#### CS 240 - Spring 2018

# Lab 3 - Creating Larger Programs

## Due Date: 11:59 p.m., February 13th, 2019

\*All programs will be tested on the machines in the Q22 lab. If your code does not run on the system in this lab, it is considered non-functioning EVEN IF IT RUNS ON YOUR PERSONAL COMPUTER. Always check that your code runs on the lab machines before submitting.\*

### Provided Files and Expected Output

\* \_Files\_

\* lab3.cpp

\* \_Expected Command > Output\_

\* Test Commands we will run

\* make

\* make run

\* make memcheck

\* make clean

\* Sample Output (with our test case):

```shell

./lab3

TEST #1: Test Planet Hill Creation with 0 planets

========================PASS========================

TEST #2: Test Add 1 Planet

========================PASS========================

TEST #3: Add Many Planets

========================PASS========================

TEST #4: Delete Planets

========================PASS========================

TEST #5: Orbit Planets Still Works

========================PASS========================

TEST #6: Print Hill Information and Free Memory

The star has 481 planets.

Planets:

Planet g10 is 2741 miles away at position 215 around the star.

Planet h11 is 2105 miles away at position 8 around the star.

...

Planet r499 is 2322 miles away at position 344 around the star.

========================PASS========================

All Tests Passed.

Don\'t forget to run with Valgrind and submit your commit hash to MyCourses!

```

### Grading Rubric

\*\*\_TOTAL: 20 points\_\*\*

\* \*\*Part A: 2 points\*\*

\* Has separate Planet and Star implementation and header files [2 points]

\* \*\*Part B: 18 points\*\*

\* Passes test #1 [3 point]

\* Passes test #2 [2 points]

\* Must allocate memory for planets with `new`

\* Passes test #3 [3 points]

\* Must allocate memory for planets with `new`

\* Passes test #4 [3 points]

\* Must deallocate memory for planets with `delete`

\* Passes test #5 [2 point]

\* Passes test #5 [2 points]

\* Passes test #6 [3 points]

\* Output prepend planet type (r,h,g) to Planet id

\* \*\*Part C:\*\*

\* If memory error, memory leak, no 'memcheck' target [-5 points]

\* Repository includes .o files or binary [-1 point]

\* Does not follow [Style guidelines](https://drive.google.com/open?id=1a5I7XhhCDRHoS8LUfILu3y2la4wW8HHh4olO30YvMVk)[-3 points]

\* Your submission will not be accepted if:

\* Does not follow requested project structure and submission format

\* Does not compile

### Guidelines

This is an individual lab assignment. You must do the vast majority of the work on your own. It is permissible to consult with classmates to ask general questions about the assignment, to help discover and fix specific bugs, and to talk about high level approaches in general terms. It is not permissible to give or receive answers or solution details from fellow students.

You may research online for additional resources; however, you may not use code that was written specifically \*to\* solve the problem you have been given, and you may not have anyone else help you write the code or solve the problem. You may use code snippets found online, providing that they are appropriately and clearly cited, within your submitted code.

\*By submitting this assignment, you agree that you have followed the above guidelines regarding collaboration and research.\*

\*\*\*

The goal of Lab 3 is to extend your previous lab by refactoring your previous code and adding a few new features. In particular: you will move most of your memory usage to the heap rather than the stack, add the ability to randomly delete an ant from the hill, enable a battle system, and further break your code out into separate files.

#### (Parts A must be completed in lab)

## Part A: Refactoring the Star Program

Refactor your C++ program to take advantage of dynamic memory (the heap). Any changes to you previous code will be highlighted with a :warning: symbol. You should make the following changes to your Star and Planet classes:

Break your Star and Planet classes into two different files. This means you will need to alter your makefile to compile Star.cpp and Planet.cpp into separate object files.

### Planet

The Planet class will now only need one constructor that takes an ID as a value. Position should be set to a random value between 0-359 (inclusive). Your Planet class should already have the following public interface:

\* public methods:

\* `Planet(int distance, int id)`

\* A value constructor that sets the Planet’s `id` to a unique value and its `distance` to a random number between 0 to 3000 (in millions of miles).

\* :warning: You should also randomly select a planet type between rocky, habitable, and gaseous. Add a new private variable, `char type` that represents the planet type with a character: 'h' = habitable, 'r'=rocky, 'g'=gaseous

\* :warning: The `pos` should now be set to a random number between 0-359.

\* `int orbit()`

\* Moves the planet +1 unit around the Star.

\* Your position should always be in the range 0-359.

\* Returns the planet's new position

\* `int getID()`

\* returns the Planet’s ID number

\* `int getDistance()`

\* returns the Planet's distance from the Star

\* `int getPosition()`

\* returns the Planet's position relative to the Star

:bulb: Next, you may want to add an additional public interface elements to the Planet class.

\* `char getType()`

\* return the type of the planet as a single character

Alter the 4 getter methods below to be inline methods. Recall from class that an inline method is defined within the class, but should be limited to very short methods such as getters and setters. This means that in your Planet.cpp should only have the value constructor and the orbit method after these changes.

\* `int getID()`

\* `int getDistance()`

\* `int getPosition()`

\* `char getType()`

\_\_Show your TA your code.\_\_

\_\_--END OF IN LAB REQUIRED WORK--\_\_

\_You may continue to work on the remainder of the lab on your own time or in lab\_

## Part B: Container Classes

### Star

\_Please read the description below describing how to refactor and add to your AntHIll class. I will only describe the changes that should be made to your class. Anything not mentioned does not change.\_

\* private:

\* `int max\_planets`

\* :warning: The max\_planets variable is no longer needed and should be removed.

\* public:

\* `Planet \*\* planets`

\* :warning: Your array of planets should become an array of planet pointers allocated on the heap. This means you will need to declare it as a double pointer, and initialize it to NULL in the constructor.

\* `Star()`

\* :warning: The constructor no longer needs an initializing value. It should set the current number of planets to 0 and the planets array pointer to NULL.

\* `~Star()`

\* You must add a destructor to deallocate any remaining planets when the Star is deleted.

\* `int addPlanet()`

\* :warning: The method to add a planet should now return the ID of the newly created Planet. The planet must be created with new (on the heap), and the resulting pointer stored in the Planet array.

\* This means that each time you add a Planet to the Star, you will have to:

\* allocate a new array of size 1 + the current number of Planets

\* copy all the pointers over to the new array

\* Create a new Planet object

\* Put the resulting Planet object pointer in the new array

\* update the current number of Planets

\* free the old array

\* Point the planet array to the new array

\* `bool removePlanet(int)`

\* Takes a Planet’s ID as a parameter, and removes the Planet from the Star, completely freeing the Planet’s memory. You must return `true` upon successful deletion and `false` on failure if the ID isn't found.

\* You must also shrink the size of the Planet array storing the Star’s Planets. This means you will have to:

\* allocate a new array of size 1 - the current number of Planets

\* copy all the pointers over to the new array except the pointer with the ID that matches the Planet to be deleted.

\* delete the Planet object with the matching ID

\* update the current number of Planets

\* free the old array

\* Point the planet array to the new array

\* `Planet \* getPlanet(int)`

\* Takes an Planet’s ID and returns a pointer to the Planet. If the Planet is not found, it returns NULL.

\* `void orbit()`

\* :warning: Although the `orbit` method does not change much, you will have to alter it to use pointers.

\* `void printStarInfo()`

\* Prints the Star information. (See sample output above)

\* :warning: Planet ids should be prepended with the planet type.

\* `int getCurrentNumPlanets()`

\* :warning: move to be inline

\* `int getMaxPlanets()`

\* :warning: no longer needed and may be removed

## Part C: Valgrind

After you have passed all of the tests, you will also need to run your code with Valgrind to ensure you do not have any memory leaks or memory errors. Valgrind is a unix tool that allows you to run many different kinds of memory tests on your code. For this class we will be concerned with two things

\* \_\_Memory Leaks\_\_

\* Valgrind catches, and if you compile with the `-g` flag, tells you where the memory leaks are (usually).

\* \_\_Memory Errors\_\_

\* If you are using uninitialized memory or you are overflowing a buffer, Valgrind will catch it.

Below is a sample of how to run Valgrind, and the output you should see:

```shell

valgrind ./lab2

… #your code runs

==18210== HEAP SUMMARY:

==18210== in use at exit: 0 bytes in 0 blocks

==18210== total heap usage: 5 allocs, 5 frees, 488 bytes allocated

==18210==

==18210== All heap blocks were freed -- no leaks are possible

==18210==

==18210== For counts of detected and suppressed errors, rerun with: -v

==18210== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

\_\_For every subsequent lab you must have 0 leaks and 0 errors or you will lose points.\_\_

## Part D : Code Organization and Submission

\* Required code organization:

\* lab3.cpp (driver code - You must include this file in your submission)

\* Star.h/.cpp

\* Planet.h/.cpp

\* makefile

\* executable should be called: lab3

\* You must have the following targets in your makefile:

\* `all` - only compiles your source code using separate compilation for each .cpp file

\* `clean` - removes all object files and binary executables

\* `run` - compiles if necessary and runs your executable

\* `memcheck` - compiles your source if necessary, then runs your executable with valgrind

Below is just a reminder of the commands you should use to submit your code. If you cannot remember the exact process, please review lab 1.

\_These commands all presume that your current working directory is within the directory tracked by `git`.\_

You will need to do the following when your submission is ready for grading.

```shell

git commit -am "final commit"

git push

```

To complete your submission, you must copy and paste the commit hash into MyCourses. Go to MyCourses, select CS240, and then assignments. Select Lab 3, and where it says text submission, paste your commit hash. You can get your latest commit hash with the following command:

```shell

git rev-parse HEAD

```