Assessment Task Information (In-College & Remote delivery)

Key details:	
Assessment title:	**Practical Programming Assignment 2**
Module Name:	Object Oriented Programming
Module Code:	IY4101
Teacher's Name:	Andrew Bradley
Assessment will be set on:	3rd March 2025
Feedback opportunities:	
Assessment is due on:	Sept Cohort: 11th April 2025, Jan Cohort: 4th July 2025
Assessment weighting:	50%

Assessment Instructions

What do you need to do for this assessment?

Task:

In this assignment, you are going to design and implement classes and use them to write and modify programs. Marks will be awarded for a correct answer that shows good understanding of the C++ concepts: classes, data types, problem solving, UML design and inheritance.

Specification

You will develop and implement a set of classes that represent geometrical shapes. The system will have a console application that will use the shapes to perform geometrical calculations.

You will be given the following material to start your work:

- the UML design of Coordinate, Shape and ShapeList classes
- specification of Triangle, Rectangle, Square and Circle
- testcases

Your task is to finish this work and produce a program that manages a list of Shapes and performs a set of operations over them as described by the ShapeManagement Class

Coordinates Class

You will start by implementing a class called Coordinates, which is used to represent two-dimensional points on the canvas. The class is defined as follows:

Coordinates
-x:int -y:int
+Coordinates(x:int,y:int) +getX():int +getY():int +distance(p:Coordinate):double +translate(dx:int,dy:int):void +scale(factor:int, sign:boolean):void +display():String

- `x` and `y` are the coordinates of the point. `x` and `y` are both positive because unlike the Cartesian coordinate system the origin (0,0) is at the top left corner.
- The `distance` method calculates the distance from this point in the coordinate system to another point `p`. The distance between two points is given by the equation:

$$distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

  Where (x_1, y_1) are the coordinates of the first point and (x_2, y_2) the coordinates of the second point.

- The `translate` method simply adds the value of `dx` to the `x` coordinate, and `dy` to the `y` coordinate of the object.
- The `scale` method multiplies or divides both x and y by the value of factor depending of the value of sign (true is used for multiplication and false for division).
- The `display()` method returns the string "X = x, Y = y" where `x` and `y` are the values of the corresponding attributes of the object of the class

Shape Class
The shape class is defined as:

Shape
-position:Coordinates -sides:int
+Shape(noOfSides:int,coord:Coordinates) +getCoordinates():Coordinates +getSides():int +setCoordinates(newcoord:Coordinates):void +translate(dx:int,dy:int):void +scale(factor:int, sign:boolean):void +getArea():double +getPerimeter():double +display():String

In addition to the getter and setters, the shape class has methods that will be overridden on the subclasses to calculate the area and perimeter of the shape and to translate and scale the shape.

Rectangle, Square, Triangle and Circle Class

You have to design the UML class association diagram of all these shapes as subclasses of the class Shapes

Rectangle

- New attributes: width and length
- Constructor sets the coordinates and the width and length of the rectangle
- area = width * length
- perimeter = 2*width + 2*length

Square

- New attributes: side
- Constructor sets the coordinates and the side of the square
- area = side *side
- perimeter = 4*side

Circle

- New attributes: radius (r)
- Constructor sets the coordinates and the radius of the circle
- area = πr^2

- perimeter = 2πr

Triangle

- New attributes: Coordinates of the three Points that correspond to the three vertices of the triangle
- Constructor receives the three vertices as parameter
- Perimeter is the sum up of the distances between the three vertices (a+b+c)
- To translate a triangle, the vertices must be moved
- To calculate the area of a triangle given the coordinates of its vertices, you can use the Heron's Formula (see http://mathworld.wolfram.com/HeronsFormula.html)

$$area = \sqrt{s(s-a)(s-b)(s-c)}$$

Where a, b and c are the distances between the three vertices and s is the semiperimeter:

$$s = \frac{a+b+c}{2}$$

The `translate` method will be inherited from the Shape class for the majority of the classes (except Triangle) as all the shapes translate by calling the equivalent methods from the Coordinates class.

The `scale` method will need to be overridden in all the classes. The scale method multiplies or divides by the factor all the features of a shape (radius, vertices, centre, length, width).

The `display` () method of each class returns a string with the name of the shape, attributes of the shape and area and perimeter.

ShapeList Class

The class ShapeList stores a list of shapes and requires the following methods:

- Add Shape: add a shape to the list of shapes.
- Delete a shape: given a position on the list the method removes that shape from the list. Method must check that there is a shape in that position
- Get shape: given a position the method returns the shape in that position. Method must check that there is a shape in that position
- Translate: given a xdistance and a ydistance, the method translates all the shape.
- Scale: given a factor and a sign all the shapes will be scaled.
- Get number of shapes: returns the number of shapes on the list
- display: return a string will information about all the shapes

ShapeList
-listofShapes:
+addShape(s:Shape):void +translateShapes(dx:int, dy:int):void +getShape(pos:int):Shape +removeShape(pos:int):Shape +area(pos:int):double +scale(factor:int, sign:boolean):void +perimeter(pos:int):double +display(): String

ShapeManagment Class

This main class consists of a menu that calls all the different functionalities as listed below:

1: add a shape
2: remove a shape by position
3: get information about a shape by position

4: area and perimeter of a shape by position
5: Display information of all the shapes
6: translate all the shapes
7: scale all the shapes
0: quit program

Test Plan

Perform the following actions and describes whether the program has behaved as you expected. You can use this table or use another format but for each action you must say what you expected from the program and the actual result that you observed (need evidence) when that test was run.

Action	Expected result	Actual result
Create a triangle whose vertices have the coordinates (50,50), (20,70), (70,70) and add it to the list of shapes		
Create a Rectangle in position (100, 20) width 10, length 15 and add it to the list of shapes		
Create a circle in position (80, 100) and radius 25 and add it to the list of shapes		
Create a square in position (90, 40) witd side 20 and add it to the list of shapes		
Add three other shapes of your choice here		
Show area and perimeter the second shape of the list		
Display information of all the shapes		
Remove the third shape and check that no error occurs		
Translate all the shapes by 10 for coordinate x and 15 for coordinate y		
Display information of all the shapes		
Increase all the shapes by a factor of 2		
Display all the shapes and see if they have changed in position and dimensions		
Add more Actions		
Add actions for error handling such as: removing a shape in position 20, display the area of the shape in position 100		

Structure:

- You need to complete the design by using UML and class association diagrams.
- You must implement all the hierarchy of shapes using inheritance properly.
- You must implement all the classed described in the specification
- You must complete the implementation of the main class (ShapeManagement)
- You should carry out testing of your program using the provided actions. You should not mark a test as having been passed unless it has actually been passed.
- The program should use standard C++ naming conventions (for example variable names must be meaningful nouns or noun phrases, and should be written using camel case).
- You have to produce an UML diagram that shows the relationships between all the classes.
- You should create a well-structured C++ program that uses the concepts that we have learned up to this point and makes sensible use of comments.
- You should use comments to explain each one of the methods that you have written.

- It is strongly recommended that you use version control management during the implementation of this program

Theory and/or task resources required for the assessment:

- Lippman, S. B., Lajoie, J., & Moo, B. E. (2005). C++ Primer. 4th Edition. Addison Wesley Professional. ISBN 9780201721485
- Perry, Greg M.; Miller, Dean, C. (2013). C Programming Absolute Beginner's Guide. 3rd edition. ISBN 978-0789751980

Referencing style:
You **may** include a Harvard style reference list at the end of your report. A full bibliography is NOT required.

Expected word count:

You are expected to write between 500 and 1000 words excluding the implementation (C++ code), following the specific structure outlined above and in the submission requirement**.**

Learning Outcomes Assessed:

- Use the basic programming constructs of C/C++ to manipulate various datatypes, such as arrays, strings, and pointers.
- Manage memory appropriately, making use of memory allocation/deallocation procedures.
- Apply the Object-Oriented Programming approach to software problems in C++.
- Identify, isolate, and fix common errors in C/C++ programs
- Write small-scale C/C++ programs using skills developed during the course.
- Develop and use test plans.
- Implement basic source control management e.g. git.
- Explain the ethical issues around open-source software, as well as trustworthy and secure software development

Submission Requirements:

The work you submit should comprise two parts: a Microsoft Word document or pdf file and a zip file with the C++ program file(s). The report should contain the design with the UML diagram, and the results of running your program with an explanation of how it has been tested (include result of tests given in specification) and all the C++ source code copied and pasted in the appendix.

How to avoid academic misconduct

You should follow academic conventions and regulations when completing your assessed work. If there is evidence that you have done any of the following, whether intentionally or not, you risk getting a zero mark:

Plagiarism & poor scholarship

- stealing ideas or work from another person (experts or students)
- using quotations from sources without paraphrasing and using citations

Collusion

- working together with someone else on an individual assessment, e.g., your work is corrected, rephrased or added to by another (both parties would be guilty)

Buying or commissioning work

- submitting work as your own that someone else produced (whether you paid for it or not)

Cheating

- copying the work of another student
- using resources or aids that are not permitted for the assessment

Fabrication

- submitting work, e.g., laboratory work, which is partly or completely made up. This includes claiming that work was done by yourself alone when it was actually done by a group

Personation

- claiming to be another student and taking an assessment instead of them (both parties guilty)

Specific formatting instructions:

You must type your assessment in Arial font 11, with single spacing.

You must submit the assessment electronically via the VLE module page. Please ensure you submit it via Turnitin.

Assessments submitted after the submission deadline may incur penalties or may not be accepted.

Addition submission information – check you have done the following:	
Formatting	Consistent font, spacing, page numbers, formatting and subheadings
Citations	Correct format and location throughout the report
Spell check	Spell check the report
Proof-reading	Proof-reading completed
Grammar	*Grammarly* has been used to check the report

How will this assessment be marked?

The assessment will be marked using the following areas and weightings:

- Design of the program. The design of your program using Class diagram and relationship between them (25%)
  - UML class diagram and relationship (15%)
  - Design of the solution (10%)
- Description of how you tested the program and explanation of the results. (15%)
  - Clear description of problems encountered
  - Description of how you have solved the problems or unable to solve them
- Implementation (60%)
  - Correct implementation of hierarchy of the Shape class and the four subclasses with appropriate constructors and overriding of methods. [20%]
  - Implementation of the option that displays all the shapes in the list [10%]
  - Implementation of the ShapeManagement class [10%]
  - All the shapes translate and scales as expected [10%]
  - Programming style [5%]
  - Testing and Results [5%].

You will receive a % mark in each of these categories. The overall mark will be a percentage (0-100%).

How will you get feedback?

Your tutor will mark the assessment and provide you with a written feedback sheet. You can use this feedback to guide your further learning on the module.