CISC2000: Computer Science II
Polygon Class

Objective: To *design and develop* a class in the C++ programming language that shows correct use of datamembers of type *const*, *static* and *static const*.

Assignment: Define a class in C++ whose purpose it is to represent a general polygon.

The class should be designed around the following characteristics:

1. The name of the class should be **Polygon**.

2. The class should be composed of **at least** the following (data) **members**:

    - **MAX_POLYGONS**, (static constant) *data member to indicate the maximum possible objects that can be instantiated for this class*. The assigned value should be 25.
    - **numPolygons**, (static) *data member to indicate the number of polygons physically instantiated*.
    - **length**, *data member to represent the length of the polygon*.
    - **width**, *data member to represent the width of the polygon*.
    - ~~**MIN**, (constant) *data member that represents the minimum possible value for the length and width of a specific polygon object*. Unless otherwise specified, this value should be 10.~~
    - ~~**MAX**, (constant) *data member that represents the maximum possible value for the length and width of a specific polygon object*. Unless otherwise specified, this value should be 100.~~

    Note that you may need to add additional data members to accomplish the objective of the driver program. Also, make sure that the data type of each member is appropriate to the type of data being stored.

3. The class should include **at least** the following **methods**:

    - ~~Default Constructor, initializes each data member with appropriate default values.~~
    - ~~Constructor, initializes the object with specific values for the MIN and MAX data members. Remaining data members should be initialized appropriately.~~
    - Constructor, initializes the object with specific values for length, width, ~~MIN, and MAX. Remaining data members should be initialized appropriately.~~
    - ~~Destructor, clears out all the data members and adjusts any class level data as needed.~~
    - A method to populate the data members {length, width} from external (user) input.

- Mutator methods for each data member or logical groupings, as necessary. *Note that these methods should ensure the integrity of the object and should not allow data to be saved that is not appropriate for that data member.* ~~Example, length and width must be between MIN and MAX inclusive.~~
- A method to calculate the area of the polygon.
- A method to calculate the perimeter of the polygon.

  *Note that these methods can be invoked implicitly when an object is constructed with specific values for length and width **or** explicitly by the application after the values for length and width have been set.*

- A method to display the pertinent data members {length, width, area, perimeter}.
- A method to draw a polygon of the specified length and width. *You can use any character to draw the shape (i.e. border) of the polygon*.

4. ~~The class should allow for the instantiation of constant objects. In other words applications which use this class should be able to create polygon objects of type const.~~

5. Write a simple main function to instantiate several polygon objects for the purpose of testing out the various methods of the class. This main function can be defined in the class source file, but make sure it is commented (or removed) after the class has been tested.

Important

1. Follow an incremental approach in developing the class. Compile and test the class as you implement each method. Do not wait until all the methods have been implemented before you begin testing.
2. If the methods compile but do not work as expected, consider using cout statements to display the contents of the appropriate members as well as to trace the flow of execution.

For full credit be sure to:

- Include a descriptive comment block at the beginning of the file
- Include a descriptive comment block before the defintion of each method.
- Use correct indentation and allignment.
- Use descriptive identifiers for variable names as well as appropriate data types.
- Use blank lines to separate the code into appropriate blocks.
- Include comments to help others understand your program, and help yourself think!