Review 2

CS202 Programming Systems, Summer 2020

Armant Touche

Date: July, 26, 2020

Design Analysis

The task for this assignment was to create a game system where the requirement for implementation was to use dynamic binding with a abstract base class. For my abstract base class, I had five function in total with a default constructor and destructor. One of my function wasn't purely virtual so I had to used run-time type identification for which of the derived games were to play. Firstly, I choose to implement the Game stand alone because I felt that the more time would be require for the game system and player list in that order. I made sure to push up self-similar into the abstract base class to create a better, cleaner interface. Once I had the abstract base class and derived class prototyped function implemented, I moved onto creating the player list. The player list was a linear linked list where each node encapsulated the player's data (name) stored per node. Then, in each player-node had a dynamic player pointer which is derived from game-list, which is a circular linked list of games. Each player is a player of games and can add games and play games where the selection ranges from number-sequence, guess a number, and whack a number.
The game list, I have add-game function, play-game function, insert function, and destroy (destructor helper-function). Adding a game, the player decides between and chooses, then add one game to the circular linked list. To play game, I find player within the linked list, then I detach player from list to re-insert later sorted highest score to smallest. To find player, I let player input the gamer-tag and compare to the name in the player list for match. For re-insertion into list, start from head of the list (highest rank portion) and traverse to end of list (lowest rank portion), comparing to each player-node's score. If pre-insertion comparison results with the node being compared from the list being less than or equal to (boolean value), then the removed-node is inserted before the listed-node. This happens one time before each found player plays one game from their list (CLL) of games. The choice for choosing whose turn it is, is free flowing but of the players wanted to take turns (strict), then the player(s) can keep track of name searches as they're being inputted. Game-node class consisted of three main functions and self-referencing pointer and a dynamic game pointer for each game in the circular linked list. The get-next function returns a game-node pointer by reference. The set-next function (void) intakes a game-node pointer to set next to. The play-game function returns the score from the game that was just played by the player to game-list, which will ultimately be used for score keeping in each player-node in the player-list. The game-list class has four member functions and four data members: (i) number of games in list (ii) number of games played by player (iii) game-node pointer called new-game-rear (iv) game-node pointer called game-to-play (for tracking which game to play in CLL). In the player class (derived from game-list) has six member functions and two data members. Read-in function reads in from user their name into a dynamic character array. Add-game function calls parent's function, which is the same case for play-game. Is-match does a string compare and return a boolean value based on supplied-match. The delete-read is more for reusing the player data-type in main for re-usage (rid of memory leaks). Player-node has seven member functions and three data members. Set-next function and get-next function works just like the set-next function and get-next function from the game-node class. Get-score function returns the score as an integer value.

Debugger

GDB was perhaps the biggest time saver because because working with different data structure was a task. Stepping through GDB really solidified how dynamic binding work and was especially important when tracing insertion into both the data structures we were assigned to implement. Another debugging program that aided me was valgrind, which helped with memory leaks. 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.