National University of Computer & Emerging Sciences

Karachi Campus

Data Structures (CS201)

Programming Assignment #1

Instructions ***Each Problem of 15 points***

Due Date: The assignment is due by 10th September 2008, 11:59 PM
Submission: The assignment has to be submitted via e-mail. You must submit the source code as well as the executable file. You should mail the assignment to DS201@nu.edu.pk, preferably from the institute mail account. With subject like: "Assignment # 1 – 07-0033"
Sample Input and Output files: The sample input and output files are available from Khi website in course section. Make sure that you have tested your programs against all the available input files and EXACT output file is produced.

Problem # 1 Abstract Shapes

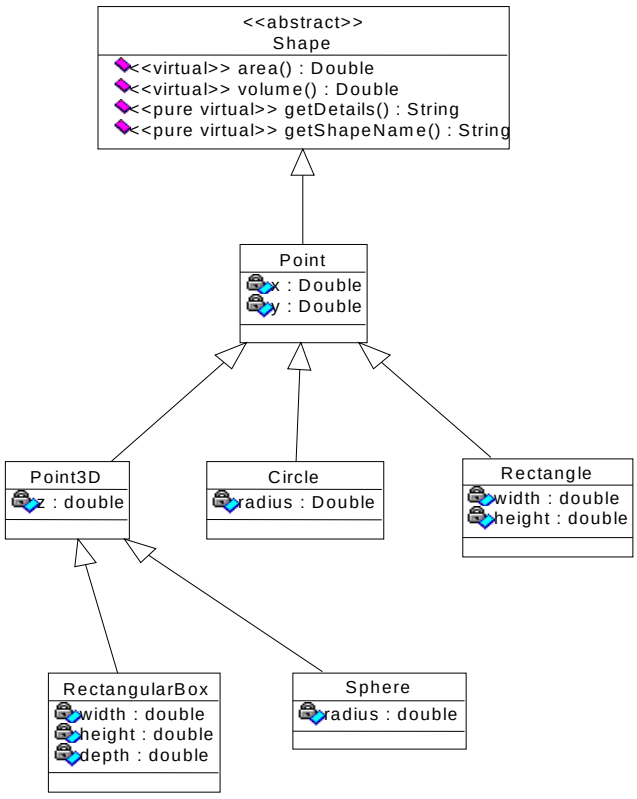
Consider the following class diagram in UML. Class Shape is an abstract base class. While all the derived classes will have to implement the *getDetails()* and *getShapeName()* as they both are pure virtual functions. The default implementations of area and volume have to be provided by the Shape class.

The 3-dimentional shapes – sphere and rectangular box will return their surface area as a result of a call to *area()*. Also, the stream insertion and stream extraction operators have to be overloaded.

Input: The input file will consist of specifications for various shapes on separate line, like:

- **C 5 7 9** – means a circle with center at (5,7) and a radius of 9
- **R 8 9 12 6** – means a rectangle with top left co-ordinate (8,9), width 12 and height 6
- **S 4 8 3 6 –** means a sphere with center at (4,8,3) and a radius of 6
- **B 3 5 12 16 8 4** – means a rectangle with top left co-ordinate (3,5,12), width 16, height 8 and depth 4

Output: The output file will contain the return values of the above 4 function calls EXACTLY in the same order the objects were entered.



Main Function: You have to make objects of these shapes and then point them with a pointer of Shape. After all of the objects have been read, you will have to make the *getShapeName(),* getDetails*(), area() and volume()* calls for each of them.

getShapeName(): This function should output the shape name in small letters. The name of the shape is same as the name of class in the above diagram.

getDetails(): This function will output the details about the shape. This is exactly the details present about the shape in the input file minus the first character.

Output: The output file will contain the return values of the above 4 function calls EXACTLY in the same order the objects were entered.

Problem # 2 Block Parity Generator

You are already familiar with parity in one-dimension, which is used to detect error in communication. There are two types of parity even or odd. For example if we have a bit of array like: 1011000 an even parity would add one more 1 at the end to make it even number of ones. Similarly, for odd parity we would add a 0 to the end to keep the number of 1s to odd. In block parity you are required to take parity in both directions i.e. in rows and in columns. Consider the example given below:

1 0 1 0 1 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 1 1 1 1 1 0 0 0	1 1 1 0
1 0 1 1 1 1 0	1

Input: The input file contains the block for which parity to be calculated. The first line tell you the dimensions of the block e.g.

4,7
1 0 1 0 1 0 0
1 1 1 1 1 1 1
0 0 0 1 1 0 1
1 1 1 1 0 0 0

Output: The output file will create a block parity representation matrix for the input read from the input file.

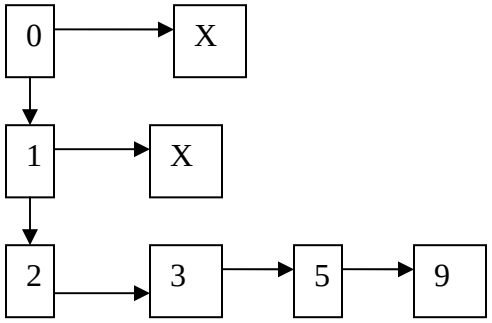
5,8
1 0 1 0 1 0 0 1
1 1 1 1 1 1 1 1
0 0 0 1 1 0 1 1
1 1 1 1 0 0 0 0
1 0 1 1 1 1 0 1

Problem # 3 Sparsely Store

A very large matrix which contains very few non-zero values are very frequent in computer graphics and such other application in computers. One Fastian, has device a very innovative method of storing such a matrix, He has called this technique Sparsely Store. The suggested approach only store non-zero values, thus we need to store row/column index for each non-zero element of our matrix. We can traverse our sparsely store in a row-major or column major fashion; we will actually have two arrays of arrays, which share elements. This is easiest to see from an example. Consider the following two dimensional matrixes

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	3	5	0	0	0	0	9	0	0
3	0	0	0	0	0	0	0	11	0	0
4	0	0	0	0	0	12	0	0	0	22
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	13	0	0	0	0
7	0	0	14	0	0	0	0	0	17	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	19

It is a 10 X 10 matrix and only 10% of elements are non-zero, we can represent this matrix in the suggested method as



This is the data array for the matrix which contains the data for each row. We also need information for each non-zero column, for this purpose we need similar information.

Input: You read the input from the file; the first row contains the matrix order and the second row contains the data of the matrix.

5,6
0 0 0 0 0 0
0 0 1 0 0 0
0 2 0 0 0 3
0 0 0 0 0 4
0 0 0 1 2 6

Output: The output file contains the two arrays, first the column information and then the data.

<The End>