

Data Structures and Algorithms

LAB #2

Linked Lists

Objectives

After this lab, the student should be able to:

- Implement a linked list data structure as a C++ class template
- Write member functions to perform common linked list operations
- Use linked lists in different applications

Linked Lists

Introduction

A linked list is a data structure consisting of nodes that hold data and are connected together with links. In a singly linked list, each node holds a data element as well as a link to the *next* element (Figure 3-1). The last link points to NULL to signal the end of the list.

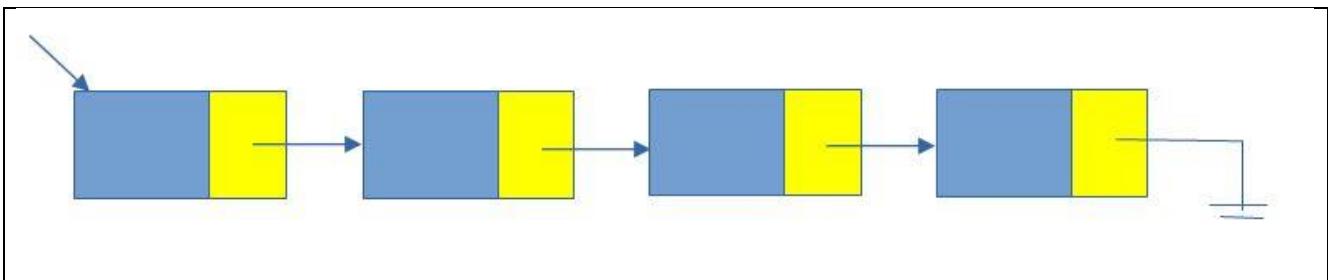


Figure 3-1 : A Singly Linked List

As the previous figure shows, a linked list is accessed via a pointer to its first element.

Because the node contains multiple primitive elements (i.e. the data and the link(s)), an appropriate way to represent it is a **class template**. The links can be represented by pointers, thus we have the definition listed in code (3-1).

```

#ifndef _NODE
#define _NODE
//First let's declare a single node in the list
template<typename T>
class Node
{
private :
    T item;          // A data item (can be any complex sturcture)
    Node<T>* next;   // Pointer to next node
public :
    Node( ); //default constructor
    Node( T newItem); //non-default constructor
    void setItem( T newItem);
    void setNext(Node<T>* nextNodePtr);
    T getItem() const;
    Node<T>* getNext() const;
}; // end Node
#endif

```

Code 3-1 : Node class

Suppose we have a pointer p defined as follows:

```
Node<double>* p; //A pointer that can point to a Node
```

Assume that p points to some node. To access the item member of this node we use “->” (arrow) operator as shorthand to access data member via a pointer as follows:

```
p->setItem(3.5);
```

Creating and Initializing a linked list

The above class represents a single node in a linked list. The linked list itself can be declared as another **class template** with one data member which is a Head pointer that points to the first Node in the list (NULL if empty list).

Note: you can also add the following data members depending on your application:

1- Count: an integer variable to hold the number of Node in the list

2- Tail: a pointer that points to the last node in the list.

See LinkedList.h file accompanied with this lab

To create a linked list we need to create an object of class template LinkedList.

```
LinkedList<string> L;
```

Passing a linked list to a function

To pass a linked list to a function, you need is to pass an object of LinkedList class template.

Important Note:


If you pass/return LinkedList by values, you should implement a **copy constructor** to the LiknedList class to operate properly.

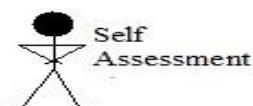
Linked List common operations

A linked list has many common operations such as: printing list, inserting and deleting nodes, searching list for certain key,.....etc

To support such operations, some member function should be added to the LinkedList class.

Example Code:

 **open Linkedlists.sln** and run the code then extend the code as specified in exercise 1 below



- ☐ Both array and linked list are possible representation of general lists. When should you use each?

Practice Exercises

Exercise 1:

Extend **LinkedList** class given in the code examples accompanied with this lab and implement the following member functions

Remember to handle ALL corner cases in your code

Function	Description
void InsertEnd(T data);	inserts a new node at end if the list
bool Find(T Key);	Searches for a given value in the list, returns true if found; false otherwise.
int CountOccurance(T value);	returns how many times a certain value appeared in the list
void DeleteFirst();	Deletes the first node in the list
void DeleteLast();	Deletes the last node in the list
bool DeleteNode(T value);	Deletes the first node with the given value and returns true. if not found, returns false Note: List is not sorted
bool DeleteNodes(T value);	Deletes ALL nodes with the given value and returns true. if not found, returns false Note: List is not sorted
void Merge(const LinkedList<T> &L);	Merges the current list to another list L Think: Can this be achieved by making the last Node in the current list point to the first Node in list L
void Reverse();	Reverses the linked list (without allocating any new Nodes)

Exercise 2:

It is required to declare a linked list to represent a polynomial.

$$\sum_{k=0}^n a_k X^k$$

- Each node in the list should represent one term of the polynomial. So a node should contain the coefficient and power of each term in addition to a link to the next node. You should declare a class **Term** (with members coeff & pwr) to be used as the Node **item**
- Declare a class **Polynomial** (**this is not a class template**) that contains:
 - Head pointer to the **Node<Term>** that contains the highest power,
 - degree (int to represent the degree of the polynomial)
- Add the following member functions to class **Polynomial**
 - Constructor to initialize data members
 - AddTerm(const & Term)** that adds a new term in its location in the polynomial
 - PrintPoly()** that prints the polynomial in the format: $5x^{12} + 4.3x^5 + 22$
 - getCoeff(int power)** returns the coeff of the term of the given power. It returns zero if such term doesn't exist.
 - setCoeff(double newCoeff, int power)** updates the coefficient of the term with the given power
 - if newCoeff = 0, the term should be deleted from the polynomial (if found)
 - if newCoeff != 0 and there is no term with this power, a new node should be created and added for that term
 - AddPoly(Polynomial P)** adds a polynomial to the current polynomial

Example:

If current polynomial is : $52x^4 - 7x^2 - 20$

And passed polynomial is: $30x^5 - 10x^4 + 2x^3 - 10$

The current polynomial should be updated to be: $30x^5 + 42x^4 + 2x^3 - 7x^2 - 30$

4. Write a program that:
 - a) Declares two polynomials P1 and P2 and fills them with arbitrary terms.
 - b) Prints P1 and P2.
 - c) Tests functions getCoeff and setCoeff
 - d) Adds the two polynomials and print the result

Exercise 3:

Extend the **LinkedList** class given in the code examples accompanied with this lab and implement the following member functions:

a) PrintKth

Write a function that prints data of the kth item in a linked list. The function should print "Beyond List Length" if K is beyond length of the list. First node index is 1.

b) InsertSorted

Given a sorted linked list, write a function that inserts a given data into its correct position. If data is already in the list, it is not inserted and the function returns false

c) RemoveMin

Write a function that extracts the node with the min data value in a linked list. The function should remove the node from the list and returns a pointer to it

Think: Can you use functions **InsertSorted** and **RemoveMin** to sort a given linked list

d) CloneList

Write a function that clones a given linked list into a new list and returns the new list.

e) SignSplit

Write a function that splits a given linked list **L** into two lists **Lneg** and **Lpos**. Where **Lneg** contains all -ve nodes and **Lpos** contains all +ve nodes. Nodes with zero data should be left in the main list **L**.

f) MakeDictionary

Write a function that makes a dictionary out of a given linked list. The dictionary is another linked list where each node contains data from original list plus number of occurrences of that data.

For example, given a linked list $L = a \rightarrow b \rightarrow k \rightarrow c \rightarrow a \rightarrow k \rightarrow c \rightarrow g \rightarrow a \rightarrow b \rightarrow \text{NULL}$, the new linked list should be $\{a, 3\} \rightarrow \{b, 2\} \rightarrow \{c, 2\} \rightarrow \{g, 1\} \rightarrow \{k, 2\} \rightarrow \text{NULL}$. The original list should be kept intact.

g) MergeSorted

Write a function that merges two sorted linked lists into a single sorted list. The merged list is returned and the two original lists should be empty.

h) SumLists

Write a function that takes two linked lists and creates (and returns) a third linked list that contains the sum of the two lists nodes. The two lists should be of the same length or

otherwise the function returns NULL.

i) Reorder_X

Write a function that reorders a linked list with respect to a value X. The list should be reordered so that all items less than or equals to X comes first before other items. Reordered list may be unsorted.

j) ShiftLargest

Write a function that shifts the largest node of a linked list to the end of the list. The function traverses the list comparing each two adjacent items together moving the larger item one step towards list tail. After one full list traversal, the largest item should be at the tail of the list. **Think:** Can you use this function to sort a linked list? How many times you need to call it to sort a linked list of N nodes?

k) RemoveDuplicates: Write a function that removes all duplicates from unordered list.