Program #2
CS 202 Programming Systems

***** Make sure to read the Background Information first!**
It applies to all programming assignments this term***

****THIS IS NO DESIGN WRITEUP and NO UML DIAGRAM for Program #2 ****

Program #2 – Purpose

In our first programming assignment, we experimented with the notion of OOP and breaking the problem down into multiple classes building a relatively large system. The purpose of that assignment was to get to know how we can create classes that have different relationships. Not every class derived from the base was expected to have the same functionality.

Now, in program #2 our focus shifts to getting experience developing a hierarchy that can be used with dynamic binding. This means that the classes derived from a base class need to have a self-similar set of functions with the same function name, same return type, and same argument list. This allows the application program to use one line of code to call one of many functions. To get the full benefit of dynamic binding, a possible design has been provided for you below.

Program #2 – General Information

Do you play games? For myself, discord is really helpful when communicating with others in the gaming community. Discord has specific commands you can use to message someone by using the @only_one or @highlander_cpp and get to them directly. You can also use commands to minimize spam and keep the clutter down so it will work more efficiently.

As we have seen with class discussions, having an online way to easily communicate is pretty important especially in this land of remote learning. Do you use the chat channel in Zoom? Or do you use Slack, Pronto, or Zoom IM? All have advantages with similar features but they don't all act the same. Some of these systems have threaded communication. Some systems allow for only individual conversations and others allow for messages to be broadcast to everyone in the channel. But in the end, all provide a way to post messages, read messages, and a few have a way to delete.

Program #2 – Building a Hierarchy using Dynamic Binding

For your second program, you will be creating a C++ OOP solution to support at least three different ways to communicate in these unusual times. Create a base class for general communication that is common among the three classes. This must be an abstract base class.

Then, create three different derived classes for specific types of communication. One form needs to be discord. The other two are of your choice; although one form needs to support threaded discussions (where you can reply to a message). If you have never used discord, I would recommend performing a web search to become acquainted with it. It is important that each type of communication have similarities– so don't pick ideas that are vastly different. Most will need a sender, receiver, and message along with a user name.

**You need 5 member functions in each of the three forms of communication in addition to your constructors and destructors **; with dynamic binding, these need to be self similar in nature. You will want to send a message, read a message, and remove a message as functions at a minimum. With discord, you will find there will be a few differences – such as using the discord commands; implement at least one discord function that is different than the other forms of communication so that you can experience RTTI in practice; this is a requirement.

The purpose of this assignment is to use and experience dynamic binding. The requirement for this application is to have at least **three DIFFERENT types of classes**, derived from **a common abstract base class**! To use dynamic binding, there needs to be a self-similar interface among the derived class methods but the implementation of the methods would need to be different in some way for you to make the most of this experience.

Program #2 – Data Structure Requirements

The real beauty of dynamic binding is that we can have a data structure be a collection of different data types! There needs to be ONE data structure of base class pointers. This data structure will keep track of all of a user's communication accounts, using upcasting. For example, I have six email addresses that I use, two slack channels, and one discord account. I would want my data structure to keep track of all of these user names and messages.

The data structure will be **a Linear linked list of base class pointers** pointing to the communication objects within the hierarchies for each account. Then there needs to be a data structure of your choice to keep track of the messages in that account. **EXTRA CREDIT** will be provided if you select an **Array of linear linked lists**. Implementation of the data structure(s) requires full support of insert, removal, display, retrieval, and remove-all.

ALL repetitive data structure algorithms should be implemented recursively to prepare for proficiency demos!

Program #2 – Important C++ Syntax

Remember to support the following constructs as necessary:

1. Every class should have a default constructor
2. Every class that manages dynamic memory must have a destructor
3. Every class that manages dynamic memory must have a copy constructor
4. If a derived class has a copy constructor, then it MUST have an initialization list to cause the base class copy constructor to be invoked
5. Make sure to specify the abstract base class' destructor as virtual (but don't make it a pure virtual function)
6. It is expected that you will experience RTTI with dynamic_cast in this assignment

IMPORTANT: ***OOP and dynamic binding are THE PRIMARY GOALS of this assignment!***