CS221-02: Comp Sci II: Data Structures

Spring 2019

Programming Assignment 2

1.0 Overview

Linked List is an important data structure which have been used in a lot of software written in C++. This programming assignment provides a basic introduction to the creation and use of linked lists.

Main idea:

This assignment is similar to the first assignment. You will create two linked lists of users: sorted user list based on birthdate and unsorted list as appears in the file.

2.0 Requirements

2.1 (Struct NodeType) Implement NodeType struct. NodeType will have two fields: i) User* user and ii) NodeType* next.

2.2 (Class UnsortedType) Implement UnsortedType class provided in Section 3.4 with modifications stated below. Create the specification file (UnsortedType.h) and implementation file (UnsortedType.cpp). The original linked list was implemented for Nodetype having ItemType. In this assignment, it will be implemented for NodeType mentioned in 2.1 of this assignment. There will be some differences in the methods of this class.

- **(Constructor)** `UnsortedType();`
- **(Transformer)** `void MakeEmpty();`
- **(Observer)** `bool IsFull() const;`
- **(Observer)** `int GetLength() const;`
- **(Transformer)** `void ResetList();`

Instead of items, you will work on users. So Item related methods are replaced with user. GetUser searches name of a user and then returns the node of the user. You work with pointers to user.

- **(Observer)** `User* GetUser(char fname[],char lname[],bool& found) const;`
- **(Transformer)** `void PutUser(User* user);`
- **(Transformer)** `void DeleteUser(User* user);`
- **(Observer)** `User* GetNextUser();`

Display the users in this list using the following method.

- **(Observer)** `void Print() const;`
- **(Observer)** `void Print(std::ofstream& out) const;`

Print() method will call Display() method of User, whereas PrintPartial() will just get the name of user and print the name of user.

2.3 (Enhance DateType) Specify and implement the comparedTo function in Section 2.3

- `RelationType comparedTo(DateType aDate) const;`

2.4 (Enhance Class User) Specify and implement the comparedTo function for User class based on dateOfBirth field.

- `RelationType comparedTo(User* aUser) const;`

This method will compare the birthdate of the current user with the birthdate of aUser.

2.5 (Class SortedType) Implement SortedType class provided in Section 4.3 with modifications stated below. Create the specification file (SortedType.h) and implementation file (SortedType.cpp). The original linked list was implemented for Nodetype having ItemType. In this assignment, it will be implemented for NodeType mentioned in 2.1 of this assignment. User data will be inserted based on dateOfBirth of users. There will be some differences in the methods of this class.

- **(Constructor)** `SortedType();`
- **(Transformer)** `void MakeEmpty();`
- **(Observer)** `bool IsFull() const;`
- **(Observer)** `int GetLength() const;`
- **(Transformer)** `void ResetList();`

Instead of items, you will work on users. So Item related methods are replaced with user. GetUser searches the name of a user and then returns the node of the user. You work with pointers to user.

- **(Observer)** `User* GetUser(char fname[],char lname[],bool& found) const;`
- **(Transformer)** `void PutUser(User* user);`
- **(Transformer)** `void DeleteUser(User* user);`
- **(Observer)** `User* GetNextUser();`

Display the users in this list using the following method.

- **(Observer)** `void Print() const;`
- **(Observer)** `void Print(std::ofstream& out) const;`

  Print() method will call Display() method of User, whereas PrintPartial() will just get the name of user and print the name of user.

2.5 (Main function) Declare **allUnsortedUsers** of **UnsortedType** and **allSortedUsers** of **SortedType**. All users will be inserted to each list. Before completion of main(), all members of the list should be deleted using MakeEmpty(). MakeEmpty() function should delete Users that are already created. DeleteUser() only deletes the node not the actual user.

2.12 (Testing) Enhance TestDriver class. Create the specification file (TestDriver.h) and implementation file (TestDriver.cpp). This file should have

- a constructor,

- **int Populate(const char* input,UnsortedType& allUnsortedUsers) const** method to read user profiles from input file "input" into the list of **allUSers**.
- **int Populate(const char* input,SortedType& allSortedUsers) const** method to read user profiles from input file "input" into the list of **allSortedUsers**.

  The file should be formatted as follows as an example:

```
FirstName: John
LastName: Rich
Major: CS
Gender: M
Email: jrich@uah.edu
StreetName: Triana
StreetNo: 188
City: Huntsville
Zip: 35806
State: AL
GPA: 3.54
BirthDate: 9 10 1996
-----
FirstName: Alice
LastName: Rice
Gender: F
Major: MATH
StreetName: Hughes
StreetNo: 76
City: Nashville
Zip: 42163
State: TN
GPA: 3.65
Email: alicerice@hotmail.com
BirthDate: 8 15 1998
```

Add one more function **void TestList(UnsortedType& allUnsortedUsers) const** for testing experiments regarding lists. In order to test the class functions TestList() should evaluate the new functions for the User class. Similarly add another function **void TestList(SortedType& allSortedUsers) const** for testing experiments regarding lists. In order to test the class functions TestList() should evaluate the new functions for the User class.

Add another test function that writes results to an output stream file as **void TestList(std::ofstream& outFile,UnsortedType& allUnsortedUsers) const** and **TestList(std::ofstream& outFile,SortedType& allSortedUsers) const**.

You may call these functions in your **void Test (SortedType& allSortedUsers) const** and **void Test(std::ofstream& outFile,UnsortedType& allUsers) const.** Similarly, you may test these functions for the sorted list using **void Test (SortedType& allSortedUsers) const** and **void Test(std::ofstream& outFile,UnsortedType& allUsers) const.**

The main() function should create an object of TestDriver and call relevant Test function.

Note: Neither your instructor nor the course GTA is your tester. You should develop a testing strategy and test every method in your code. Error detected during grading will result loss of points. Make sure that you test your program carefully. We will go over testing strategies in class.

***** BONUS *** (20% Bonus)** You may use inheritance and implement SortedType as a derived class of UnsortedType class. This also allows one version of functions in Tester rather a function for UnsortedType and another one for SortedType.

3.0 Commenting Code

Your code should be commented well. Each method should have preconditions and post conditions listed after method declaration in the implementation files.

4.0 Deliverables

These products shall be delivered to the instructor electronically via Canvas as specified below.

4.1 Program source files -- The student shall provide fully tested electronic copies of the .cpp and .h files. These files should be uploaded to Canvas as a **single zip file**.

4.2 Software Document – The student shall provide general description of the software in a structured way and test plan as mentioned in Programming Assignments document. Results of test experiments should be provided. If class members are updated, they should be displayed after being updated. Only test the new methods.