Abraham Meza

Christos Papachristos

CS 202 – 1001

March 13, 2019

Project 6 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).

This project expands on a more simplified version of Project 5. 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 class (**Car**). You have to translate these specifications into class implementations (header and source files) and test them against a test driver (**proj6.cpp**) which is provided. You are also required to explain in your documentation the observed output from running the test driver.

Continuing through Computer Science II (202), our instructor assigned us a similar project that demonstrates our abilities to complete a fragment of code; however, the project was designed to expand upon the idea of inheritance and operator overload to create a method polymorphism that created child class implementations. Through the assignment, our instructor wants students to demonstrate their abilities to create class skeletons that incorporate initializer lists, inheritances, and operator overload functions within the private member categories, 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 monitor the motion of a vehicle by tracking the LLA at a consistent throttle. Similar to the previous project, the program would create the same functions and methods to toggle the vehicle location; however, virtual methods would be used to create a more in-depth environment through the provided test 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 referred to the instructions and traced the main code to see the missing portions in which the professor wanted us to program. Based on the information provided, students had to create header and programs that were illustrated as “Vehicle” and “Car” as they link their functionalities to support the main file. Similar to the previous project, students were to create class skeletons that would pass inheritance to a child class in order to create an in depth analysis within the program. By overloading the base class, students will be able to fluently modify and override base functions and pointers. I began to declare, initialize, and implement my members, functions, operators, and methods in accordance to the instructions. The structure of the skeletons was very identical to project 5 as they portrayed the same data. Even so, there were added class functions that expanded its intended functionality. To be more specific, the professor implemented a sterilize method within the private illustrations. At first I was very confused to why the class method was in a restricted interface, but as I continued to program, I realized that the idea allowed the implemented function to be inherited and overridden; thus, allowing the child class to implement its own structure to its functionality. As I finished up defining all of my members and implementations, I modified my insertion operator and serialize to the conditions listed. Through the process, I created simple array printing and organizing algorithms that would allow me to access the LLA data within the “Vehicle” class. Once I finished my “Vehicle” program, I quickly debugged the simple mistakes and reviewed the parameters to double check the functionality.

Afterwards, I move onto the “Car” skeleton and implementation. The content within the files were very identical to those in “Vehicle”; however, I needed to inherit the data from the base class and modify the members to fit the ones reflected in the instructions. Once I mapped and wrote the qualified members, functions, methods, and operator overloads, I simply referenced to the project parameters to shape the skeleton. As it continued to mimic the implementation of the base class, the program’s definitions were intuitive and quick. However, the professor explicitly instructed the students to not write another insertion operator for the child class. In doing so, I became very confused as to why we wrote different serialize functions when only one would be used. Once I finished and compiled the code, I realized that the “Vehicle” serialize method was never used and continued to use the one displayed in car. After some careful review and confirmation from the professor, I realized that the polymorphism functionality allowed the class operator/method to be overridden. The real purpose of the project was to show students how to override private methods through the utilization of polymorphism. Once I understood the content, I revised and created simple fixes to the files for better efficiency.

After a few tests, I was satisfied with my results. Unfortunately, students were not provided a sample output to compare their programs, thus had to rely on their ability to follow instructions without a safety net. 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 their knowledge until it seems correct.

All in all, the project was very efficient in teaching students about polymorphism, while allowing them to further master inheritance and implementations within classes. Personally, I enjoyed the simple task of completing the instructors code. Even so, the project proved challenging in a different way as I have always had a sample output to reference my work. In the situation, I had to rely on my programming knowledge and experience to overcome the task provided.

Project 6 Output and Explanation

|  |  |
| --- | --- |
| //////////////////////////////// | \*\* cout labeling design |
| ///// Constructor Tests ///// | \*\* cout labeling design (Constructor Tests) |
| //////////////////////////////// | \*\* cout labeling design |
|  |  |
| Testing Derived Default ctor | \*\* cout confirmation that system is testing default constructor (derived) |
| Vehicle: Default-ctor | \*\* cout confirmation that system tested default constructor within “Vehicle” class; left everything uninitialized |
| Car: Default-ctor | \*\* cout confirmation that system tested default constructor within “Car” class; set throttle to default |
|  |  |
| Testing Derived Parametrized ctor | \*\* cout confirmation that system is testing parameterized constructor (derived) |
| Vehicle: Parameterized-ctor | \*\* cout confirmation that system tested parametrized constructor within “Vehicle” class; set LLA based on passed argument |
| Car: Parameterized-ctor | \*\* cout confirmation that system tested parameterized constructor within “Car” class; set throttle to default |
|  |  |
| Testing Derived Copy ctor | \*\* cout confirmation that system is testing copy constructor (derived) |
| Vehicle: Default-ctor | \*\* cout confirmation that system tested copy constructor within “Vehicle” class; copies vehicle.LLA to destination |
| Car: Copy-ctor | \*\* cout confirmation that system tested copy constructor within “Car” class; copies referenced throttle value to destination |
|  |  |
| Testing Derived Assignment operator | \*\* cout confirmation that system is testing assignment constructor (derived) |
| Car: Assignment | \*\* cout confirmation that system tested assignment constructor within “Car” class; assigns “Car” data if destination data is not the same |
|  |  |
| ///////////////////////////////// | \*\* cout labeling design |
| ///// Polymorphism Tests ///// | \*\* cout labeling design (Polymorphism Tests) |
| ///////////////////////////////// | \*\* cout labeling design |
|  |  |
| Testing VIRTUAL Move Function for DERIVED Class Objects | \*\* cout confirmation that system is testing virtual move function in inheritance to child class (Car) |
| Car: DRIVE to destination, with throttle @ 75 | \*\* cout confirmation that “Car” class move function works; prints confirmation, assigns an argument to drive method, and updates LLA through inheritance |
|  |  |
| Testing Insertion operator<< Overload for BASE Class Objects | \*\* cout confirmation that system is testing insertion overload operator overridden by child class (“Car”) |
| Car: Throttle: 0 @ [39.54, 119.82, 4500] | \*\* insertion operator output based on serialize function that organizes programmer data given within file in a specific order (overridden by “Car” child class, thus prints “Car” serialize format) |
|  |  |
| /////////////////////////////////////////////////// | \*\* cout labeling design |
| ///// Polymorphic Base Class Pointer Tests ///// | \*\* cout labeling design (Polymorphic Pointer Tests) |
| /////////////////////////////////////////////////// | \*\* cout labeling design |
|  |  |
| Testing VIRTUAL Move Function on Base Class Pointers | \*\* cout confirmation that system is testing virtual move function in base class pointers (inheritance to “Car”) |
| Car: DRIVE to destination, with throttle @ 75 | \*\* cout confirmation that “Car” class move function works; prints confirmation, assigns an argument to drive method, and updates LLA through inheritance (loop 1st data) |
| Car: DRIVE to destination, with throttle @ 75 | \*\* cout confirmation that “Car” class move function works; prints confirmation, assigns an argument to drive method, and updates LLA through inheritance (loop 2nd data) |
| Car: DRIVE to destination, with throttle @ 75 | \*\* cout confirmation that “Car” class move function works; prints confirmation, assigns an argument to drive method, and updates LLA through inheritance (loop 3rd data) |
|  |  |
| Testing Insertion operator<< Overload for Base Class Pointers | \*\* cout confirmation that system is testing insertion overload operator on base class pointers (override by “Car”) |
| Car: Throttle: 75 @ [40.71, 74, 10] | \*\* insertion operator output based on serialize function that organizes programmer data given within file in a specific order [1st data] (overridden by “Car” child class, thus prints “Car” serialize format) |
| Car: Throttle: 75 @ [39.54, 119.82, 4500] | \*\* insertion operator output based on serialize function that organizes programmer data given within file in a specific order [2nd data] (overridden by “Car” child class, thus prints “Car” serialize format) |
| Car: Throttle: 75 @ [34.05, 118.24, 71.01] | \*\* insertion operator output based on serialize function that organizes programmer data given within file in a specific order [3rd data] (overridden by “Car” child class, thus prints “Car” serialize format) |
|  |  |
| //////////////////////////// | \*\* cout labeling design |
| ///// Tests Done ///// | \*\* cout labeling design (Tests Done) |
| //////////////////////////// | \*\* cout labeling design |
| Car: Dtor | \*\* system cout confirmation that car destructor was a success and that all recorded memory and data has been cleared from the program [1st data] |
| Vehicle: Dtor | \*\* system cout confirmation that vehicle destructor was a success and that all recorded memory and data has been cleared from the program [1st data] |
| Car: Dtor | \*\* system cout confirmation that car destructor was a success and that all recorded memory and data has been cleared from the program [2nd data] |
| Vehicle: Dtor | \*\* system cout confirmation that vehicle destructor was a success and that all recorded memory and data has been cleared from the program [2nd data] |
| Car: Dtor | \*\* system cout confirmation that car destructor was a success and that all recorded memory and data has been cleared from the program [3rd data] |
| Vehicle: Dtor | \*\* system cout confirmation that vehicle destructor was a success and that all recorded memory and data has been cleared from the program [3rd data] |