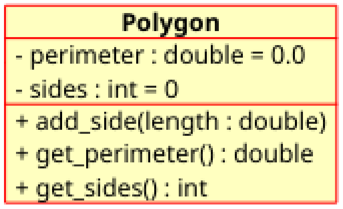
Taking Shape
Modeling a Polygon

CSE 1325 – Fall 2018 – Homework #2
Due Tuesday, September 4 at 8:00 am

Assignment Overview

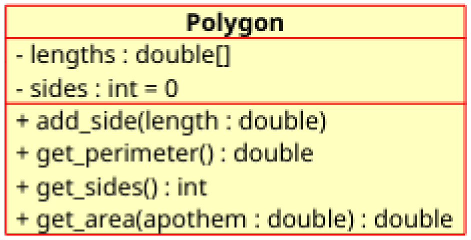
You used types in Java – boolean, byte, long, double. You used types in C – bool, int, double, char. Now it's time to create a first-class type in C++ of your very own!

Full Credit: Using git (with *at least* 3 commits):


Polygon
- perimeter : double = 0.0
- sides : int = 0
+ add_side(length : double)
+ get_perimeter() : double
+ get_sides() : int

- **Write the Polygon class in file polygon.cpp** as shown to the right.
- **Write a main() function in file main.cpp** that instances Polygon and, in a loop, accepts (via cin) the length of each side and passes that length to your Polygon instance's add_side method.
  Once a non-positive side length is entered, stream out (cout) the perimeter of the polygon *using your Polygon class instance's get_perimeter method*.

Deliver file CSE1325_02.zip to Blackboard. In **subdirectory "full_credit"**, include your full git repository, including your most recent files **polygon.cpp, main.cpp,** and screenshots **main.png** (interactive bash-based test session), and **git.png** (git log).


Polygon
- lengths : double[]
- sides : int = 0
+ add_side(length : double)
+ get_perimeter() : double
+ get_sides() : int
+ get_area(apothem : double) : double

Bonus: Using git (3+ additional commits) and a copy of the above code, **update Polygon to store the length of *each* side** in an array (from CSE1320) of doubles named "lengths". Use this array to calculate the perimeter *each time the get_perimeter method is called*. **Also add a get_area method** that accepts the apothem (a double) as a parameter, and returns the area of the (regular) polygon as a double calculated as 0.5 * apothem * perimeter.

Test your new version of Polygon with your main.cpp *from Full Credit*. Does it still work? **Include screenshot regression_test.png** of the error message(s) OR of the program working.

Finally, **update main.cpp** from your Full Credit solution to accept (via cin) the length of the apothem after the loop and then **stream out (cout) both the perimeter and the area of the polygon.** In CSE1325_02.zip **subdirectory "bonus"**, include your git repository with updated versions of **all source files** plus screenshots **main.png** (interactive bash-based test session) and **regression_test.png**.

Extreme Bonus: Using git (3+ additional commits) and the Bonus code, create another implementation of the Polygon class using a **vector** instead of an array for lengths, and modify (or add) a **Makefile** to build with either implementation. Are vectors "better" than arrays for this application?

Notes

Use Git

You are REQUIRED to use git locally (not GitHub, unless you so choose[1]) while developing this code. You will be required to show a screenshot of your git commit log at each level as part of the archive you submit to Blackboard, and will lose points if you don't. **This is not hard!**

- Type "**`git init`**" in your full_credit directory before you start developing code.
- Whenever you have a stable set of files, type "**`git add`** ***`filename`***" for each filename you want to keep. Once you've added all files, type "**`git commit -m '`*`a descriptive message`*`'`**" (replace the italicized text with your filenames and messages, of course), and save as **git.png**.
- When you're ready to submit your homework, type "**`git log`**" and take a screenshot (e.g., "**`gnome-screenshot -i`**", or you may use your host operating system's screenshot utility).
- Repeat in your bonus and extreme_bonus directories. You might consider committing the code from the previous level as the first commit of a new level – what we call a "baseline".

Use a Makefile

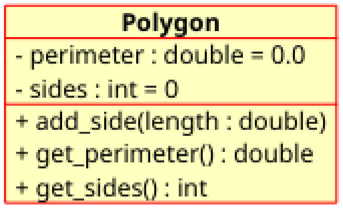
You are ALLOWED (but not yet required) to provide a simple Makefile that can build and execute your main program given the simple command "**`make`**" (or Tools → External Tools → Build in Gedit). Here's a working example to get you started. Remember: the indentations are tabs, not spaces!

```
# Suggested Makefile for CSE_1325 Homework #2 full_credit
main: main.cpp polygon.cpp
	g++ --std=c++17 -o main main.cpp
	@echo "Type ./main to run the program"
clean:
	-rm -f *.o main
```

Full Credit

Writing the Polygon Class, Step by Step

First, write a C++ class named Polygon, and store it in a file named **polygon.cpp**. The **required** class design is shown in the UML class specification to the right.


Polygon
- perimeter : double = 0.0
- sides : int = 0
+ add_side(length : double)
+ get_perimeter() : double
+ get_sides() : int

- Method "add_side" accepts double "length" as a parameter, and adds it to attribute "perimeter". It also increments "sides".
- Method "get_perimeter" simply returns attribute "perimeter".
- Method "get_sides" simply returns attribute "sides".

No error checking is required for this code. Handling errors will be covered in Lecture 04.

1 If you decide to use Github, please include a text file named GitHub.txt with a link to your public GitHub repository in CSE1325_02.zip.

Writing the Main Function, Step by Step

Write a main() function, and store it in a file named **main.cpp**.

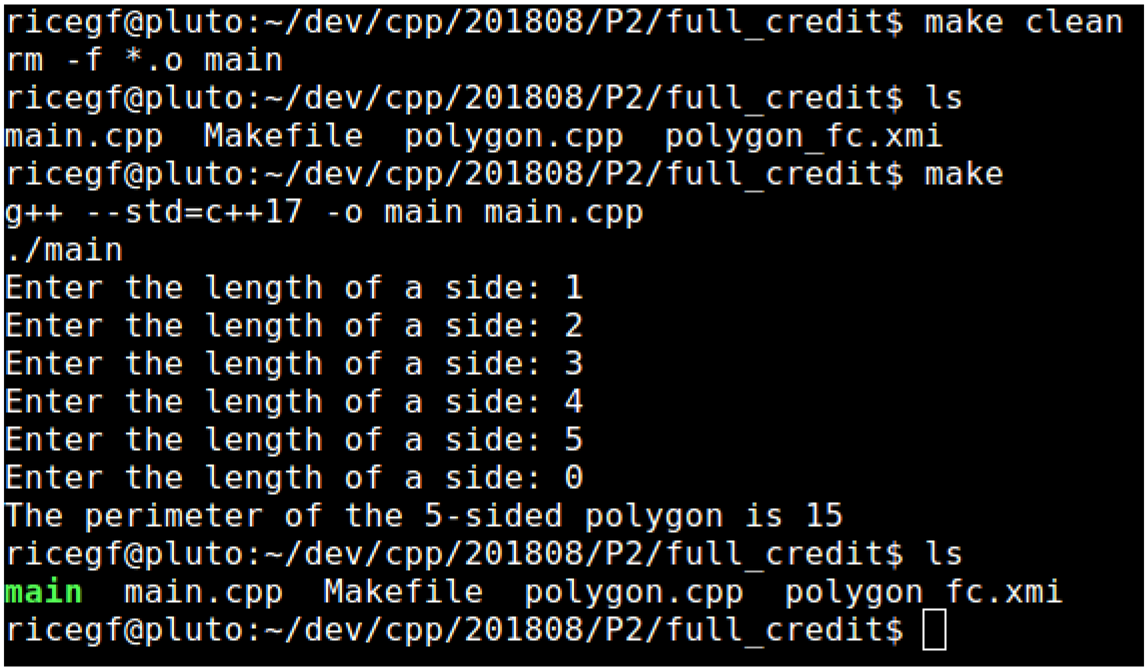
- In addition to including <iostream> and using namespace std, also include polygon.cpp. Declare your main function.
- Instance Polygon as variable named "polygon" (the name of a class instance is often the lowercase version of the class name). Also provide a temporary double variable to accept input from the user.
- Begin a loop. Each pass, prompt for the length of the next side, and read it from cin into the temporary double variable you declared above.
  - If the length of the side is less than or equal to 0, exit the loop.
  - Otherwise, pass the size as a parameter to polygon's add_side method.
- Send the polygon's perimeter **as provided by the polygon's get_perimeter method** to cout. Feel free to adorn the results with some additional text, e.g., "This 5-sided polygon has a perimeter of 25.3".

It is an egregious error to keep separate track of the number of sides that you have read, or the sum of the side lengths that you have read, in main(). It is an egregious error to try to calculate the perimeter as part of main(). **This is what the Polygon class does!** All main() needs to do is to instance Polygon, give that instance each side length, and then ask it for the polygon's perimeter. That's the point!

Deliverables

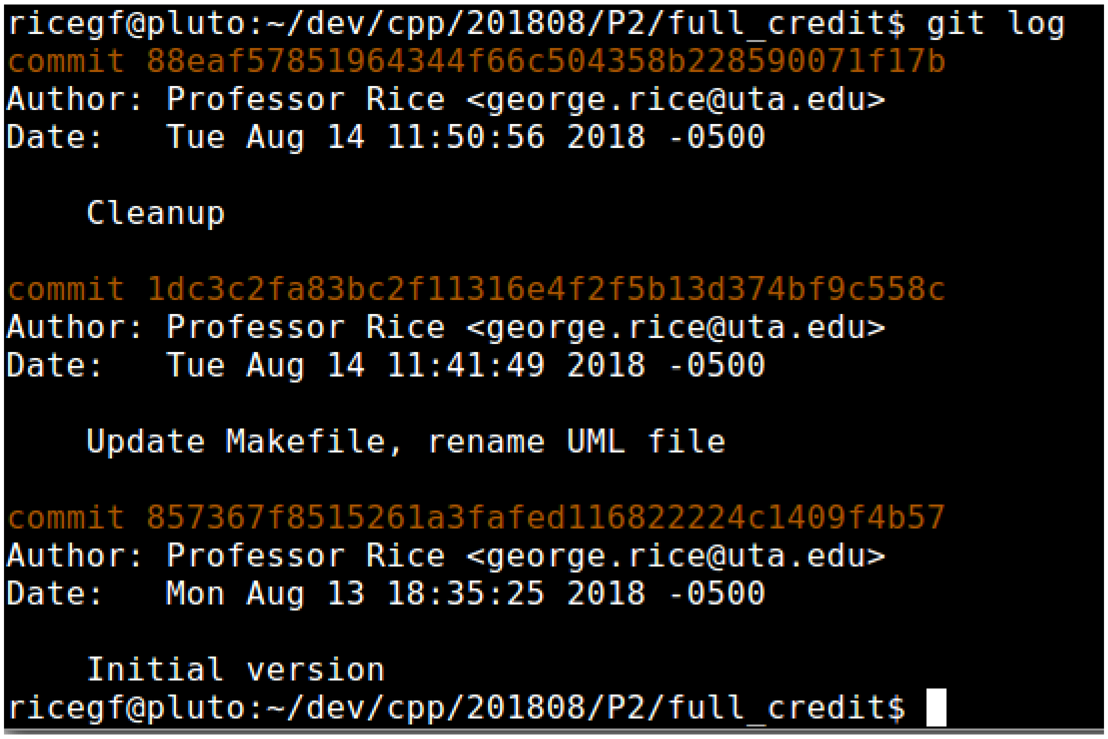
In the full_credit directory, you will provide:

- 2 C++ source files named **polygon.cpp** and **main.cpp**, respectively.
- An optional make file named **Makefile**.
- A screenshot named **main.png** of your interactive testing session similar to mine below.



```
ricegf@pluto:~/dev/cpp/201808/P2/full_credit$ make clean
rm -f *.o main
ricegf@pluto:~/dev/cpp/201808/P2/full_credit$ ls
main.cpp  Makefile  polygon.cpp  polygon_fc.xmi
ricegf@pluto:~/dev/cpp/201808/P2/full_credit$ make
g++ --std=c++17 -o main main.cpp
./main
Enter the length of a side: 1
Enter the length of a side: 2
Enter the length of a side: 3
Enter the length of a side: 4
Enter the length of a side: 5
Enter the length of a side: 0
The perimeter of the 5-sided polygon is 15
ricegf@pluto:~/dev/cpp/201808/P2/full_credit$ ls
main  main.cpp  Makefile  polygon.cpp  polygon_fc.xmi
ricegf@pluto:~/dev/cpp/201808/P2/full_credit$ 
```

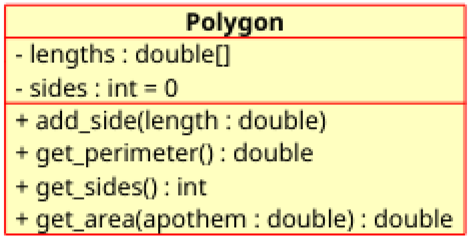
- A screenshot named **git.png** of your git log similar to mine below.



```
ricegf@pluto:~/dev/cpp/201808/P2/full_credit$ git log
commit 88eaf57851964344f66c504358b228590071f17b
Author: Professor Rice <george.rice@uta.edu>
Date:   Tue Aug 14 11:50:56 2018 -0500

    Cleanup

commit 1dc3c2fa83bc2f11316e4f2f5b13d374bf9c558c
Author: Professor Rice <george.rice@uta.edu>
Date:   Tue Aug 14 11:41:49 2018 -0500

    Update Makefile, rename UML file

commit 857367f8515261a3fafed116822224c1409f4b57
Author: Professor Rice <george.rice@uta.edu>
Date:   Mon Aug 13 18:35:25 2018 -0500

    Initial version
ricegf@pluto:~/dev/cpp/201808/P2/full_credit$ 
```

Bonus (may use some material from Lecture 03)

Continue to use git, and **update Polygon to store the length of *each* side** in an **array** of doubles named “lengths”. Use this array to calculate the perimeter *each time the get_perimeter method is called*. **Also add a get_area method** that accepts the apothem (a double) as a parameter, and returns the area of the (regular) polygon as a double calculated as 0.5 * apothem * perimeter.


Polygon
- lengths : double[]
- sides : int = 0
+ add_side(length : double)
+ get_perimeter() : double
+ get_sides() : int
+ get_area(apothem : double) : double

- Copy your current work to a new folder named ‘bonus’, e.g., **“`mkdir ../bonus ; cp * ../bonus ; cd ../bonus`”**
- Update your Polygon class (in polygon.cpp) to match your updated UML specification, using a C array as in CSE1320.
  - You will need to initialize your lengths array (it’s OK to use a static number of elements) and sides counter using direct assignment in the constructor.[2]
- Modify get_perimeter to dynamically calculate the perimeter by adding up the lengths of all of the sides in your array. (Yes, this is less efficient, but we’re making a point about class interface versus implementation here.)
- Define a method named area to accept a double parameter named apothem, and return the area of the regular polygon.[3] It isn’t necessary to do any error checking yet (we’ll cover that in a later lecture).
- Rebuild the *unmodified* main function from the Full Credit version of your program with the *updated* class specification. Note whether it builds and runs correctly, or generates errors – that is, is it “backward compatible”? Take a screenshot of your results named **regression_test.png.**
- Now *modify* your main function to request the apothem after collecting all of the side lengths, and if positive, also stream out the area of the regular polygon. You needn’t verify that the polygon is actually regular; we’ll assume smart users for this assignment only. :-D
- Take a screenshot of your interactive test as **main.png**, and of your git log as **git.png.**

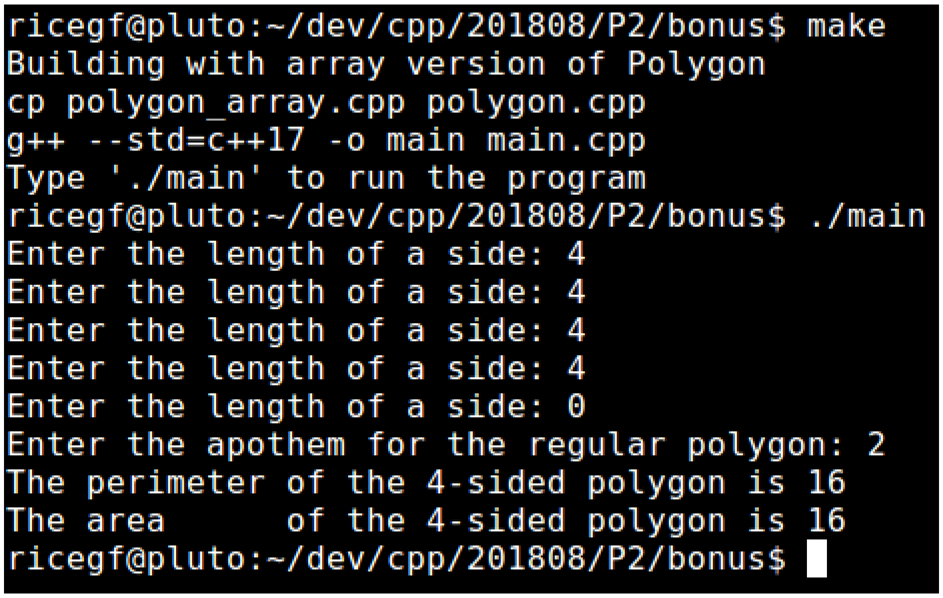
2 Remember, direct assignment places the variable after the constructor name and colon, with the value to be assigned it within curly braces – e.g, “**`Foo() : bar{0} { }`**”. This is covered in Lecture 03.

3 See https://www.dummies.com/education/math/geometry/how-to-calculate-the-area-of-a-regular-polygon/ for details.

Deliverables

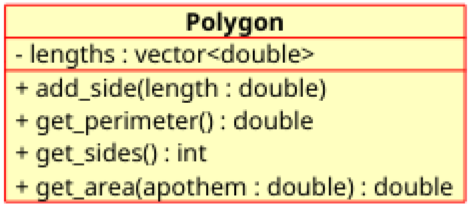
In the bonus directory, you will provide:

- **At least 2 C++ source files** named **polygon.cpp** and **main.cpp**, respectively.
- An *optional* make file named **Makefile**.
- A screenshot named **regression_test.png** of your test of your updated Polygon class with your original (Full Credit) main function, either error messages or a successful build and run.
- A screenshot named **main.png** of your interactive testing session, as before.
- A screenshot named **git.png** of your git log, as before.



```
ricegf@pluto:~/dev/cpp/201808/P2/bonus$ make
Building with array version of Polygon
cp polygon_array.cpp polygon.cpp
g++ --std=c++17 -o main main.cpp
Type './main' to run the program
ricegf@pluto:~/dev/cpp/201808/P2/bonus$ ./main
Enter the length of a side: 4
Enter the length of a side: 4
Enter the length of a side: 4
Enter the length of a side: 4
Enter the length of a side: 0
Enter the apothem for the regular polygon: 2
The perimeter of the 4-sided polygon is 16
The area      of the 4-sided polygon is 16
ricegf@pluto:~/dev/cpp/201808/P2/bonus$ 
```

Extreme Bonus


Polygon
- lengths : vector<double>
+ add_side(length : double)
+ get_perimeter() : double
+ get_sides() : int
+ get_area(apothem : double) : double

Repeat the Bonus, but use a vector (covered in Lecture 03) instead of an array for the lengths attribute. Modify (or add) a Makefile that can build using either the array or the vector version of the Polygon class.

Initialization isn't needed for a vector, by the way, because it initializes itself with a "default constructor" when defined. The vector class also tracks how many elements have been added automatically, so you don't need a separate size attribute.

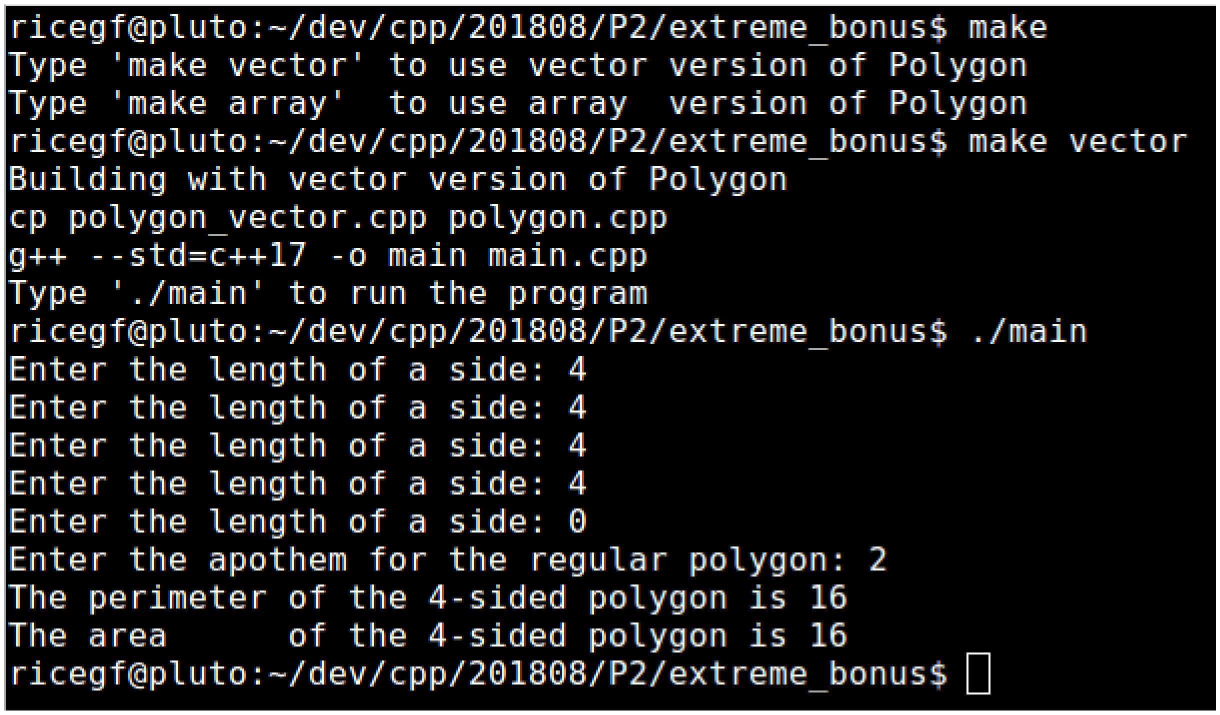
Compare the two versions of Polygon. Is a vector more useful for managing data collections than an array? Why or why not?

You do NOT need to recompile using the Full Credit main, by the way – we made that point in Bonus. Thus, you needn't deliver a regression_test.png screenshot.

Deliverables

In the extreme_bonus directory, you will provide:

- **At least 3 C++ source files** named **polygon_vector.cpp, polygon_array.cpp** and **main.cpp**, respectively.
- A *required* make file named **Makefile**.
- A screenshot named **main.png** of your interactive testing session, as before.
- A screenshot named **git.png** of your git log, as before.



```
ricegf@pluto:~/dev/cpp/201808/P2/extreme_bonus$ make
Type 'make vector' to use vector version of Polygon
Type 'make array'  to use array  version of Polygon
ricegf@pluto:~/dev/cpp/201808/P2/extreme_bonus$ make vector
Building with vector version of Polygon
cp polygon_vector.cpp polygon.cpp
g++ --std=c++17 -o main main.cpp
Type './main' to run the program
ricegf@pluto:~/dev/cpp/201808/P2/extreme_bonus$ ./main
Enter the length of a side: 4
Enter the length of a side: 4
Enter the length of a side: 4
Enter the length of a side: 4
Enter the length of a side: 0
Enter the apothem for the regular polygon: 2
The perimeter of the 4-sided polygon is 16
The area      of the 4-sided polygon is 16
ricegf@pluto:~/dev/cpp/201808/P2/extreme_bonus$ 
```