**Effectiveness & Approach**

The design and intent of Program 4 was to create a Course Media Playlist manager. The program would enable the user to be able to add different types of media to multiple playlists that would represent different lecture material for a college course. The user would be able to input the different qualities of the media as well as add comments, a quiz, and other various information depending on the type of media.

This goal was achieved utilizing single inheritance with an Abstract Base Class named Media and two Derived Classes, Long video, and Short video and the use of dynamic binding. The Base Class would house the majority of the media’s information while the derived would focus on what made that type of media unique among its class. I believe this hierarchy made the execution of this program was better as it removed the need for the Derived classes to focus on anything but their specific role.

The program was broken down into three menus. The first menu was where the user could create, remove, display all the types of media and operated as the main menu for the program and operated primarily on the fields that were in the base class, including adding comments to the media. The other two menus were utilized for use with the long and short video materials specifically and enabled the user to further customize their media by adding a URL, work cited page, and a quiz. The quiz feature was implement utilizing a “has a” relationship. This enabled either of the media types to be able to have a quiz that the user can create or not is they choose to.

**Object Oriented Programming**

In terms of object oriented programming, I believe that this program implemented it much more efficiently than previous programs. The use of an abstract base class enabled not only different types of media to be stored within a data structure, but also enabled all the classes to engage in their role and not have to repeat work of another class. I chose to have my abstract base class contain most of the data used for a media object and using dynamic binding perpetuated the programs ability to take the data as needed and create the objects at runtime. This design lent itself well to the idea that another media type could be added to the program with little need to alter a bulk of program itself.

Dynamic binding also allowed for classes to change unique data members as needed. There were several pure virtual functions that were implemented in both the short and long video classes. While these took care of the unique fields in their respective classes, the base class worked separately on its own fields. Using RTTI further enabled me to separate the derived classes, I could determine what type of object my was at run time and then call the appropriate method as needed. This removed the need to create virtual functions that only serviced one type of derived class and not the other and reduced the overall amount of code.

I chose to implement the quiz feature as its own class and house an object of that class inside both of the derived classes. This allowed the quiz class to manage itself and engage in its role without the need of the derived classes to do anything but call a single method. Using this method made the most sense, as it allowed some media objects to not create quizzes if the user did not wish too.

I also utilized function overloading between classes heavily in this project. I decided to overload all functions that worked with the same fields of a class. I would overload the function and provide arguments that the specific function would need. The largest application of this was the same named functions to do both adding data to an object as well as displaying the object. The use of dynamic binding furthered this by needing to use RTTI to determine the type of data as to call the correct non-virtual function, as well as call the correct overloaded version of that function.

**Data Structures**

The data structure used for this assignment was an array of linear linked lists. Each index of the array represented a different playlist for the user to choose from. Each node inside of the linear linked list represented a single media object. Dynamic binding allowed for different types of media to be stored in each of the nodes. I believe this data structure was the most appropriate for the program as it allowed the user to interact with the created media in a way that simulated a catalog. The use of an array of the linear linked lists also allowed the user to categorize the media they wished to add in whatever way they thought was best.

**External Data File**

This program utilized two external data files. The first allowed the user to import data from a file that would populate one of the linear linked lists in the array with all of the information stored in the file. This feature could act as a showcase for that the user might wish to do with their own custom information. The other external file was appended to anytime a media object was created, including the imported data from the other file. This allowed all of the work the user did to be saved in an external file and organized for future reference.

**Conclusion**

I believe that my approach and implementation of the programming assignment was successful. My program was able to implement an object oriented design that had data flow through the hierarchy fluidly. The program itself provided many options to user to create, alter, customize, and remove their media to their own preference. The only thing I would have changed would have been the ability for the user to manage the number of playlists they had access to and not limit them in this capacity. Despite this I believe my program fulfilled its intentions well and preformed as designed.