BLG 252E Object Oriented Programming, Spring 2017

Homework #2 (Adder and Remover, due April 29 at 23:55)

1 Introduction

In this assignment, you will implement an adder remover in which the program will implement the primary operations for the different container classes to handle the storage and processing of data.

2 Container (Data Storage) Classes

A container (collection) is a way that organizes storage of data in memory.

The primary operations for the container classes which you are asked to implement are:

- **add:** The function for adding element to the appropriate place of the list.
- **remove:** The function for removing element from the appropriate place of the list.
- **removeAll:** The function for deleting all elements in the list.
- **display:** The function for printing AdderRemover list elements from first element to last element to the screen.
- **setTraverser:** The function for setting/initializing traverser property to desired traverser strategy.
- **traverse:** The function for printing AdderRemover list elements using traverser.

3 The Problem Statement

You are going to implement different container classes to store data of integer. You will use **doubly linked list representation** to store elements. You are not allowed to use any STL containers in this homework. You should create your own classes.

First of all, define **AdderRemover** base class depicted in Figure 1 to extend classes and write polymorphic functions. Its add, remove and setTraverser methods are **pure virtual** since there are different strategies to add, remove and traverse elements from its internal list. This internal data structure is a doubly linked list of **integer** elements. However, derived classes will directly use inherited display, removeAll and traverse methods.

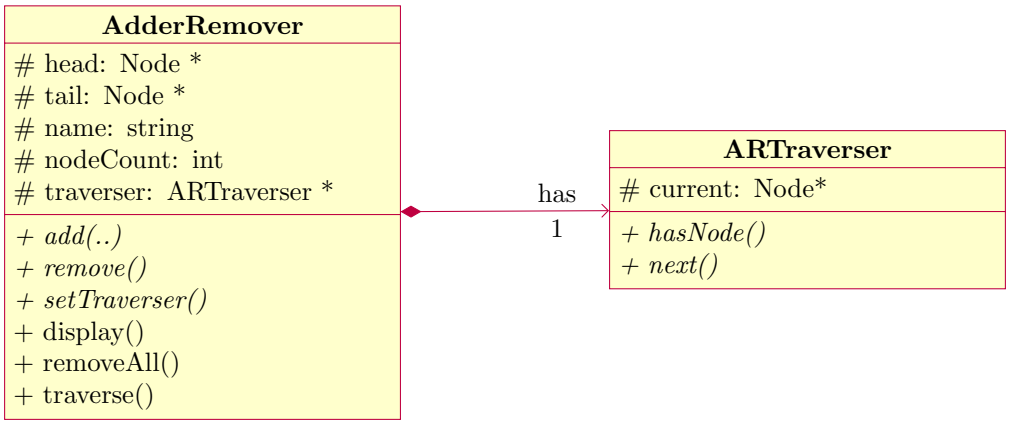

Figure 1: Structure of the AdderRemover and ARTraverser Classses.

Some of the strategies (classes) for adding and removing elements from their internal lists are the following :

1. **First Add First Remove (FAFR):** When a new elements to be added, it adds to the beginning of the list, an element to be removed it removes from the beginning of list.

2. **First Add Last Remove(FALR):** When a new elements to be added, it adds to the beginning of the list, an element to be removed it removes from the end of list.
3. **Last Add First Remove (LAFR):** When a new elements to be added, it adds to the end of the list, an element to be removed it removes from the beginning of list.
4. **Last Add Last Remove (LALR):** When a new elements to be added, it adds to the end of the list, an element to be removed it removes from the end of list.

Secondly, define **ARTraverser** abstract class. It offer a way of getting sequential access to all elements of a container class. It will be used to "read" data from a container. Its two methods:

- **hasNode:** Returns true if the list has element.
- **next:** Returns the current element in the list and advances the current position.

are **pure virtual** since they should be implemented appropriately for each type of strategies. Two strategies will be defined:

1. **First AdderRemoverTraverser (ARTF):** It is used to traverse through a container from the beginning element to the end element. *FAFR* and *FALR* classes are going to use this strategy as its traverser.
2. **Last AdderRemoverTraverser (ARTL):** It is used to traverse through a container from the end element to the beginning element. *LAFR* and *LALR* classes are going to use this strategy as its traverser.

4 Main Program

In main program, declare an array of pointers for base class AdderRemover. The pointer array will be a polymorphic array which will store memory addresses of four different types of AdderRemover objects (FAFR, FALR, LAFR, LALR). Figure 2 shows an example of the objects involved. You would add/remove elements to polymorphic array FAFR, FALR, LAFR, LALR instances in order to test whether the functions work well or not. Your program should print the name, nodeCount and the elements of internal list as separate groups (Figure 3).

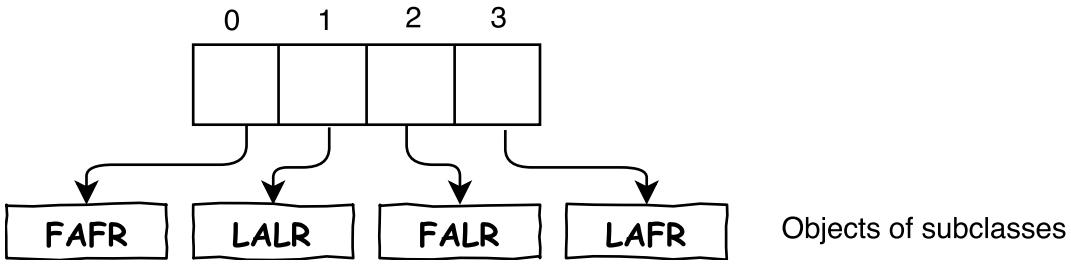

Figure 2: Pointer array of AdderRemover base class

5 Implementation Notes

- Consider OOP approach in your design with well-chosen variable, method, and class names and comments where necessary. You would get **0** from implementation that do not contain **inheritance** and **polymorphism**.
- Your program should compile and run on **Linux** environment using **g++ (version 4.8.5 or later)**. You can test your program on ITUs Linux Server using SSH protocol.
- Your homework assignments SHOULD NOT include any copy-paste material (from the Internet or from someone elses paper/thesis/project). Check the Academic honesty section in the syllabus.
- For any questions about the assignment, contact Hasan Kivrak directly (office no: Research Lab3) or via mail (kivrakh@itu.edu.tr)

6 Test Code

Your program will be tested using below test code and expected to have desired output.

```
int main(int argc, char **argv) {

  AdderRemover *myList[4];

  myList[0] = new FAFR();
  myList[1] = new LALR();
  myList[2] = new FALR();
  myList[3] = new LAFR();

  for(int j=0; j<4; j++) {
    for(int i=0; i<5; i++)
      myList[j]->add(i);
  }

  for(int j=0; j<4; j++){
    myList[j]->remove();
    myList[j]->display();
    myList[j]->removeAll();
    myList[j]->display();
  }

  cout << endl << "Test case for ARTraverser"<< endl;
  for(int j=0; j<4; j++) {
    for(int i=0; i<5; i++)
      myList[j]->add(i);
    myList[j]->remove();
  }

  for(int j=0; j<4; j++){
    myList[j]->setTraverser();
    myList[j]->traverse();
  }

  return 0;
}
```

```
FAFR | NodeCount:4
------
3
2
1
0

FAFR | NodeCount:0
------
There is no element to print

LALR | NodeCount:4
------
0
1
2
3

LALR | NodeCount:0
------
There is no element to print

FALR | NodeCount:4
------
4
3
2
1

FALR | NodeCount:0
------
There is no element to print

LAFR | NodeCount:4
------
1
2
3
4

LAFR | NodeCount:0
------
There is no element to print

Test case for ARTraverser

FAFR | NodeCount:4
------
3
2
1
0

LALR | NodeCount:4
------
3
2
1
0

FALR | NodeCount:4
------
4
3
2
1

LAFR | NodeCount:4
------
4
3
2
1
*** Exited normally ***
```

Figure 3: Example Test Code and Output