Review 1

CS202 Programming Systems, Summer 2020

Armant Touche

Due: July, 7, 2020

Design Analysis

In this program, I attempted to implement a time management system with Object Oriented Principles. I used a single inheritance hierarchy with my base class being Activity. From Activity, I derived three classes: (i) Hobby (ii) Work (iii) School. In my base class, I implemented dynamic binding for functions that were common through out the hierarchy which allowed function overloading to take place across the hierarchy, starting from the object data type invoking the polymorphic function. With dynamic binding, I was able to lower the amount of wrapper functions within the derived classes by using a up casting and calling a helper function. The helper function intakes a base object by reference and using direct access to the virtual function of the object data type that was passed by reference. This technique allowed less data type checking on the object. My function across the board weren't too unique but did take into consideration the different data members within each class. The difficult part was taking the created activity and copying it to the array of doubly linked list of days. For my derived classes, I have function like copy, display, read, and check data type. My list of function I chose to dynamically bind to the common hub class, activity, are: (1) read (2) display (3) check data type. Even though these are common, each derived class different data member which is why I chose to implemented dynamic binding. The array indices corresponds with the day in which is being planned for which satisfies the requirement for keeping track of the days. The single inheritance hierarchy allowed my client program to easily create and call function from the derived class and base class which was nice and paired with dynamic binding, I just had to upcast and use some helper functions that expected a base object to passed by reference. Up casting and passing the object to helper function, which the helper function expects a base object to be passed by reference. No copy was made which allowed me to use the dot operator and through polymorphic function overloading, I was able to go straight to the derived object type. Every class has a default constructor (i.e., a constructor with no arguments). This constructor does nothing unless you implement one. As soon as you implement a constructor for a class, the automatically supplied default constructor is not provided. Initialization lists apply only to constructors of derived objects.

Debugger

Using GDB was perhaps the biggest time saver because I kept losing track in the beginning which constructor was being invoked first. Stepping through GDB really solidified in my mind how single-hesitance really worked and was especially important when doing dynamic binding because I could be able to trace effectively through looking code when it came time to insert into the planner array and doubly linked list. I used watch, breakpoints, and backtrace a lot to aide in debugging program one. Working the base class and it's virtual methods in GDB really helped solve issues like reading incorrectly or displaying data members from the base class and not any of the derived one. I realize a little to late into programming that dynamic binding was only useful for function and if you needed to access the derived classes data, that wasn't going to happen. I had to pull an audible a little late in the game but I think I am pretty set up for program number two. I just need to learn return type identification and abstract base class concepts which will be covered next week. I will there was

a straightforward way to access data members in a derived class via dynamic binding but I need to practice anyway. No worries.