1801ICT Object-Oriented Programming
School of ICT, Griffith University
Gold Coast Campus, Trimester 2, 2019

Project 3 on Object-Oriented C++ Programming

Release 25.08.2019 Due 12.09.2019 23:59

Total Marks Allocated: 14

1 Answering Questions

Week 7 is Gold Coast show holiday. So only 1 mark in week 8. Make groups of 3 to 4 students. Lab instructors will ask you to discuss a number of exercise questions (which ones that will be told in the lab) among yourselves and then ask questions verbally to check your understanding. Some small formative programming taks may also be given. Marks will be awarded based on your satisfactory performance.

2 Programming Problems

- All programs should be submitted online by the due date and time. You can get your programs marked anytime in week 8, but week 9 lab is dedicated for marking programs on first come first serve basis.
- If you need any help, please ask the instructors in the lab; they can give you hints and suggestions.

2.1 My Geometric Shapes

We consider two-dimensional geometry. Each geometric shape has an anchor point (x, y) and a symbol s that is used to draw the shape. Moreover, each type of shape has its own attributes as describe below.

- A point has no other attributes; just the anchor coordinate and the symbol as each shape has.
- An ellipse has a the x-radius and b the y-radius. An ellipse becomes a circle if $a = b$.
- A polygon has n the number of sides and l the length of each side, e.g. trigon (3), tetragon (4), pentagon (5), hexagon (6). Note we are considering only those with all the sides being equal.
- A line has another point (x', y') as its target point. You can use as many lines as you want to create various other regular or irregular shapes.

You are supplied a `screen.cpp` C++ program that has a class named `screen`. I wrote the program to simulate graphics on text mode with a window of size (40×40) characters having its origin at the center. So the min and max coordinates in both axes are respectively -20 and 20 and only integer coordinates are allowed. That class supports drawing a point, an ellipse, a polygon, and a line. You need not look at the private section of the class. You need not look at the implementation of the class either. Just make yourself familiar with the method prototypes in the public section. Then look at the main function in the program. You can compile the program and run it. You can change the main program as you wish to play with the methods of the class, to draw more shapes of these four types. Note that drawing of the shapes on the simulated screen is far from being perfect. You are free to improve it, without loosing its current capabilities. We will just use the `screen` class.

2.2 Task 1: My Template Painter

We will create classes and templates in this task.

1. Write four classes `point`, `ellipse`, `polygon`, and `line` adding appropriate attributes as describe in my geometric shapes. For each of these classes, write the required essential methods, the getter and setter methods, and a draw method. The `draw` method will involve on screen drawing of the respective shape.

For drawing the shapes on screen, you will use the simulated screen class as described as part of my geometric shapes. Copy-paste the supplied screen class in your program and use it.

2. Write a template class named `bunch` that can keep maximum 32 items. This will be a simple container class, implemented using C-style arrays, statically or dynamically created. The `bunch` class will have a getter to know the number of items and an overloaded [ ] operator with an index to provide read access to each item. The `bunch` class will have just an add method to add an item at the end of the array and a remove method to remove the last item. This class is a simple exercise of having a template class instead of a regular class. So basically the integer stack class that you implemented in project 2 can be copy-pasted and converted into a template class and that will do.

3. Write the main function of your program with four `bunch` objects: `bunch<point>`, `bunch<ellipse>`, `bunc<polygon>`, `bunc<line>`. It will then provide a menu to draw a point, an ellipse, a polygon, and a line. As inputs are taken, each type of shape will be added to its own `bunch`. The menu will also include a list item to show the list of the shapes that have been taken as input so far. The menu will also include a display item where the type of an item and the index of an item will be given and the corresponding shape will be drawn on screen; if the number of items in the list is given as the index, then all the shapes in the list will be drawn on screen. You might have a menu option to draw all shapes of all types.

4. Write the main function of your program, which will define an array of shape pointers. It will then provide a menu to draw a point, an ellipse, a polygon, and a line. The menu will also include a list item to show the list of the shapes that have been taken as input so far. The menu will also include a display item where the index of an item is given and the corresponding shape will be drawn on screen; if the number of items in the list is given as the index, then all the shapes in the list will be drawn on screen.

5. Draw some figures e.g. human face, some drawings using points, ellipses, polygons, and lines. Include a menu item to store the list of shapes in a file and then another menu item to load from the file.

Marks: 7 marks in total. Correct implementation of classes 2 marks, correct implementation and use of the template class 2 marks, Implementation of the main function 2 marks, and file read/write 1 mark.

2.3 My Polymorphic Painter

We will redesign the classes from my template painter using inheritance. Also, we will use polymorphism to have class specific versions of the `draw` method and just one `bunch` to hold all types of geometric shapes.

1. Write an abstract `shape` class with attributes (x, y) for the anchor, and s for the symbol. The `shape` class will have a pure virtual method named `draw`, that will have no parameter. Also, make the destructor of the shape class pure virtual. Then, deriving from the `shape` class using inheritance, write four concrete classes `point`, `ellipse`, `polygon`, and `line` adding appropriate attributes, if needed, to the respective classes, as described above. For each of these concrete classes, write the required essential methods, the getter and setter methods, and of course override the `draw` method. The `draw` method will involve on screen drawing of the respective shape and you will use the supplied simulated screen class as described above.

2. Like the virtual `draw method`, you can have virtual `display` method in each class to display the attributes of each object. Also you can have another virtual method to save the information of each shape to a file.

3. Use your `bunch` template here. However, instead of four bunch objects, you will use only one: `bunch<shape *>` to hold shape pointers. Note you are using the parent class pointer to store an array of child class objects here. Handle memory allocation and deallocation carefully, so that no memory leaking is possible. Similar to the template painter, the polymorphic painter program will have a menu to draw a point, an ellipse, a polygon, and a line. The menu will also include an option to draw a particular shape by providing its index, or if an index equal to the number of shapes is given, then all shapes will be drawn. Other functionalities of the polymorphic painter will be the same as of the template version. Note no switch-case or if-else or down-casting is allowed here. You have to call the polymorphic methods using the `shape` pointer. That is the purpose of using inheritance and polymorphism.

Marks: 6 marks in total. Correct implementation of class inheritance 2 marks, correct implementation of polymorphism 2 marks, correct use of overridden virtual methods and memory deallocation 2 marks.