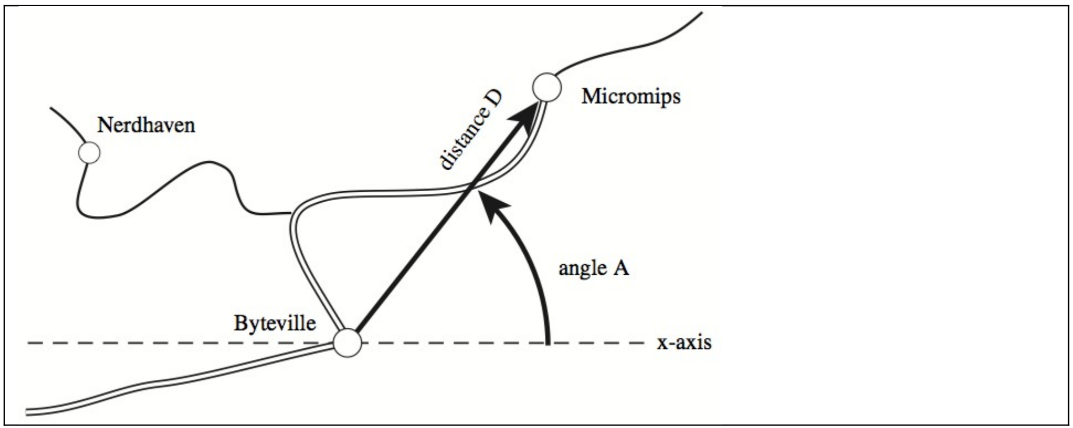

Figure 2 Polar coordinates of Micromips relative to Byteville.

Define a base class **Coord** and two distinct derived classes: **Rect** and **Polar** with following attributes and member functions.

Coord:

Attributes: origin and **name**, both being char arrays
Member Functions: set_origin() function to set origin attribute, **set_name()** function to set name, **show()** function to show origin and name attributes

Rect:

Attributes: x and **y** (in double data type) coordinates of the point
Member Functions: show() function to print out x and y values

Polar:

Attributes: distance and **angle** (in double data type)
Member Functions: show() function to print out distance and angle values, **getx()** function to calculate x using distance and angle, **gety()** function to calculate y using distance and angle
Rect and **Polar** classes must support **+** and **–** operators. The conversion from Rect to Polar and Polar to Rect must be supported via type casting operator.

Test the code (both Polar and Cartesian) using well-known conditions. For example, define (1,1), (-1,1), (-1,-1) and (1,-1) points in Rectangular Cartesian coordinate system and convert them into Polar coordinates. The results must be $\sqrt{2}$ for the distance and 45, 135, 225 and 305 degree for the angles. The student should also test the + and – operations.

Hints:

$$r=\sqrt{x^2+y^2}$$

$$\theta=\begin{cases} atan\left(\frac{y}{x}\right) if\ x>0 \\ atan\left(\frac{y}{x}\right)+\pi\ if\ x<0 \wedge y\geq 0 \\ atan\left(\frac{y}{x}\right)-\pi\ if\ x<0 \wedge y<0 \\ \frac{\pi}{2} if\ x=0 \wedge y>0 \\ \frac{-\pi}{2} if\ x=0 \wedge y<0 \\ 0\ if\ x=0 \wedge y=0 \end{cases}$$

$$x=r\cos(\theta)\quad y=r\sin(\theta)\quad \pi=4\,atan(1)\quad deg2rad=\frac{\pi}{180}\quad rad2deg=\frac{180}{\pi}$$

2. **(20pts)** Create a function called swaps_arr() that interchanges the values of the two array arguments sent to it from index 0 to N (a third argument of type integer). Make the function into a template, so it can be used with all numerical data types (char, int, float, and so on). Native behavior of the assignment operator "=" can be used to cast a numerical data type into an other. Write a main() program to exercise the function with several types.

Hint: It should be able to swap a double array and an int array, for example.

3. **(25pt)** Read a dense matrix from the input file, count for the number of non-zero elements (NNZ), allocate memory for three arrays that will be used, take the iterator to the beginning of the file and read and store the matrix with 2 integer and 1 double arrays denoting positions and values of the non-zero elements (similar to the coordinate list format COO). At the end, write and output file in the specified format.

Input:
The input file has number of rows and columns in the first line (integers) and the elements of the matrix in the next entries. An example is below:

```
4 3
1.2 3.2 0
0 0 0
0 3.43 0
0 0 -0.12
```

Output:
The output file should have dimension of the matrix (row and column number) in the first line, followed by the non-zero entries as
[row index] [column index] [value] like the example (produced by the input file):

```
4 3
1 1 1.2
1 2 3.2
3 2 3.43
4 3 -0.12
```

Your program should work fine when used with q3-input.txt in the ninova.

4. **(25pt)** Build a class Matrix_COO and perform the reading operation in the third question inside a constructor. That constructor should take a string (or const char *) as argument, open the file with that name (if it cannot, then throw an exception), be sure that first two entries are integers (for this, read them as double and check the difference between *number* and *(int) number*) and throw exception if they are not integers, than read n*m entries (throw an exception if EOF comes before n*m entries) and count the NNZ and allocate memory to the arrays, take the file iterator to the beginning and read and store non-zero entries in COO format.
Add a show() method to this class, and use it to print matrix in the output format in question 3 to the standart output (screen, with cout) **[5 pts, except throw statements]**

Use the constructor of this class in a try block, put " cout<<"successfully read"<<endl; " line after constructor in the try block, and use catch blocks to catch exceptions if they are thrown by the constructor. Exceptions may be integers or strings (or const char *), you can use any exception that works for three above-mentioned exceptional case (cannot open file with given input name, first two entries not being integers and EOF comes before n*m entries) , but be sure that at least 2 different catch blocks can be triggered in case of matching exceptions, so use at least 2 different type of exceptions. **[15 pts, including throws]**

Make sure that your destructor in the Matrix_COO class doesn't rise a run-time error if any memory is not allocated to the array (pointers) in the constructor because of an exception. For that purpose, you can use " pointer=NULL " in the constructor before any exception throwing, and check if any memory is allocated to the pointers as " if(pointer != NULL) delete[] pointer; " in the destructor. **[5pt]**

Your program should work fine with q3-input.txt, and trigger adequate exceptions when used with q4-input-exc2.txt and q4-input-exc3.txt (all this files are uploaded to ninova with this questions sheet.)

Rules

- As with all programs in this course, your code should contain useful comments. In particular, your name, the date, how to compile code and a brief description of what the program does should appear at the top of your source file.
- Each question should have its own folder (named with q1, q2, q3 etc.), which contains source code and input output files and extra explanations if any.
- All files must be packed and compressed with following naming convention,
  hw-02-[STUDENT-ID].tar.gz
  The created file must be uploaded to
  http://ninova.itu.edu.tr