Assignment 3: Simple Classes / Objects

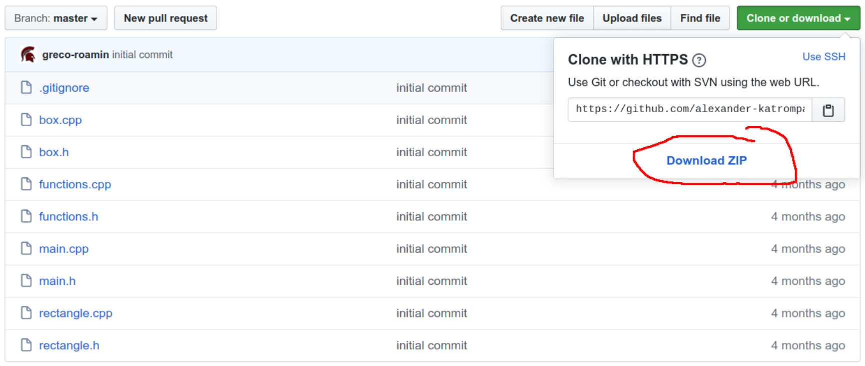
Description and purpose: To demonstrate (and advance) your knowledge of classes, header files, objects, compiling, version control, and GitHub. At the completion of this assignment, you should be ready to begin working on real data structures code. This assignment contains all the techniques and processes you will use throughout the semester. It should be just a review of Programming II, but if you are missing or not understanding any of this information, this is your opportunity to get up to speed.

Background

- You will need a GitHub account for this. Start with the GitHub Classroom guide, read it all very carefully.
- You will need to install git and g++ (if you have not already).
- Read this entire assignment thoroughly before you begin any coding.

Instructions:

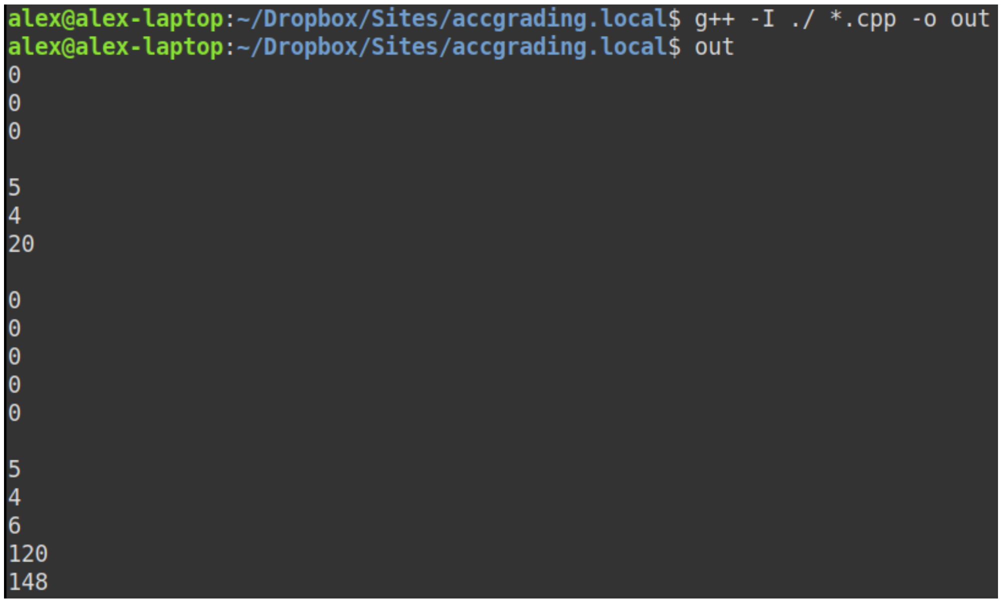
- Download the code from this repo. Place the code in an empty directory.



greco-roamin initial commit

.gitignore	initial commit	
box.cpp	initial commit	
box.h	initial commit	
functions.cpp	initial commit	4 months ago
functions.h	initial commit	4 months ago
main.cpp	initial commit	4 months ago
main.h	initial commit	4 months ago
rectangle.cpp	initial commit	4 months ago
rectangle.h	initial commit	4 months ago

- Compile the code using the command: **g++ -I ./ *.cpp**
- Run the program by typing (Mac and Linux) **a.out** or **./a.out** or (Windows) **a.exe** or **.\a.exe**, whichever is appropriate for you OS and path configuration.
- **Take a screenshot of your compilation and run results.** It will look something like this…



```
alex@alex-laptop:~/Dropbox/Sites/accgrading.local$ g++ -I ./ *.cpp -o out
alex@alex-laptop:~/Dropbox/Sites/accgrading.local$ out
0
0
0

5
4
20

0
0
0
0
0

5
4
6
120
148
```

- Study the Classes Starter code and make sure you understand the structure of the files, how header files work, how inheritance works, how pass by value and pass by reference works, etc. This is all material you should have taken away from Programming II, so this should be just a review.

Now it's time to code your own work....

- The GitHub Classroom invite is https://classroom.github.com/a/vk50TFnA
- Follow the *Assignment Specific Instructions* on the *github-classroom* page.
- When you have reached the step, "*Now you are ready to begin working,*" you will have only a **README.md** and **.gitignore**. Fix the **README.md** to be descriptive of what you are doing, something like, "*Classes and Objects refresher assignment.*" You can leave the **.gitignore** alone, it's complete and correct.
- Commit that change with the message "*updated README with relevant information.*"
- Push the changes and view them in your repo so you know everything is working correctly.
- Now create your own complete and working console application that is similar to the Classes Starter you downloaded. It is up to you to come up with a simple "toy" application like you see in the Classes Starter code. Follow these guidelines:
    - Pick any two other real world "objects" similar to 'rectangle' and 'box' that have a *hierarchical* relationship. For example, "car" and "race car" or "mammal" and "dog" or "circle" and "sphere." See below for more on this concept.
    - Create classes similar to the ones in the Classes Starter code. Inherit from one class to the other in similar fashion.
    - Use a functions module as shown in the Classes Starter for functions that act on the objects.
    - Make sure you structure your files properly (this applies to all code you will write for this class).
        - Prototypes and directives go in header files.
        - Code does ***not*** go in header files.
        - Prototypes and directives do ***not*** go in cpp files (except the *one-and-only-one* directive to #include that file's header).
        - You may not use **using namespace std;** but you may (for example) use either:
            - this form **using std::cout;**
            - or this form **std::cout << "hello world";**

          Note: the global directive **using namespace std;** will **not** be allowed for any assignment in the class as it is considered very bad practice in professional code.
    - Make sure your setters have protections in them so the object cannot be put in an invlaid state. For example, in the Classes Starter code, you cannot set height or width below zero because that makes no sense.
    - All properties (aka attributes, aka variables) of classes must be private. This goes for all attributes in all assignments and in the professional world as well.
    - All methods that are public should *need* to be public, and all others should be private.
    - Use your own setters inside your objects (especially in the constructor).
    - No getter should ever modify the object, getters are strictly for reading data.
    - Have at least one example of a method override. For example in the Classes Starter code **getArea()** is overridden in the box class, from the rectangle class.
    - Have at least one example of more functionality in the child class, for example in the Classes Starter code the box class has a **getVolume()** method which rectangle does not have, because it makes no sense for a rectangle to have volume.
    - Keep your code and application simple. It does not need to be any more sophisticated or interesting than the Classes Starter code.
    - Make a **main()** that *thoroughly* tests your application in automated fashion (do not use user input).

Grading: You will be graded primarily on attention to detail, adherence to good programming practices, proper architecture, and well tested code.

- Your objects will be thoroughly tested for grading. If your objects can be broken (i.e. your program crashes), this will result in lowering your grade up to and including receiving a zero.
- Any run time errors or compilation errors will result in lowering your grade up to and including a zero.
- Your repo should have ONLY .cpp files, .h files, a README.md, and a .gitignore. You must use intelligent and regular commits (*commit small, commit often, commit smart*). For an application this size you should have at least 8-10 commits (more is better). Your commits have to have intelligent and proper messages. For example, "*Fixed bug in display function that was causing the program to crash due to a one-off error in the loop.*" ***DO NOT*** make commit messages like "*done*" or "*committing code*" or "*fixed bugs*." Commit messages are for telling a "story" about what you did, your thought process, and explaining step-by-step how and why you build things. You should have a commit for every small and well tested portion of code (for example, when one method is complete and tested or when one bug is fixed). Penalties for incorrect repos are as follows:
    - Failure to commit often, small, and smart will result in an automatic -10% penalty regardless of your code quality.
    - Any stray files in your repo will result in an automatic -10% penalty regardless of your code quality.
    - If you do not have proper comment headers and/or do not use the Write Submission feature (not the comment section), and/or do not submit your link correctly, it's an automatic 10% penalty regardless of your code quality.
    - Failure to use the correct branch in your repo (main) will result in an automatic -5% penalty regardless of your code quality.

Notice that if your repos and repo usage is not correct, it is impossible to achieve an A in this class. It is also impossible to even pass a job interview, let alone work as a professional developer, if you do not master git and GitHub. This is an essential and mandatory skill in this field.

Submission: When you are ready for grading, use the write submission feature in Blackboard and submit your repo's link. If you need to fix/change something *after* you submit but *before* it's graded, just fix/change it and push again. **Do not re-submit the assignment before getting a grade**. Only re-submit after you get a grade and want a *re*-grading.

More on the next page ⬇️

More on OOP class relationships…

In this assignment you are making a “is a” relationship between the base (parent) class and derived (child) class. An “is a” relationship means that the child is a kind of the parent. For example, a dog is a kind of canine. A Husky is a kind of dog. Do not confuse this relationship with the other two object oriented relationships, “uses a” and “has a.” The relationship “uses a” means one class uses another, but they do not share attributes/methods. The relationship “has a” means one class is an attribute of another class but they still do not share their attributes/methods (usually).

Examples:

- **is a** : a Husky is a kind of Dog.
- **has a** : a DogSledTeam has 1 or more Husky in it.
- **uses a** : a Driver uses a DogSledTeam.

In each example above the thing in blue is a class (i.e. it’s a noun). What you are doing in this assignment is the first case. Later in the semester we will do the other two examples. It’s up to you to come up with an example of classes in this assignment which conform to the “is a” relationship (i.e. inheritance).