CPSC 122 Computer Science II

Gonzaga University

Daniel Olivares

PA6 – Player Battle Simulator

Inheritance, Linked Lists ADTs, and Recursion (100 points)

Individual, non-collaborative assignment

Learner Objectives

At the conclusion of this programming assignment, participants should be able to:

- Implement derived classes inheriting from an abstract base class
- Implement and use virtual functions, redefined functions, and overridden functions in base and derived classes
- Implement and use template functions
- Implement and use a linked list and class member functions
- Use and manipulate pointers and pointer operators for a linked list class.
- Use Dynamic Memory Management in C++ to create a linked list.
- Use public and private class member variables and functions for a linked list class.
- Implement and use recursive functions

Prerequisites

Before starting this programming assignment, participants should be familiar with C++ topics covered in CPSC 121 including (but not limited to):

- Conditional statements
- Implement loops to repeat a sequence of C++ statements
- Define/call functions
- Implement and use class objects
- Implement and use dynamic memory and linked list member functions
- Demonstrate the software development method and top-down design

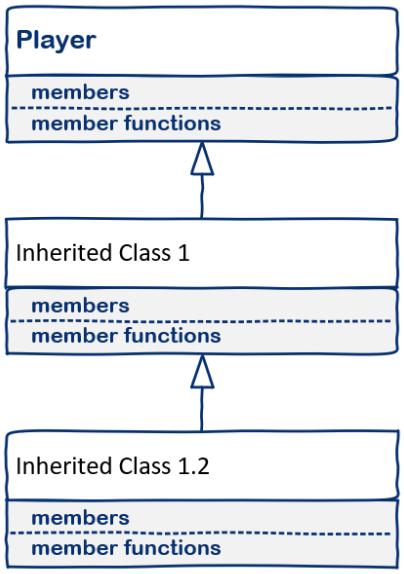
Overview and Requirements

This program will build upon the foundation of our ongoing “Player” class that has been covered during previous course lectures. For this assignment, you are tasked with using an Abstract Base Class version of the Player class and implementing multiple classes inheriting from this class (Base Class > Inherited Class 1 > Inherited Class 2). You are also tasked with implementing a class named “PlayerInventory” that will implement a linked list version of (your choice) a doubly linked list, a stack, or a queue (see requirements for each). Finally, you must implement a “print in forward order” function for your list using recursion.

See the “main tasks” for an overview of the program flow and output.

Required Classes

See the starter code and the below descriptions of the required class definitions




```
class PlayerInventory {}
```

This is your "list" class. You will choose a doubly linked list, stack linked list, or queue linked list to implement for this class.

Doubly Linked List Requirements:

This is a list that is similar to the singularly linked list except each node also keeps track of the previous node.

- insertAtPosition()
  - inserts at the specified "index" in the list
  - if the index is not valid, it inserts it at the front of the list
- removeItem()
  - removes the specified item from the list
  - does nothing if the item is not found
- displayAllItems()
  - displays all items to the console in forward order (recursively)
- erase()
  - removes all items from the list
- size()
  - determines and returns the size of the list

Stack Linked List Requirements:

This is a LIFO (last in, first out) list where we popping removes the last item that was added to the list.

- push()
  - pushes an item onto the "top" (back) of the stack
- pop()
  - removes an item from the "top" (back) of the stack
- displayAllItems()
  - displays all items in forward order (recursively)

- empty()
  - removes all items from the list
- size()
  - determines and returns the size of the list

Queue Linked List Requirements:
This is a FIFO (first in, first out) list where we add to the back of the list (enqueue) and remove from the front of the list (dequeue).

- enqueue()
  - adds an item to the back of the list
- dequeue()
  - removes an item from the front of the list
- displayAllItems()
  - displays all items in forward order (recursively)
- clear()
  - removes all items from the list
- size()
  - determines and returns the size of the list

`class Player {}`
This is your Abstract Base Class. See the starter code for the required details. You must implement the incomplete class member functions.

`class InheritedClass1 {}`
This is a derived class from the Player class. You are free to design this as necessary to complete your version of the demonstrated tasks. *Please do not call your class "InheritedClass1" …* 😑

`class InheritedClass2 {}`
This is a derived class from the InheritedClass1. You are free to design this as necessary to complete your version of the demonstrated tasks. *Please do not call your class "InheritedClass2" …* 😒

Note that you are not allowed to modify provided starter code definitions.

Required Variables

See the starter code and the below descriptions of the required variables. Beyond the provided variables, you are free to create your own.

Reminder: **you are not allowed to use non-constant global variables for your solution**. All modifiable variables must be within scope of main() or your programmer-defined functions.

Required Functions

There are no required functions outside of the class definitions of class member functions. *You are still required to create additional functions in order to demonstrate proper top-down design* (see **Gaddis Chapter 1.6 The Programming Process** and the **Software Development Method** sheet attached to this assignment) demonstrating modular programming with the use of functions (see **Gaddis Chapter 6 Functions**).

Specifications

Main tasks:

Note: There are additional directions in the comments on the starter code.

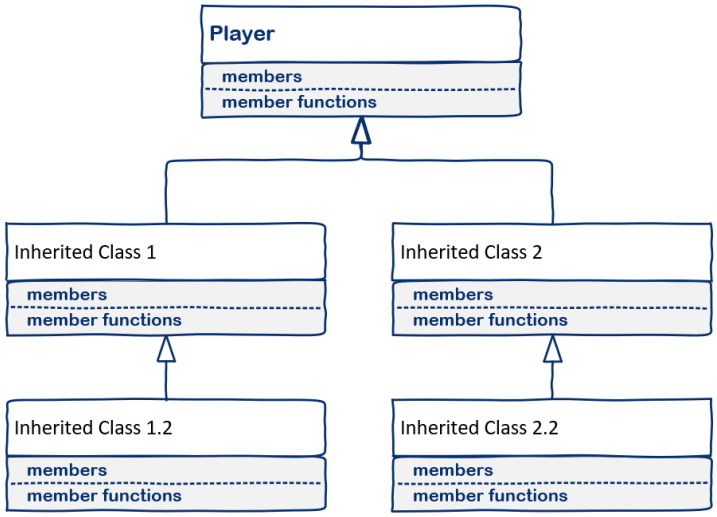
1. Declare and initialize an instance of InheritedClass1 and InheritedClass2
    a. Note: be creative! You do not have to follow my "Wizard" example as long as you meet the requirements
2. Demonstrate use of your list class object -> insertion
    a. "fill up" each of your players' inventory with at least 10 items from your possibleItems list. These should be chosen randomly. Duplicates are fine.
    b. What are you filling up? Each player will have a protected member "playerInventory" that is your "list" (doubly linked list, stack linked list, or queue linked list) where you can add and remove "items" from.
3. Demonstrate use of your class object -> printing contents
    a. Reminder: printing your list contents should be done in forward order via a recursive function
    b. Before starting the 'game' print out the contents of one player's list.
4. Demonstrate use of your list class object -> deletion/removal
    a. Print the size before and after.
    b. remove an item from the list.
5. Demonstrate use of your class object -> printing contents
    a. Reminder: printing your list contents should be done in forward order via a recursive function
    b. print the contents of the list again.
6. Game Time!
    a. print out their starting info
    b. force the two class objects to fight each other!
7. at the start of each turn, display their health
    a. for each turn, display
        i. who is attacking who
        ii. which weapon they are using
        iii. hit and damage (or miss)
    b. damage should be deducted from target player health
    c. hits should grant experience to the attacking player
    d. when one player health drops to 0 or lower, the 'game' ends.
8. display a message indicating the winner and display the ending player info

Tip: I encourage you to use the software development method to plan out your algorithm to solve this problem and ***incrementally code-compile-test each portion of your program*** as you implement it! It may take a little longer at the start but I promise you that it will save you time and headache in the long run!

(Bonus 10 pts)

You must state in your top level comment block that you are attempting the extra credit.

Implement a secondary derived class and specialized derived class. Demonstrate it by including a round 2 "fight" between InheritedClass 1.2 and InheritedClass 2.2 after the required "fight" between InheritedClass1 and InheritedClass 1.2. (see the image below)



Submitting Assignments

1. Submit your assignment to the Canvas course site. You will be able to upload your three source files (**two .cpp files and one .h file**) to the PA assignment found under the Assignments section on Canvas. You are ***REQUIRED*** to use the three-file format for this submission. Use the "+ Add Another File" link to allow choosing of three files (see reference image below).

2. **Your project must compile/build properly.** The most credit an assignment can receive if it does not build properly is 65% of the total points. Make sure you include all necessary libraries, e.g. string, ctime, etc. (though this does not mean add every single library you've ever used!)
3. Your submission should only contain your three source files (plus any bonus submission files). You may submit your solution as many times as you like. The most recent submission is the one that we will grade. *Remember to resubmit all three files if you submit more than once.*

4. Submit your assignment early and often! You are NOT penalized for multiple submissions! We will grade your latest submission unless you request a specific version grade (request must be made prior to grading).

5. If you are struggling with the three-file format, I recommend you complete the program in one file and submit that (working) version first, then convert it to the three-file format. **A working one-file format program is worth more points than a broken three-file format program!**

Note: *By submitting your code to be graded, you are stating that your submission does not violate the CPSC 122 Academic Integrity Policy outlined in the syllabus.*

Grading Guidelines

This assignment is worth 100 points. Your assignment will be evaluated based on a successful compilation and adherence to the program requirements. We will grade according to the following criteria:

- 20 points for implementation of your list class
    - 3 pts for Insert
    - 3 pts for Remove
    - 10 pts for Display recursively
    - 2 pts for Empty
    - 2 pts for Size
- 5 points for demonstrating list class
- 20 points for implementing Abstract Base Class functions
- 20 points for implementation of InheritedClass 1
- 10 points for implementation of InheritedClass 1.2
- 10 points for main tasks properly outputting to the console
- 5 points for using the three-file format and guard code
- 10 pts for adherence to proper programming style and comments established for the class and for demonstration of top-down design and modular programming using functions

Expected Console Output *(Note: feel free to customize the output as long as the required information is present)*

```
Filled the list...
Demonstrating printing the list...
Weapon: Programming Assignment -  damage: 20
Weapon: Fireball -  damage: 5
Weapon: Candle -  damage: 2
Weapon: Absurd Object -  damage: 10
Weapon: Fish -  damage: 0
Size: 5
Demonstrating Removing an item from the list...
Weapon: Programming Assignment-  damage: 20
Weapon: Fireball-  damage: 5
Weapon: Candle-  damage: 2
Weapon: Absurd Object-  damage: 10
Size: 4

Welcome to the battle arena!
Today we will be seeing a battle between two Wizards!
Our first participant is:

>>>>>>>>>>>>>>>>>>>>>>>>|PID:176|
>>Enomaex's info
Health: 150
Experience: 0
---------------------------------
Class: Wizard
Mana: 69
_________________________________
```

InheritedClass 1 Info
printPlayerInfo() output

```
And our second participant is:

>>>>>>>>>>>>>>>>>>>>>>>>|PID:57|
>>Nobess's info
Health: 83
Experience: 0
---------------------------------
Class: Wizard
Mana: 179
Specialization: fire
Damage Multiplier: 2.0
_________________________________

Let the battle commence!

|------- Turn 1 -------|
Enomaex's Health: 150 - Nobess's Health: 83
Enomaex attacks Nobess with a(n) Fireball! ...hits for 5!
Nobess attacks Enomaex with a(n) Fish! ...hits for 0!

|------- Turn 2 -------|
Enomaex's Health: 150 - Nobess's Health: 78
Enomaex attacks Nobess with a(n) Fish! ...hits for 0!
Nobess attacks Enomaex with a(n) Fireball! ...hits for 10!

|------- Turn 3 -------|
Enomaex's Health: 140 - Nobess's Health: 78
Enomaex attacks Nobess with a(n) Fireball! ...hits for 5!
Nobess attacks Enomaex with a(n) Absurd Object! ...hits for 20!

|------- Turn 4 -------|
Enomaex's Health: 120 - Nobess's Health: 73
Enomaex attacks Nobess with a(n) Fireball! ...hits for 5!
Nobess attacks Enomaex with a(n) Candle! ...hits for 4!
```

InheritedClass 1.2 Info
printPlayerInfo() output

printBattleCard() output

```
|------- Turn 5 -------|
Enomaex's Health: 116 - Nobess's Health: 68
Enomaex attacks Nobess with a(n) Fireball! ...misses!
Nobess attacks Enomaex with a(n) Programming Assignment! ...hits for 40!

|------- Turn 6 -------|
Enomaex's Health: 76 - Nobess's Health: 68
Enomaex attacks Nobess with a(n) Fireball! ...hits for 5!
Nobess attacks Enomaex with a(n) Programming Assignment! ...hits for 40!

|------- Turn 7 -------|
Enomaex's Health: 36 - Nobess's Health: 63
Enomaex attacks Nobess with a(n) Candle! ...hits for 2!
Nobess attacks Enomaex with a(n) Absurd Object! ...hits for 20!

|------- Turn 8 -------|
Enomaex's Health: 16 - Nobess's Health: 61
Enomaex attacks Nobess with a(n) Fireball! ...hits for 5!
Nobess attacks Enomaex with a(n) Fish! ...hits for 0!

|------- Turn 9 -------|
Enomaex's Health: 16 - Nobess's Health: 56
Enomaex attacks Nobess with a(n) Candle! ...hits for 2!
Nobess attacks Enomaex with a(n) Fish! ...hits for 0!

|------- Turn 10 -------|
Enomaex's Health: 16 - Nobess's Health: 54
Enomaex attacks Nobess with a(n) Absurd Object! ...hits for 10!
Nobess attacks Enomaex with a(n) Programming Assignment! ...hits for 40!

Nobess wins!

Ending Info:

>>>>>>>>>>>>>>>>>>>>>>>>|PID:176|
>>Enomaex's info
Health:
       ,-=-.
      /  +  \
      | ~~~ |
      |R.I.P|
      |_____|

Experience: 90
---------------------------------
Class: Wizard
Mana: 69
_________________________________

>>>>>>>>>>>>>>>>>>>>>>>>|PID:57|
>>Nobess's info
Health: 44
Experience: 100
---------------------------------
Class: Wizard
Mana: 179
Specialization: fire
Damage Multiplier: 2.0
_________________________________
```

InheritedClass 1 Info
printPlayerInfo() output

InheritedClass 1.2 Info
printPlayerInfo() output