Program submitted on time (late submission subject to as much as 100% deduction).

Use of global **variables** is cause for an immediate zero (0) on the project.

Program builds without errors or warnings 5/5

Program makes use of class interfaces, inheritance, and general OOP programming practices:

* Class interfaces 2/2
* Inheritance
  + Base class is abstract 2/2
  + Line, Ellipse, Rectangle inherit from Base class 2/2
  + Filled Ellipse, Filled Rectangle inherit from Ellipse and Rectangle 1/2
  + Additional derived types inherit from the appropriate base or derived type 2/2
  + Placement of function is correct 0/2
* General OOP programming practices
  + Program makes correct use of abstraction 1/2
  + Access to member data is correctly restricted and controlled via mutator / accessors 0/2
  + Polymorphism is used where appropriate 1/2
  + Late binding is used and used correctly 2/2
  + Class destructors are used appropriately 2/2

Program is readable, modular, well documented, and reasonably efficient, as well as correct:

* Readable
  + Descriptive variable names 2/3
  + Descriptive function names 3/3
  + Limited (no) use of obfuscated code 3/3
  + Appropriate internal comments 2/3
* Modular
  + Functions perform a single task 2/3
  + Classes are general and reusable 1/3
  + Functions interact only through interfaces 2/3
* Well documented
  + Doxygen main page describing overall program 3/3
  + Classes described 1/3
  + Class member functions documented 1/3
  + Doxygen runs without errors or warnings 2/3
  + Doxygen generates navigable documentation 3/3
* Reasonably Efficient
  + When compiled with –O is screen flicker minimal 2/2
  + Does adding 100 shapes cause noticeable performance deterioration 2/2
  + Does the window re-size smoothly with minimal flickering 2/2
  + Program doesn’t leak memory 0/2
* Correct
  + Program is named ‘paint’ 5/5
  + Program provides at least 8 colors 2/2
  + Program allows at least 5 shapes 2/2
  + Can select border color with left mouse click on palette 2/2
  + Can select fill color with right mouse click on palette 2/2
  + Can select each of the five shapes 2/2
  + Can select each of the 8 (or more) colors 2/2
  + Can draw lines 2/2
  + Can draw rectangles 2/2
  + Can draw ellipses 2/2
  + Can draw filled rectangles 2/2
  + Can draw filled ellipses 2/2
  + Re-sizing window behaves appropriately 2/2
  + Can move shapes 2/2
  + Can move lines 2/2
  + Can bring shapes to front 2/2
  + ‘d’ key deletes shapes focused shape 2/2
  + ‘c’ clears shape list and clears screen 2/2
  + ‘q’ and <esc> exit program 2/2
  + Screen is redrawn pressing other keys 2/2
  + Right mouse selects focused shape 2/2
  + Focused shape is brought to the front of overlapping shapes 2/2
  + Right mouse drag moves focused shape 2/2
  + Program is named ‘paint’ 5/5

Code Review – A place for comments from just reading through the code and noting good, bad, ugly, and interesting features. Can account for as much as 15% of the total grade. 10/15

Get up to 10 points back from the Code Review section by completing and returning Team Evaluation Form on time

Total points: 110/135 :

Comments:

* Several of your functions are leaking memory:
  + drawPaletteShapes
  + selectShape
  + constructPalette
* File headers (where present) are just the file tag and no description. Only the Shapes class has a description in the class header. Some inline comments, but they are sparse. Most functions are undocumented.
* Filled rectangle and ellipse classes properly inherit from their unfilled variants, but contain no implementation code. The draw function implementations are in the unfilled variants and have control logic to determine whether to draw a filled or unfilled shape.
  + These classes and functions are fulfilling multiple responsibilities and the work is not being properly distributed. The idea of inheritance is that each class handles a single well-defined task or responsibility which is extended or modified by subclasses.
  + Creating monolithic functions and classes makes debugging and tracing execution much more difficult. Splitting the logic into multiple functions would make the call-stack much more informative in the event of an exception.
* In this circumstance it is acceptable to place the drawing code inside the shapes themselves, however this makes the shapes specific to the interface exposed by GLUT.
  + A more general and portable approach would be to either:
    - Make the shape provide accessors that an ‘Artist’ or ‘Painter’ class knows how to consume (i.e. shapes are expected to present a list of edges in the order they are to be drawn)
    - Or, make the shape accept a ‘Canvas’ object that presents an interface for the shape to draw to (i.e. the canvas exposes a method to draw lines or fill regions)
  + Either approach would limit the need to recreate the code to a single class (the ‘artist’, ‘canvas’, etc.) rather than requiring a recreation of every shape. In the event that this was a program like the real MSPaint which has many more shapes, if we wanted to make a port to another graphics framework every single class would need to be altered.
* Similar to the above, your Shape class is responsible for tracking all of the shapes in your program. This means that the manner of storing Shapes for drawing is defined by the Shape class. Even if the class were used with another shape storage/drawing class, the static vectors will be present and could potentially be performing in ways that are not clear to other code that is consuming Shapes (that is, storing the shapes in static variables on the class leaves room for functions to have undesirable side-effects).
* This is a stylistic note, but generally your classes’ names should be singular and not plural. This is more of a convention for readability (i.e. we have one Shape, we have an array of Shapes) but is still common. It is acceptable to have plural names where the object would represent multiple things (i.e. an argument storage struct could be named ‘ArgumentsBundle’).
  + Also stylistic, class files should generally be ‘className.filetype’. This is a common practice, and some languages even enforce it. Some examples would be ‘Shape.cpp’ or ‘cube.h’.
* Your events are functions, but could be reorganized into classes. Using a class to organize/type your functions can allow you to use type-checking to determine if a callback should be accepted or not.
  + For example I could have a function ‘acceptsCallback(void (\*myFunc)(int,int))’ accepting a void function pointer expecting two integers as arguments. I could instead make a class ‘MyCallback’ that defined a pure virtual function ‘callback(int,int)’ and rewrite the function signature as ‘acceptsCallback(MyCallback myFunc)’ and later use ‘myFunc.callback’ to perform the operation.
  + With a function pointer you can perform signature based type-checking but with a Class you could add other validation based on the Class and/or what its super types are.
* Generally, when you override a virtual function you should use the ‘override’ keyword. The override keyword lets the compiler check that you are actually overriding the function and not just creating a new one.
  + For example, given a function ‘virtual void draw() const’ that we have written an override for with the signature ‘void draw()’ we have actually not overridden the draw function. The draw without the const has a different signature than the const virtual and it would be possible at runtime to use either implementation. Adding ‘override’ would flag the fact that you are trying to override a function that doesn’t exist.
* You have duplicated code, output.cpp has a definition for drawSquare but your rectangle class has a draw function that contains almost exactly the same code.

Team Evaluation

This was a significant project and unlikely to have been completed by a single individual. Team projects can sometimes be a challenge and feedback is important and useful when taken in the right context. Use this form to:

\*Get 10 points added to the Code Review section of this report by returning a digital copy to me (Dr. Hinker [paul.hinker@sdsmt.edu](mailto:paul.hinker@sdsmt.edu)) by Friday, Feb 22.

**Honestly** evaluate your effort.

Give your team mate an **honest** evaluation of your perception of their effort on the project.

Team Member Name:

Communication 1 2 3 4

Work Quality 1 2 3 4

Overall Satisfaction 1 2 3 4

My Name:

Communication 1 2 3 4

Work Quality 1 2 3 4

Overall Satisfaction 1 2 3 4

**Communication:** This person responds to e-mail/texts promptly, they clearly communicate their ideas and offer salient opinions when asked.

**Work Quality:** This person takes on responsibilities within their experience and skill set and completes them professionally on a consistent schedule. This person’s contributions enhance the product’s quality.

**Overall Satisfaction:** If I had a choice, would I work with this team member again?

Scale: 1-deficient, 2-needs some work, 3-solid marks, 4-exemplary