7: A Circular Linked List

CSCI 6626 / 4526 March 2018

1 Goals

- To use C++ pointers to build a linked container class.
- To use either a friend declaration or an inner class to implement a one-many relationship.
- To implement a data structure as a template so that it can be reused.

2 The CList Class

The CList (circular linked list) class is a template container class (or, in Java terminology, a collection class) that can hold objects of any specified base type. The objects are kept in Cells that are linked together in a circle. Each Cell holds one Player*. CList forms the only public interface for itself and for class Cell.

Data members. A CList has four data members:

- A counter, to store the number of players in the list.
- A head pointer. Initially, this pointer should be NULL. When the first player is created, a Cell will be created for that player and the head pointer will point at the new Cell. The `next` field of that Cell will point to itself, forming a circular structure that will be maintained by all additions to the list. As players join the game, Cells are created for them and added to the list.

  When a player quits in mid-game, his Cell is removed from the list. If the first player quits, the head pointer will be set to point to the next player in the list.
- A tail pointer. is needed in the CList class to keep the order of the players in the list the same as the order in which they arrive. The tail pointer should always point at the most recently inserted cell.
- A pointer to the current player. This is set to the player at the head of the list at the beginning of each game. Thereafter, it progresses around the circle whenever a player's turn ends. When the current player resigns, this pointer should be set to the next player on the list. The logic for taking turns must be written in such a way that this player gets the next turn.

Functions.

- A constructor. No parameters are needed. Initialize the list to empty.
- A destructor that will delete all the cells in the list.
- An accessor function, `count()`, that returns the number of players in the list.
- A function, `bool empty()`, that returns `true` if there are 0 players in the list. `false` otherwise.
- A `print()` function and `operator<<` that will print all the players in the list. Start at the first player and be careful to stop when you get to the head of the list again. Delegate the printing to the Player class.
- A function `void insertBack( Player* )` that will build a new Cell and insert the new Cell at the end of the CList. The special case for insertion of the first player is explained above.

Note: This is a general-purpose container class, so it is not appropriate to limit the number of Players that can be added to the list. Although Can't Stop is limited to 4 players, the limitation is a task for the Game class, not for the CList class.

- A function `Player* first()` that sets the current pointer to the Cell at the head of the CList and returns a pointer to the Player in that Cell. Implement an appropriate behavior for an empty list. DO NOT ABORT execution if this happens.
- A function `Player* next()` that moves the current pointer to the next Cell in the CList and returns a pointer to the Player in that Cell. This function will be called if a player stops or goes bust. Implement an appropriate behavior for an empty list. DO NOT ABORT execution if this happens.
- A function `void remove()` that will remove the current player (and the corresponding Cell) from the list, deleting both. Set the `current` pointer to the next cell in the list. If the cell that was removed was the head cell, `head` must be set to the next cell in the list. This function will be called if a player chooses to quit (instead of roll or stop) during his turn.

3 The Cell Class

This is a servant class that works with CList as its master. Because of that, it is a very simple class. You may choose one of these implementations:

- Friend Class: All members (data and functions) of this class will be private, and the class will give friendship to CList. This architecture leads to a neater and less confusing code layout. This friendship is necessary because CList must have access to the pointers that form the list.

  If you use a friend class, please put the Cell class and the CList class in the same file. To do this, start with a forward declaration for CList, then follow with the class declaration for Cell, and end with the class declaration for CList. A forward declaration looks like this:

  ```
  class PlayerList;
  ```
- Inner Class: All members (data and functions) of this class will be public, but the class definition, itself, will be inside the CList class, in the private part of the class. This architecture is harder to parse but a little easier to make into a template class.

You will need two data members and the usual functions for Cell:

- Data member: a Player*, used to hold the data.
- Data member: a Cell*, used to form the linked list.
- A constructor with two parameters (a Player* and a Cell*). Provide a default value of NULL for the second parameter.
- A destructor that will delete the Player that is attached to the Cell.

4 Testing

In the next phase of the program, the CList will be integrated with the Game class. This assignment asks only to build and test the CList as a container for Players.

Write a unit test that will test the list class with the Player class. Call it from main(). In your unit test, call a private function to create four dummy Players and use them to test all the CList functions. Give me test output to prove that:

- You can add Players.
- You can remove a Player and the list remains unbroken and functional.
- After removing all of the players, you can add Players again.
- The `next()` function can progress around the circle and reach the first player again.
- You can print an empty CList or a list of 2 to 4 players.

5 Make it into a Template

When the entire class is debugged, go back through the module's two files and make the changes necessary to make the class into a template. Declare a CList as a data member of your Game class and instantiate the template in your Game constructor. Remove the declarations individual players. Debug again. Turn in the program and debugging output after this step.