Abraham Meza

Christos Papachristos

CS 202 – 1001

March 6, 2019

Project 5 Documentation and Output Explanation

**Description:**

For this project you may use **square bracket**-indexing, **pointers**, **references**, all **operators**, as well as the **<string.h>** or **<cstring>** library functions (however the std::string type is still not allowed).

The required functionality is as follows: You are given the specifications for a 2 Classes that have an Inheritance relationship – one being the Base class (**Vehicle**) and one the Derived (**Car**). You have to translate these specifications into class implementations (header and source files) and test them against a test driver (**proj5.cpp**) which is provided.

You are also required to explain in your documentation the observed output from running the test driver, line-by-line.

Continuing through Computer Science II (202), our instructor assigned us a new kind of project that demonstrates our abilities to complete a fragment of code by making multiple linking files that support the content in the main files. Through the assignment, our instructor wants students to demonstrate their abilities to create class skeletons that incorporate initializer lists and inheritance, while completing the code provided to us. The instructor provided us with a main debugging file that would run sample tests through the student’s file to see if they follow the guidelines displayed in the project parameters. The design of the program is to create a license plate creator/tracker and a location indicator. In other words, students are assisting in creating a program that monitors and tracks specific vehicles; however, for the sake of simplicity, we are running with a hypothetical test on a made up driver. As students continue to polish their knowledge and different implementations with classes, they are also being shown that there will be times when they have to finish a program.

For my design, I simply looked at the parameters given in the instructions and scanned through the provided main file to see where to begin. Based on the information given, it seemed as if the “Vehicle” and “Car” files are the only missing pieces of the code. From there, I began to create separate header and program files for each integrated header in the main. I followed the order displayed in the instructions and began with the “Vehicle” skeleton. Once I created my “Vehicle” class and declared all of my constants, variables, members, and methods, I simply drafted my “Vehicle” program to mirror and initialize the information based on the descriptions given. When defining the constructors, I implemented initializer lists that would refer to the vin of vehicles to execute if there was not a duplicate. A difference I noticed within this project was the fact that we had to implement print statements within our constructors to print a debugging message. At first I did not know what to do since I was used to initializing numerical and member values, rather than calling statements within my constructors, so I simply tested “cout” commands in my compilations and they worked perfectly fine. After formatting my constructors, I created the member getter/setter definitions to reflect the requirements (specific parameters, etc.)), then created loops (bracket notation, etc.) to execute the class functions/methods when referenced, and operators that called member information accordingly. Once I finished my “Vehicle” program, I quickly debugged the simple mistakes and reviewed the parameters to double check the functionality.

Afterwards, I began to work on the “Car” files. Similarly to the “Vehicle” files, the concept of the class skeleton was very identical; however, I realized that I had to modify the class declaration to inherit the “Vehicle” class data. Once I inherited the class, I simply declared all of the necessary information listed within the instructions of the project and mirrored the same design as my “Vehicle” header file. After declaring all of the necessary constants, members, methods, and operators, I began to work on their definitions in the “Car” program. As a similar class skeleton, the definition structures were very identical; however I had to figure out how to inherit the “Vehicle” class information to my “Car” definitions. In order to call the necessary information, I simply referenced the class in my initializer lists. I implemented the initializer lists according to the instructions and obtained the information instructed (LLA, vin, car, etc.). From there, I simply referenced to the parameters of the project to define all of my constructors accordingly, labeled my getter/setter rules, and mimicked the structure of the methods and the insertion operator in the “Vehicle” class. Once I compiled the file and fixed all of my minor errors, I linked the files together.

Once I compiled and tested the files, there was no complication with the output. It seemed as if the main file provided was able to run through all of the necessary tests and printed the debugging messages I implemented. Unfortunately, our instructor did not provide a sample output in which we can reference to check our implementations and accuracy; thus students had to go through each line of output and trace their missing code in order to check that its functionality was correct. The instructor expects students to paste their output into their documentation to prove they understand the concept of classes, initializer lists, methods, operators, and inheritance. At the end of the day, the functionality of the project depends on the students’ ability to experiment with heir knowledge until it seems correct.

All in all, the project was very efficient in teaching students about inheritance and further implementations and functions within the concept of classes. Personally, I enjoyed the simple task of completing the instructors code; however, I have always had a sample output to reference my work, so I did not enjoy the constant doubt in my mind that wondered if my code was correct.

Project 5 Output and Explanation

|  |  |
| --- | --- |
| //////////////////////////// | \*\* cout labeling design |
| ///// Base Tests ///// | \*\* cout labeling design (Base Tests) |
| //////////////////////////// | \*\* cout labeling design |
|  |  |
| Testing Base Default ctor | \*\* cout confirmation that system is testing default constructor |
| Vehicle #0: Default-ctor | \*\* system is printing out default vehicle number (initialized to 0) |
|  |  |
| Testing Base insertion operator | \*\* cout confirmation that system is testing insertion operator |
| Vehicle #0 @ [0, 0, 0] | \*\* system is printing default LLA for default vehicle (initialized to 0) |
|  |  |
| Base idgen: 1 | \*\* system incremented vin #0 by one and printed vin to the screen to avoid re-usage |
|  |  |
| Testing Base Parametrized ctor | \*\* cout confirmation that system is testing parameterized constructor |
| Vehicle #99: Parameterized-ctor | \*\* system assigned and printed vehicle value to 99 based on programmer indication within code |
| Vehicle #99 @ [39.54, 119.82, 4500] | \*\* system assigned LLA to programmer specifications within code and printed out vehicle data and its current location |
|  |  |
| Base idgen: 100 | \*\* system incremented vin #99 by one and printed vin to the screen to avoid re-usage |
|  |  |
| Testing Base Copy ctor | \*\* cout confirmation that system is testing copy constructor |
| Vehicle #100: Copy-ctor | \*\* system copying vehicle #99 information to vehicle #100 |
| Vehicle #100 @ [39.54, 119.82, 4500] | \*\* system printed out vehicle #100 information and its current location |
|  |  |
| Base idgen: 101 | \*\* system incremented vin #100 by one and printed vin to the screen to avoid re-usage |
|  |  |
| Testing Base Assignment operator | \*\* cout confirmation that system is testing operator overload (assignment) |
| Vehicle #0: Assignment | \*\* Vehicle #0 (first base vehicle) is assignment destination to specific vehicle information (programmer indicated Vehicle #99 [LLA] as source) |
| Vehicle #0 @ [39.54, 119.82, 4500] | \*\* system printed out assigned information from Vehicle #99 (LLA) |
|  |  |
| Base idgen: 101 | \*\* system kept vin count the same since information was not modified |
|  |  |
| Testing Base Move Function | \*\* cout confirmation that system is testing “Move” class function |
| Vehicle #0: CANNOT MOVE - I DON'T KNOW HOW | \*\* debug message from “Move” function applied to Vehicle #0 (first base vehicle) |
|  |  |
| //////////////////////////// | \*\* cout labeling design |
| ///// Derived Tests ///// | \*\* cout labeling design (Derived Tests) |
| //////////////////////////// | \*\* cout labeling design |
|  |  |
| Testing Derived Default ctor | \*\* cout confirmation that system is testing derived constructor through vehicle inheritance in “Car” class skeleton |
| Vehicle #101: Default-ctor | \*\* system is printing out derived vehicle number from last vin count (101) |
| Car #101: Default-ctor | \*\* system is printing out derived car number from last vin count (101) |
|  |  |
| Testing Derived insertion operator | \*\* cout confirmation that system is testing insertion operator through vehicle inheritance in “Car” class skeleton |
| Vehicle #101 Plates: , Throttle: 0 @ [0, 0, 0] | \*\* system printing out current throttle and LLA state of Vehicle #101 |
|  |  |
| Derived idgen: 102 | \*\* system incremented vin #101 by one and printed vin to the screen to avoid re-usage |
|  |  |
| Testing Derived Parametrized ctor | \*\* cout confirmation that system is testing parameterized constructor through vehicle inheritance in “Car” class skeleton |
| Vehicle #999: Parameterized-ctor | \*\* system assigned and printed vehicle value to 999 based on programmer indication within code |
| Car #999: Parameterized-ctor | \*\* system assigned and printed car value to 999 based on programmer indication within code |
| Vehicle #999 Plates: Gandalf, Throttle: 0 @ [39.54, 119.82, 4500] | \*\* system assigned LLA and License string to programmer specifications within code and printed out vehicle/car data and its current location |
|  |  |
| Derived idgen: 1000 | \*\* system incremented vin #999 by one and printed vin to the screen to avoid re-usage |
|  |  |
| Testing Derived Copy ctor | \*\* cout confirmation that system is testing copy constructor through vehicle inheritance in “Car” class skeleton |
| Vehicle #1000: Copy-ctor | \*\* system copying vehicle #999 information to vehicle #1000 |
| Car #1000: Copy-ctor | \*\* system copying car #999 information to car #1000 |
| Vehicle #1000 Plates: Gandalf, Throttle: 0 @ [39.54, 119.82, 4500] | \*\* system printed out vehicle/car #1000 information and its current location |
|  |  |
| Derived idgen: 1001 | \*\* system incremented vin #1000 by one and printed vin to the screen to avoid re-usage |
|  |  |
| Testing Derived Assignment operator | \*\* cout confirmation that system is testing overload operator (assignment) through vehicle inheritance in “Car” class skeleton |
| Car #101: Assignment | \*\* Vehicle #101 (first derived car) is assignment destination to specific vehicle/car information (programmer indicated Vehicle #999 [LLA] [Plates] [Throttle] as source) |
| Vehicle #101 Plates: Gandalf, Throttle: 0 @ [39.54, 119.82, 4500] | \*\* system printed out assigned information from Vehicle #101 (Plates) (Throttle) (LLA) |
|  |  |
| Derived idgen: 1001 | \*\* system kept vin count the same since information was not modified |
|  |  |
| Testing Derived Move Function | \*\* cout confirmation that system is testing “Move” class function through vehicle inheritance in “Car” class skeleton |
| Car #101: DRIVE to destination, with throttle @ 75 | \*\* ouput message from “Move” function applied to Vehicle #101 (first derived car); assigned drive to value of “75”. |
|  |  |
| //////////////////////////// | \*\* cout labeling design |
| ///// Tests Done ///// | \*\* cout labeling design (Tests Done) |
| //////////////////////////// | \*\* cout labeling design |
| Car #1000: Dtor | \*\* system cout confirmation that car #1000 destructor was a success and that all recorded memory and data has been cleared from the program |
| Vehicle #1000: Dtor | \*\* system cout confirmation that vehicle #1000 destructor was a success and that all recorded memory and data has been cleared from the program |
| Car #999: Dtor | \*\* system cout confirmation that car #999 destructor was a success and that all recorded memory and data has been cleared from the program |
| Vehicle #999: Dtor | \*\* system cout confirmation that vehicle #999 destructor was a success and that all recorded memory and data has been cleared from the program |
| Car #101: Dtor | \*\* system cout confirmation that car #101 destructor was a success and that all recorded memory and data has been cleared from the program |
| Vehicle #101: Dtor | \*\* system cout confirmation that vehicle #101 destructor was a success and that all recorded memory and data has been cleared from the program |
| Vehicle #100: Dtor | \*\* system cout confirmation that vehicle #100 destructor was a success and that all recorded memory and data has been cleared from the program |
| Vehicle #99: Dtor | \*\* system cout confirmation that vehicle #99 destructor was a success and that all recorded memory and data has been cleared from the program |
| Vehicle #0: Dtor | \*\* system cout confirmation that vehicle #0 destructor was a success and that all recorded memory and data has been cleared from the program |