|  |  |  |
| --- | --- | --- |
|  | Data Structures and Algorithms  LAB #3 - Linked Lists | Fall 2018 |

Objectives

After this lab, the student should be able to:

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

1. **Linked Lists**
2. **Introduction**
3. 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.

|  |
| --- |
| 1. List.JPG |
| 1. Figure 3-1 : A Singly Linked List |

1. As the previous figure shows, a linked list is accessed via a pointer to its first element.
2. Because the node contains multiple primitive elements (i.e. the data and the link(s)), an appropriate way to represent it is a class. The links can be represented by pointers, thus we have the definition listed in code (3-1).

|  |
| --- |
| #ifndef \_NODE  #define\_NODE  class Node  {  private :  int Item; // A data item  Node\* next; // Pointer to next node  public :  Node();  Node( int newItem);  void setItem(int newItem));  voidsetNext(Node\* nextNodePtr);  int getItem() const ;  Node\* getNext() const ;  }; // end Node   1. #endif |
| 1. Code 3-1 : Node class |

1. Suppose we have a pointer p defined as follows:

**Node\* p;** //A pointer that can point to a Node

1. 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);**

1. **Creating and Initializing a linked list**
   * + 1. The above class represents a single node in a linked list. The linked list itself is declared as another class that has two data members
2. Head pointer that points to the first Node in the list (NULL if empty list).
3. Count: an integer variable to hold the number of Node in the list
4. See LinkedList.h and LinkedList.cpp files accompanied with this lab
5. To create a linked list we need to create an object of class LinkedList.
6. **Passing a linked list to a function**
7. To pass a linked list to a function, all what you need is to pass an object of LinkedList class
8. **Linked List common operations**
9. A linked list has many common operations such as: printing list, inserting and deleting nodes, searching list for certain key,……etc
10. To support such operations, some member function should be added to the LinkedList class.
11. **Example Code:**
12. **open Linkedlists*.sln*** and run the code then extend the code as specified in exercise 1 below

**Practice Exercises**

**Exercise 1:**

Extend ***LinkedList*** class given in the code examples accompanied with this lab by implementing the member functions specified in ***LinkedList.h*** file

**Exercise 2:**

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

1. 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*** to represent the node.

1. Declare a class ***Polynomial*** that contains: (1) Head pointer to the first node that contains the highest power, (2) degree (int to represent the degree of the polynomial), and (3) count to store the number of nodes (terms) currently in the polynomial.

1. Add the following member functions to class ***Polynomial***
   1. Constructor to initialize data members
   2. ***AddTerm(double coeff, int power)*** that adds a new term in its location in the polynomial
   3. ***PrintPoly()*** that prints the polynomial in the format: 5 x^12 + 4.3 x^5 + 22
   4. ***getCoeff(int power)*** returns the coeff of the term of the given power. It returns zero of such term doesn’t exist.
   5. ***getHead( )*** returns a pointer to the first term in the polynomial
   6. ***getNext(Term\* )*** returns a pointer to the next node to the currently pointed term
   7. ***setCoeff(double newCoeff, int power)*** updates the coefficient of the term with the given power
      1. if power = 0, the term should be deleted from the polynomial
      2. if power != 0 and there is no term with this power, a new node should be created and added for that term
   8. ***AddPoly(Polynomial P)*** adds a polynomial to the current polynomial

1. Write a program that:
   1. Declares two polynomials P1 and P2 and fills them with arbitrary terms.
   2. Prints P1 and P2.
   3. Tests functions getCoeff and setCoeff
   4. Adds the two polynomials and print the result

**Exercise 3:**

1. Use class template to generalize the linked list declared in file LinkedList.h so that a node can store any generic data type.
2. Update the member functions for the created class template.
3. Test your class for the following data: int, double, and string.