Programming Assignment #2
CS 302 Programming Methodologies

*****This program is about Generic Programming with Operator Overloading and Class Templates *****

Program #2 - Goals

With abstractions, the key idea is to break down the problem into small pieces and assign those responsibilities to individual classes. For each assignment, your job will be to build relationships between abstractions. You will want to focus on how to design classes that are well structured, efficient, that work together, and where each class has a specific **"job"** to perform. In Programming Assignment #1, the core hierarchy was given to you. This time, you will be creating your own hierarchical design given some basic requirements. As part of your solution, you will be applying the concepts from Labs 4 and 5 with **operator overloading and class templates** as the new syntax for your solution!

Operators are a great way to support similar functionality across a variety of classes. By pairing this with class templates, we can write the client software once and use it for any data type desired. For us, that means we will write our data structure code once and have it apply to data types within our solution. This does assume that the proper operators are overloaded so our underlying types used by the templates will perform properly.

With this assignment, you will be expected to design your own core hierarchy that is based on the concept of supporting operators. We are starting with the first week creating a specification and an overall design. This is followed by a week designated to implement the core hierarchy, which should begin after completing Lab 4. Then in the second phase of the assignment, we will implement a class template for a doubly linked list. The data type specified in the node will now be based on generic programming and class templates (Lab 5). For this to all work seamlessly, it will be important that your core hierarchy supports a full set of operators. **Exception handling** is also vital for handling for erroneous situations when working with operators – so the concepts we have been covering will all fit together! With operators, it is no longer possible to return success/failure from a method!

Please follow the assignment's guidelines carefully.

Program #2 – Background

My family often plays games – but have you ever noticed that how you play a game varies depending on the level of the player? For example, when we play with our 4 year old – the approach and strategy is completely different then when we are playing with experienced players.

We have decided for Program #2 to build our own game that supports three different levels of player (such as: Novice, Intermediate, and Advanced). The game will be a simulation of a board game, where the doubly linked list will represent the game board with each node being a player. Players will navigate through the game board with different events taking place. The goal will be to go from the start of the game to the end and collect as many points; along the way, a player may encounter and overcome obstacles. The winner is the one that reaches the end with the most points.

Games are played with the same level of player. So when we play with the 4 year old, we all are playing at the novice level. Decisions have to be made at the start which level we want to play at as it will impact how the tame is to be played.

Here are some basic requirements for this game:

1. There must be three levels of players represented by three classes and derived from a common base class. You may decide what three levels you would like to support. Please add your own twist on similarities and differences between the three level of players; here are some ideas:
    a. Novice (for example, only travel forwards along the game board)
    b. Intermediate (possibly allowing travel in both directions)
    c. Advanced (for example could add extra obstacles or challenges as players move through the board)
    d. All three types of players must have at least 2 attributes in common
    e. The base class and one derived must have a char *, a copy constructor and an assignment operator. All other string data must be represented by the string class.
2. Support the concept of a list of Events or Actions using the STL
    a. An Event or Action should be a class. Think of this as something that will happen when a player takes their turn. It could cause the player to move forward, backward, start over, switch with an opponent operation takes place, lose a turn, etc.
    b. Use the STL Array, List, or Forward List for the list of events to select from. A random number generator should also be part of your salutation when applying events or actions.

Specification and Design Week Deliverables

We will start this first week with a specification (due on Wednesday) with a design and UML diagram at the end of the week (due on Friday). During this first week you will need to make some major decisions about what the game will be like and clarify the key details about the software being developed.

Specification:
The specification is a written document that represents the details on what the game will be doing. It should be at least 600 words long. Discuss how you will handle each of these situations:

1. What is the name of your game?
2. What type of obstacles or events or actions could take place while a player is navigating the game board?
3. How many players will you support?
4. How do you decide who starts first?
5. What three levels of players do you want to support?
6. How will a player collect points?
7. What types of events or actions will you support and how will they affect the movement of a player through the game?
8. How will the events relate to the level of players in the game?

Design:
By the end of the first week, turn in a UML diagram and design writeup (at least 600 words) which discusses the data and functionality supported in each class:

1. For the three levels of players:
    a. What the data will be in the base player class?
    b. What data will be in the derived player classes?
    c. Specifically, what will be the difference between the three different types of players in how they navigate the board, points get collected, and ultimately win?
2. And, for those players, what types of operations will you support?
    a. What happens when it is a player's turn?
    b. ***IMPORTANT*** What operators will make the most sense when working with a player? (hint – we can use the equality and relational operators to compare points)
3. Plan how to support the list of Events?
    a. Which STL container will you use?
    b. How will the random number generator be applied?

First Progress Submission: The core hierarchy

After you have made the major decisions about the game functionality, it will be time to begin implementing the core hierarchy. The core of your assignment is to create three different levels of players. Keep in mind that when playing the game, only a single level of player will be on the board at any given time – which means your game board will contain objects of a particular level. When you are picking the level of players you want to support – there must be similarities but there also must also be some differences. Have fun and be creative!

The three levels of players need to be derived from a common base class. Push up the common elements of the three different levels of players. Pick 2 or 3 items of interest that you would like to keep track of for each level. For example, the base class might keep track of the player's points and provide the functionality to manage those points. *Start with this hierarchy – any data structure you want for a collection of data within these classes should come from the STL!*

*As with Program #1, one base class must have a char * data member, and one derived class should have a char * data member. All other strings should be supported by the STL.*

Applying Operator Overloading to your FIRST progress submission

After Lecture and Lab during Week 4, it will be time to begin adding operators for your three level of players. Operator overloading is necessary so that your Class Template (for progress submission #2) can be truly written independent of the underlying data. **Support the assignment, relational and equality operators, and I/O (<< and >>).** Support one of the binary arithmetic operators (such as + or -) and the corresponding compound assignment operator (+= or -=) that goes along with the operator you select. You might decide to use the ++ to increment the number of points. If you do that, please implement both prefix and postfix versions.

Here is a summary of the operators to overload:

1. Assignment
2. Relational (< <= > >=)
3. Equality (== !=)
4. I/O (<< >>)
5. One Binary arithmetic (e.g., +) and compound assignment (e.g., +=)
6. Optional: ++, --

As you decide how to apply the operators, make sure to stay within the rules of how the operators are expected to behave. You may find that some operators

don't apply at all (and therefore shouldn't be implemented in each class). Don't forget your copy constructor and assignment operators for any class that manages dynamic memory! *The Powerpoint slides do have examples of how to overload each of these operators!*

For the first progress submission, complete at least 4 of the assigned operators.

Second Progress Submission: Creating the Application

You will be creating a Doubly Linked List using class templates for this assignment. This data structure should be used to keep track of the game board (of a particular level of player specified). You should not be mixing different data types in a given data structure. Instead, you will implement the DLL once and with templates and have three different instantiations of the class created based on the level of player that game will be supporting.

You will be implementing a Doubly Linked List in this programming assignment. You will be implementing it ONLY ONCE - it must be implemented using templates – so both the Node class and the DLL class need to be class templates; we will show an example of how to do this in lecture during Week #4 and you will use what you learn in Lab5 to organize this part of the program. This means we will implement the code only once to support each game board. **As usual, all repetitive methods for the DLL must be implemented recursively.**

For the second progress submission, progress should be shown on data structures implementation and towards being able to play the game. But a completed implementation is not expected until the finished due date.

Completed Program Expectations:

For the completed program, we expect the game to be able to be played by two players per your specification and design.

Syntax REQUIRED for this assignment:

- **Single Inheritance Hierarchy**
  - A core hierarchy with at least there classes derived from a base class
- **Support Operator Overloading for the Core Hierarchy**
  - The core hierarchy needs the specified list of operators overloaded
  - This is primarily expected in your core hierarchy and not elsewhere
- **Exception Handling**

- When something unexpected is experienced within an operator, we can't return success or failure. An exception must be thrown and handled by the client program.
- This is primarily expected within your core hierarchy

- **Use the string class**.
  - Two char *'s should be used as data members within the hierarchy (base-and-derived relationship). All others should be strings.
- **Use at least one data structure from the STL**
  - **Array, List, Forward List**
- **The DLL and Node class should be implemented using class templates and raw pointers**
  - Every class that manages dynamic memory must have a copy constructor and an assignment operator

- ***DO NOT USE THE VIRTUAL keyword in this assignment ***

Questions to ask…about operator overloading

When using operator overloading, remember to ask yourself the following questions:

a) What should be the residual value (and what data type is this)?
- Avoid a void return type, with the exception of the assignment operator
- Throw exceptions when an unexpected situation takes place, such as bad data being passed in as an argument
- The only place structs may be used is with exception handling to create self-documenting exceptions to throw

b) Should residual value be a modifiable lvalue or an rvalue?
- Lvalues are returned by reference,
- Most rvalues are returned by value.

c) What are the data types of the operator's operands?
- Remember to never pass a class type by value
- Pass class types as constant references whenever possible
- Passing pointers to operators is not recommended (keep in mind that operators are already supported for pointer arithmetic within the language – we can't change the language!)

d) Is the first operand always an object of class?
- If so, then it should be a member function.
- If not, then it should be a friend function.

e) Can the operator be used with constant operands?

- If the first operand can be a constant, and IF it is a member function, then it should be a constant member function.