Assignment

Assignment # 3

Unsorted Linked List

Make sure you have read and understood

- ***lesson modules week 4 and 5***
- ***chapters 4 and 5 of our text***
- ***Coding Style Guidelines (module week 1)***

before submitting this assignment. Hand in only one program, please.

Background:

In many applications, the composition of a collection of data items changes over time. Not only are new data items added and existing ones removed, but data items may be duplicated. A list data structure is a member of the general category of abstract data types called containers, whose purpose is to hold other objects. In C++, lists are provided in the Standard Template Library. However, for this assignment you will design and write your own linked list implementation to support the ADT operations specified below.

Objective:

Design and implement the specifications for a List Abstract Data Type where the items in the list are unsorted.

Requirements:

Define a list and develop a set of operations for creating and manipulating a list that satisfies the list ADT specification.

List ADT Specification

Structure: The list elements are of ItemType. The list has a property called the *current position* which designates the position of the last element accessed by GetNextItem during an iteration through the list.
Only ResetList and GetNextItem alter the *current position*.

Definitions (provided by the user):

MAX_ITEMS: A constant specifying the maximum capacity of items allowed on the list

Item Type: Class encapsulating the type of items in the list

RelationType: An enumeration type that consists of LESS, GREATER, EQUAL

Member function of ItemType that must be included:

RelationType ComparedTo(ItemType Item)

Function: Determines the ordering of two ItemType objects based on their keys

Precondition: Self and item have their key members initialized

Postcondition:

Function value = LESS if the key of self is less than the key of item

= GREATER if the key of self is greater than the key of item

= EQUAL if the keys are equal

Operations (provided by Unsorted List ADT)

Make Empty

Function: Initializes list to empty state

Preconditions: None

Postcondition: List is empty

Boolean IsFull

Function: Determines whether list is full

Preconditions: List has been initialized

Postcondition: Function value = (list is full)

int GetLength

Function: Determines the number of elements in list

Preconditions: List has been initialized

Postcondition: Function value = number of elements in list

ItemType GetItem(Item Typeitem, Boolean& found)

Function: Get list element whose key matches item's key (if present)

Preconditions: List has been initialized

Key member of item is initialized

Postconditions: If there is an element someItem whose keymatches item's key, then found = true and copy of someItem is returned; otherwise found = false and item is returned

List is unchanged

PutItem(ItemType item)

Function: Puts item into list

Preconditions: List has been initialized

List is not full

Item is not in list

Postcondition: Item is in the list

DeleteItem(ItemType item)

Function: Deletes the element whose key matches item's key

Preconditions: List has been initialized

Postcondition: One and only one element in list has a key matching item's key

ResetList

Function: Initializes current position for an iteration through the list

Preconditions: List has been initialized

Postcondition: Current position is prior to list

ItemType GetNextItem()

Function: Gets the next element in list

Preconditions: List has been initialized

Current position is defined

Element at current position is not last in list

Postconditions: Current position is updated to next position

Returns a copy of element at current position

UML

The **UML diagrams** for the linked class UnsortedList are included as an attachment.

Test Run Requirements: Only submit one run that shows a sample client that instantiates a list object, reads in test data from the **listData** test file and writes the test results to the**a3testFirstnameLastname** file (for FirstnameLastname *your* Firstname and Lastname).

Grading Criteria:

ItemType class is correctly defined and implemented.

UnsortedType class is correctly defined and implemented.

Program compiles and runs.

Implementation supports the operations given in the ADT functional requirements.

A test driver is included to satisfy the test run demonstration.

A copy of your test run output display is included in an output file named a3testFirstnameLastname

Be sure to include 6 separate files:

- ItemType.h
- ItemType.cpp
- unsorted.h
- unsorted.cpp
- listDr.cpp
- a3testFirstnameLastname