**Program 1: Move-to-front Lists**

**Definitions**

Move-to-front list (MTF): A list where every time an element has been accessed, it is moved to the front of the list. Elements in the list that are accessed more frequently will be near the front and therefore will result in fewer traversals across lists over time.

Example:

If node ten is accessed the most and a MTF is being used, node ten will be at the front of the list. This decreases the number of total traversals therefore increasing efficiency. Otherwise, if a MTF is not put in place, the list would have to traverse the list node by node until it has reached the 10th element and it would have to do this every time.

Below is a diagram of a linked list. If the integer in node C was accessed the most, it would be moved to the front of the list for the MTF list.

Diagram, box and whisker chart

Description automatically generated

The image below represents a MTF list. When the node with an integer 4 is searched, it is moved to the front.

Diagram

Description automatically generated

**Specification**

The program will have two different types of lists, a linked list and a MTF list. The purpose of the program is to store data in a linked list and an MTF list and compare the efficiency of accessing certain nodes within the lists. The efficiency will be defined as how many nodes on average are traversed when finding a particular element in the “vanilla” linked list vs the MTF list.

The problem to be solved is to compare the number of traversals of a uniform vs normal distribution of a linked list and a MTF list. The assumptions to be made are that the lists will be made of integers, they will inherit from the IList interface, and both lists will be composed of nodes. The MTF list will be a subclass of the LinkedList class that has the functionality described above in the definition. Special cases are as follows: having duplicate integers in a list (this will be ignored as requested by professor Stiber) and moving a node to the front which is already at the front.

The picture below describes a uniform vs normal distribution. With the uniform distribution, the performance of a sequential search should not improve. The normal distribution will peak at a certain value and drop of quickly but does continue to +/- infinity. With the MTFList, the traversals should be shorter for the normal distribution.

![A picture containing text, businesscard

Description automatically generated]()

The input data will come from the LinkedListStats.cpp and will provide integers to fill the lists. The output will be from the LinkedListStats.cpp file, which creates a list and performs various tests on it to compare how many nodes are traversed when using a normal distribution or uniform distribution. The output will let the user know how many nodes on average were traversed with the uniform and normal distribution of the list. The test will be run twice, once with the linked list and again with the MTF list, where the test will be modified slightly to accommodate the MTF list. There will be no error handling as there is no input for errors to occur.

**Design**

The program is composed of four classes and a struct**: IList, LinkedList, MTFList**, and a **Node** struct**. MTFList** inherits from **LinkedList** and **LinkedList** inherits from **IList**.

**IList:**

The IList is an abstract class that is the base class for LinkedList. It has a constructor that initializes the traverseCount to 0.

**Protected Variables:**

**int travserCount:** keeps track of how many nodes have been traversed in a list.

**Public:**

**Constructor:** The constructor initializes traverseCount to 0.

**Methods:** The rest of the methods in IList are all virtual and implemented in LinkedList.

**LinkedList:**

The LinkedList is derived from IList and implements all pure virtual methods from IList.

**Private:**

**Node \*head:** represents the head of the linked list.

**int itemCount:** keeps track of how many elements are in the list

**Public:**

**Constructor:** default constructor that initializes a linked list

**virtual getCurrentSize():** returns the number of elements in the list which is stored in itemCount.

**virtual isEmpty():** returns true if the list is empty.

**virtual add(int newEntry):** this functions takes in an integer value, newEntry, and creates a node (via the Node struct) that holds the integer and adds it to the beginning of the list.

**virtual remove(int anEntry):** this function takes in an integer value, anEntry, and searches through the list for the integer. If the integer is found, the node that contains the integer is removed.

**virtual clear():** deletes all nodes in the list.

**virtual contains(int anEntry):** this functions takes in an integer, anEntry, and returns true if the integer is found. While traversing through the list, traverseCount is incremented by 1.

**virtual getTraverseCount():** this functions returns the value of traverseCount.

**virtual resetTraverseCount():** this functions resets traverseCount to 0.

**MTFList:**

The MTFList is a subclass of LinkedList with a few added functions. The MTFList moves the most accessed data variables in a node to the front of the list.

**Public:**

**virtual bool contains(int anEntry):** this function is similar to the LinkedList function, except if anEntry is found in the list, the node containing anEntry is moved forward by one position via **moveToFront()** function.

**void moveToFront(Node \*\*nodeAccessed):** this function evaluates if the accessed item is at the front of the list already or not. If it is not, it calls **moveNodeToFront()** in order to move the node one position forward.

**bool moveNodeToFront(Node \*\*nodeAccessed):** this function moves the nodeAccessed to the front of the list.

**Node:**

The Node struct holds two data members that initialize a node.

**public:**

**int data:** holds the integer value

**Node\* next:** pointer to next node

**Constructor:** default, one that takes in an integer, number, and sets the first node to be it, and another that takes in an integer, sets it and also sets the next node, nextNode.

**void setItem(const int& newData):** sets the node’s data to a new integer, newData

**void setNext(Node\* nextNode):** sets the next node, nextNode

**int getItem() const**: returns the data value

**Node\* getNext() const:** returns the next node

**Implementation Plan**

**Unit Test:**

Individual tests will be performed on the following functions in the order of:

LinkedList:

1. isEmpty(): should return true
2. Add(): test to see if add works
3. isEmpty(): should return false
4. getCurrentSize(): returns the number of elements and is called before and after remove to ensure that remove works
5. Remove(): tests to see if remove worked
6. getCurrentSize(): test to see if the correct number of elements is returned
7. getTraverseCount(): ensures that traverseCount starts out at 0
8. contains(): return either true or false and increments traverseCount
9. getTraverseCount(): returns the vaue of traverseCount depending on how many traversals were made in the previous call contains()
10. resetTraverseCount(): resets traverseCount to 0
11. getTraverseCount(): ensures that the resetTraverseCount works
12. equals(): test to see if when a list is created with the copy constructer that they are equal
13. operator override??
14. Copy constructor

MTFList:

1. Contains(): test this after having added various elements to the list
2. getTraverseCount(): test to see what the traversalCount is after calloing the contains() method
3. moveToFront(): this function is called in the contains method and should move the node that had the integer in contains to the front of the list

**Integration Test:**

The integration test will be performed with the LinkedListStats.cpp file. This file initializes a list with random numbers ranging from 1 to 1000. It then loops through the list and calls the contains method 100000 times to compare a uniform distribution’s sequential search to a normal distribution’s sequential search via number of nodes traversed. This will be done with a linked list and an MTFlist.