A Project report on

MAKING LINKED LISTS EASY

A Dissertation submitted in partial fulfillment of the academic requirements for the award of thedegree.

Bachelor of Technology

In

Computer Science and Engineering

Submitted by

(Student Name)	(Roll No)
B. GANESH	20H51A05B1
B.SHRAVYA	20H51A05B5
K.SAI HARSHA	20H51A05C7

Under the esteemed guidance of
Major Dr. V.A.NARAYANA
PRINCIPAL, CMRCET

Department of Computer Science and Engineering

CMR College of Engineering & Technology

(An Autonomous Institution, Approved by AICTE, Affiliated to JNTUH, NAAC 'A+')
Kandlakoya, Hyderabad 501401

2020- 2024

CMR COLLEGE OF ENGINEERING & TECHNOLOGY

KANDLAKOYA, MEDCHAL ROAD, HYDERABAD – 501401

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Mini Project-1 report entitled **"MAKING LINKED LISTS EASY"** being submitted by **B.GANESH (20H51A05B1), B.SHRAVYA (20H51A05B5), K.SAI HARSHA (20H51A05C7)** in partial fulfillment for the award of **Bachelor of Technology in Computer Science and Engineering** is a record of bonafide work carried out his/her under my guidance and supervision.

The results embodies in this project report have not been submitted to any other University or Institute for the award of any Degree.

MAJOR DR.V.A. NARAYANA
Principal of CMRCET
Dept. of CSE

DR.S. SHIVA SKANDA
Professor and HOD
Dept. of CSE

ACKNOWLEDGEMENT

With great pleasure We want to take this opportunity to express my heartfelt gratitude to all the people who helped in making this project work a grand success.

We are grateful to **Major Dr.V A Narayana**, Principal,CMRCET , Dept of Computer Science and Engineering for his valuable technical suggestions and guidance during the execution of this project work.

We would like to thank **Dr.S.Siva Skandha,** Head of the Department of Computer Science and Engineering, CMR College of Engineering and Technology, who is the major driving forces to complete my project work successfully.

We are very grateful to **Dr.Vijaya Kumar Koppula**, Dean-Academic, CMR College of Engineering and Technology, for his constant support and motivation in carrying out the project work successfully.

We are highly indebted to **Major Dr. V A Narayana,** Principal, CMR College of Engineering and Technology, for giving permission to carry out this project in a successful and fruitful way.

We would like to thank the Teaching & Non- teaching staff of Department of Computer Science and Engineering for their co-operation

Finally I express my sincere thanks to **Mr. Ch. Gopal Reddy**, Secretary, CMR Group of Institutions, for his continuous care. I sincerely acknowledge and thank all those who gave support directly and indirectly in completion of this project work.

B.GANESH-20H51A05B1
B.SHRAVYA-20H51A05B5
K.SAI HARSHA-20H51A05C7

DECLARATION

We hereby declare that results embodied in this Report of Project on **"MAKING LINKED LISTS EASY"** are from work carried out by using partial fulfillment of the requirements for the award of B. Tech degree. We have not submitted this report to any other university/institute for the award of any other degree.

NAME	ROLL NO	SIGNATURE
B.GANESH	20H51A05B1	
B.SHRAVYA	20H51A05B5	
K.SAI HARSHA	20H51A05C7	

TABLE OF CONTENTS

ABSTRACT

The idea of our project is to increase the reusability and ease of insertion, deletion etc in a Linked List. There isn't an existing solution that addresses the issue of increasing the reusability of code. In our project we will be including the following functions that can help in performing specific operations on linked lists.

- Insert
- Delete
- Detect cycles and remove cycles
- Check whether two linked lists are connected or not
- Reversing
- Sorting
- Checking a key in a linked list

Operations on linked lists are done by calling the functions enclosed in a Custom Header file which we created to increase the reusability of code

CHAPTER 1

INTRODUCTION

- Linked list is a widely used data structure which is mainly used in Image Viewer, Previous and next page in a web browser, music player etc.
- There are majorly two types of linked lists:
  1.Singly Linked lists
  2.Doubly Linked lists
- Linked lists are used in many competitive programming problems and some web applications etc .In such cases we cannot repeatedly write the same code to perform necessary operations on the linked lists.
- Hence,in order to overcome this issues we came up with an idea to include some frequently used operations in the form of functions.

OBJECTIVE:

- The objective of this project is to increase the reusability of code by using functions for linked lists. So that they can be used since they are not present in C/C++ library by default. Hence we tried to find existing systems and tried to find out a efficient solution to solve this issue.
- To achieve this we have gone through online forums and and other sources to find out solution that increases the reusability of the code and also decrease the code redundancy therefore. This can be helpful to the applications that use linked lists in the domain they are working in.
- We found some solutions and came up with an idea to incorporate the concept of User Defined Header Files in solving this issue

CHAPTER 2

EXISTING SOLUTIONS

There are no fully working or well efficient existing solutions for the problem statement which we have chosen. But, one of the Existing Systems is given below: -

a. Using Functions: -

We generally use functions to increase the reusability of our code and to decrease the effort. So hence we can make use of functions concept to perform operations on Singly Linked Lists such as Insertion, deletion etc.

Disadvantages of Existing System: -

There is no Header File which can be used to perform basic operations on the Singly Linked Lists. Functions can be used at a local scope/global scope. But, the functions in one file cannot be accessed inside a different C/C++ File.

CHAPTER 3

PROPOSED SYSTEM

3.1 PROPOSED METHODS:

1. void beginInsert(type data);
2. void lastInsert(type data);
3. void locInsert(type data, int loc);
4. void beginDelete();
5. void lastDelete();
6. void locDelete(int loc);
7. void makeCycle(int pos);
8. bool isCycle();
9. void removeCycle();
10. void reverse();
11. bool isKey(type key);
12. bool isSorted();
13. void intersect(dsa::Node<type> head1, int pos);
14. bool isConnected(dsa::Node<type> head1);
15. void sort();

3.2 DESCRIPTION:

- There are many cases where we use Linked Lists and by linked lists we mean Singly Linked Lists very often. And there are many situations in which we need to perform operations on the linked lists such as Insertion, Deletion, Reversing, Sorting etc.
- In order to perform such operations on the linked lists. But in real world scenario we might need to use the operations quite often, which means that the code ,must be written according to our needs and this leads to unnecessary redundancy of the same code.
- Hence to overcome this problem we're incorporating the concept of the Header Files using which we can reuse the code just by including the header file and we can access the functions defined in it.

3.3 ADVANTAGES OF PROPOSED SYSTEM

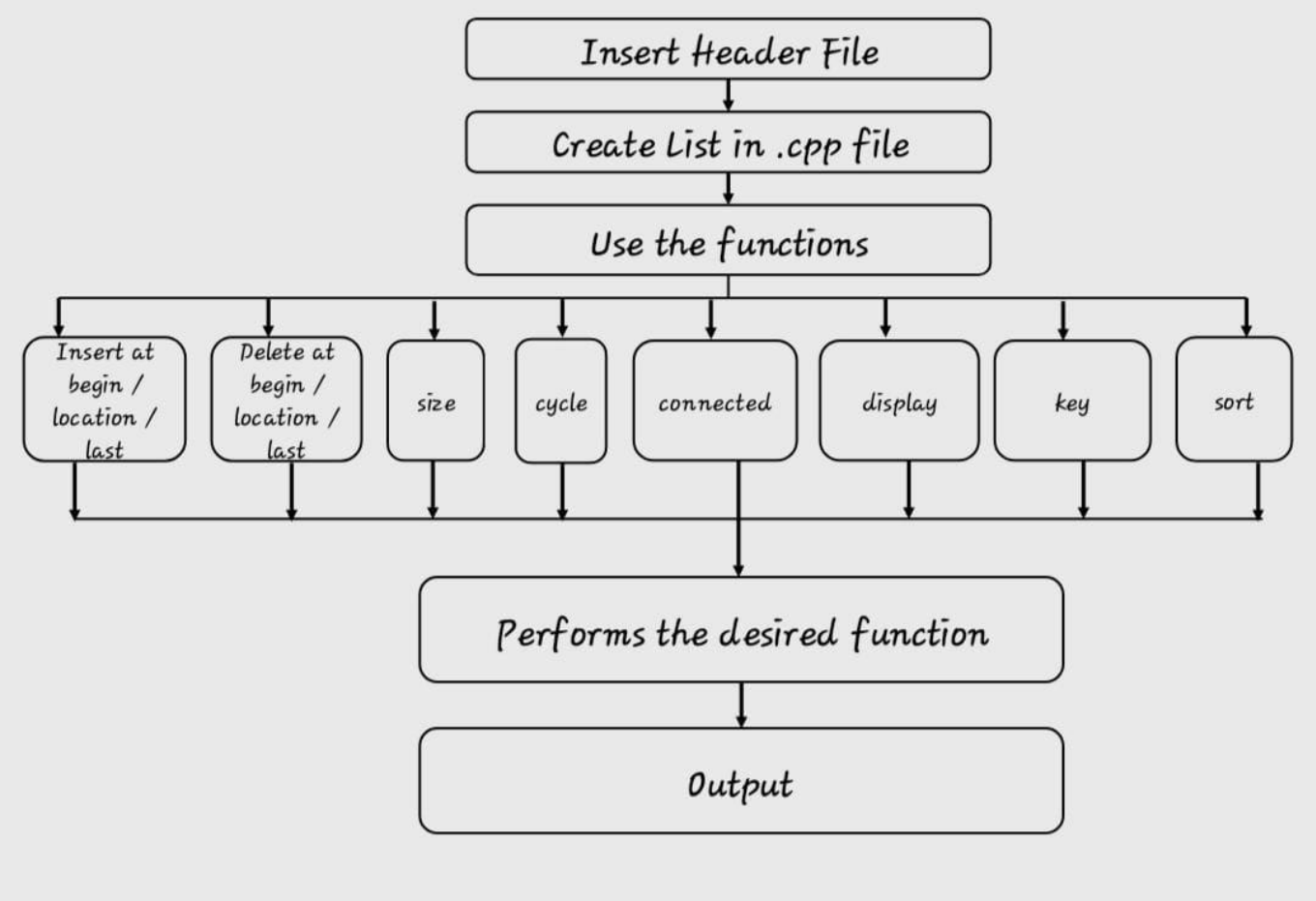
- The accuracy will be more and the time complexity will be less due to the inbuilt functions in the header file.

- Our project provides a basis to increase the reusability of code and also the help in removing the redundant code.

3.4 SYSTEM REQUIREMENTS:

OPERATING SYSTEM	:	ANY OPERATING SYSTEM
PROGRAMMING LANGUAGE	:	C++
COMPILER	:	GCC

3.5 PROPOSED SYSTEM ARCHITECTURE:



CHAPTER 4

Source code

```
// HEADERFILE
#include <bits/stdc++.h>
using namespace std;

//DECLARATIONS ---------------------------------------------------------------------
namespace dsa
{
   template <typename type> class Node
   {
   public:
      // PROPERTIES
      type data;
      Node<type> *next;

      // CONSTRUCTOR
      Node() : data(NULL), next(NULL){};
      Node(type x) : data(x), next(NULL){};

      // CLASS FUNCTIONS
      void setData(type data);
      type getData();
      void setNext(Node<type> *node);
      Node<type> getNext();
   };

   template <typename type> class LinkedList
   {
   public:
      // PROPERTIES
      dsa::Node<type> *head;
      dsa::Node<type> *tail;
      int size;

      // CONSTRUCTORS
      LinkedList() : head(NULL), tail(NULL), size(0){};

      // FUNCTIONS
      void beginInsert(type data);
      void lastInsert(type data);
      void locInsert(type data, int loc);
      void beginDelete();
      void lastDelete();
      void locDelete(int loc);
      void display();
      void makeCycle(int pos);
      bool isCycle();
```

```
        void removeCycle();
        int Size();
        void reverse();
        bool isKey(type key);

        bool isSorted();
        Node<type> getHead();
        void intersect(dsa::Node<type> head1, int pos);
        bool isConnected(dsa::Node<type> head1);
        void sort();
    };
};

// IMPLEMENATIONS ---------------------------------------------------------------

// CLASS FUNCTIONS
template <typename type>
void dsa::Node<type>::setData(type data)
{
    this->data = data;
}

template <typename type>

       type dsa::Node<type>::getData()

{

    return this->data;

}

template <typename type>

void dsa::Node<type>::setNext(Node<type> *node)

{

    this->next = node;

}



template <typename type>
```

```
dsa::Node<type> dsa::Node<type>::getNext()
{
    return this->next;
}
// Insert at Beginning
template <typename type>
void dsa::LinkedList<type>::beginInsert(type data)
{
    // MAKING A NEW NODE FROM DATA
    dsa::Node<type> *newNode = new dsa::Node<type>(data);
    // LIST IS EMPTY
    if (this->size == 0)
    {
        this->head = newNode;
        this->tail = newNode;
        this->size++;
        return;
    }
    else{
        // LIST IS NOT EMPTY
        newNode->next = this->head;
```

```
        this->head = newNode;
        this->size++;
        return;
    }
}
// Insert at End
template <typename type>
void dsa::LinkedList<type>::lastInsert(type data)
{
    // MAKING A NEW NODE FROM DATA
    dsa::Node<type> *newNode = new dsa::Node<type>(data);
    // // LIST IS EMPTY
    if (this->size == 0)
    {
        this->head = newNode;
        this->tail = newNode;
        this->size++;
        return;
    }
    else{
        this->tail->next = newNode;
```

```
        this->tail = this->tail->next;
        this->size++;
    }
}
// Insert at location
template <typename type>
void dsa::LinkedList<type>::locInsert(type data, int loc)
{
    dsa::Node<type> *newNode = new dsa::Node<type>(data);
    dsa::Node<type> *temp = this->head;
    if(this->size == 0){
        this->head = newNode;
        this->tail = newNode;
        // this->size++;
    }
    else if(loc>this->size){
        this->tail->next = newNode;
        this->tail=this->tail->next;
        // this->size++;
    }
    else{
```

```
        if(loc==1){
            newNode->next = temp;
            temp=newNode;
        }
        else{
            for(int i=1;i<loc-1;i++){
                temp=temp->next;
            }
            newNode->next=temp->next;
            temp->next=newNode;
        }
    }
    this->size++;
}
// Begin delete
template <typename type>
void dsa::LinkedList<type>::beginDelete()
{
    // Node *temp = head;
    dsa::Node<type> *temp = this->head;
    if (this->head != NULL)
```

```
    {
        this->head = this->head->next;
        this->size--;
    }
    else
    {
        cout << "List is empty\n";
    }
    free(temp);
}
template <typename type>
void dsa::LinkedList<type>::lastDelete()
{
    dsa::Node<type> *temp = this->head;
    dsa::Node<type> *temp1 = this->head;
    // Node *temp = head;
    // Node *temp1 = head;
    if (temp == NULL)
    {
   cout << "List is empty\n";
        this->size=0;
```

```
    return;
}
else if (this->head->next == NULL)
{
    temp=this->head;
    this->head = NULL;
    this->tail = NULL;
    this->size--;
    free(temp);
    return;
}
else
{
    while (temp->next != NULL)
    {
        temp1 = temp;
        temp = temp->next;
    }
    temp1->next = NULL;
    this->tail = temp1;
    this->size--;
```

```
        free(temp);
        return;
    }
}
template <typename type>
void dsa::LinkedList<type>::locDelete(int loc)
{
    // Node *temp = head;
    dsa::Node<type> *temp = this->head;
    dsa::Node<type> *temp1 = this->head;
        if(loc!=0){
        if(this->size==0){
            cout<<"List is Empyt\n";
        }
        else if(loc>this->size){
            if(temp->next==NULL){
                this->head=NULL;
                this->tail==NULL;
                this->size=0;
                free(temp);
            }
```

```
    else{
        while(temp->next!=NULL){
            temp1=temp;
            temp=temp->next;
        }
        temp1->next=NULL;
        this->tail=temp1;
        this->size--;
        free(temp);
    }
}
else{
    if(loc==1){
        this->head=this->head->next;
    }
    else{
        for(int i=1;i<loc;i++){
            temp1=temp;
            temp=temp->next;
        }
        temp1->next=temp->next;
```

```
            free(temp);
        }
        this->size--;
    }
  }
}
template <typename type>
void dsa::LinkedList<type>::display()
{
  dsa::Node<type> *traverseNode = this->head;
  while (traverseNode != NULL)
  {
    cout << traverseNode->data << " -> ";
    traverseNode = traverseNode->next;
  }
  cout<<"NULL\n";
}
// Make Cycle
template <typename type>
void dsa::LinkedList<type>::makeCycle(int pos)
{
```

```
    dsa::Node<type> *temp = this->head;
    dsa::Node<type> *start;
    int count = 1;
    while (temp->next != NULL)
    {
        if (count == pos)
        {
            start = temp;
            // break;
        }
        temp = temp->next;
        count++;
    }
    temp->next = start;
}
template <typename type>
bool dsa::LinkedList<type>::isCycle()
{
    dsa::Node<type> *tortoise = this->head;
    dsa::Node<type> *rabbit = this->head;
    while (rabbit != NULL && rabbit->next != NULL)
```

```
    {
        tortoise = tortoise->next;
        rabbit = rabbit->next->next;
        if (tortoise == rabbit)
        {
            return true;
        }
    }
    return false;
}
template <typename type>
bool dsa::LinkedList<type>::isSorted()
{
    bool flag=1;
    dsa::Node<type> *temp = this->head;
    if(temp==NULL || temp->next==NULL){
        flag=1;
    }
    else{
        if(temp->next!=NULL){
            if(temp->data < temp->next->data){
```

```
            while(temp->next!=NULL){
                if(temp->data > temp->next->data){
                    flag=0;
                    break;
                }
                temp=temp->next;
            }
        }
        else{
            while(temp->next!=NULL){
                if(temp->data < temp->next->data){
                    flag=0;
                    break;
}
                temp=temp->next;
            }
        }
      }
   }
   return flag;
}
```

```
template <typename type>
void dsa::LinkedList<type>::removeCycle()
{
    dsa::Node<type> *tortoise = this->head;
    dsa::Node<type> *rabbit = this->head;
    if (head != NULL && head->next != NULL && head->next->next != NULL)
    {
        do{
            tortoise = tortoise->next;
            rabbit = rabbit->next->next;
        } while (rabbit != tortoise);
        rabbit = head;
        while (tortoise->next != rabbit->next)
        {
            tortoise = tortoise->next;
            rabbit = rabbit->next;
        }
        tortoise->next = NULL;
    }
}
template <typename type>
```

```
int dsa::LinkedList<type>::Size()
{
    return this->size;
}
template <typename type>
void dsa::LinkedList<type>::intersect(dsa::Node<type> head1, int pos)
{
    int count=0;
    dsa::Node<type> *temp1 = this->head;
    dsa::Node<type> *temp2 = &head1;
    // size+=l-pos+1;
    pos--;
    while (pos--)
    {
        if(temp1!=NULL)
            temp1 = temp1->next;
    }
    while (temp2->next != NULL)
    {
        count++;
        temp2 = temp2->next;
```

```
    }
    temp2->next = temp1;
}
template <typename type>
void dsa::LinkedList<type>::reverse()
{
    dsa::Node<type> *prev = this->head;
    dsa::Node<type> *current = this->head->next;
    if (this->head == NULL || this->head->next == NULL)
    {
        return;
    }
    while (current != NULL)
    {
        dsa::Node<type> *nextNode = current->next;
        current->next = prev;
        prev = current;
        current = nextNode;
    }
    this->head->next = NULL;
    this->head = prev;
```

```
}
template <typename type>
bool dsa::LinkedList<type>::isKey(type key)
{
    dsa::Node<type> *temp = this->head;
    while (temp != NULL)
    {
        if (temp->data == key)
        {
            return true;
        }
        temp = temp->next;
    }
    return false;
}
template <typename type>
dsa::Node<type> dsa::LinkedList<type>::getHead()
{
    return *this->head;
}
template <typename type>
```

```
bool dsa::LinkedList<type>::isConnected(dsa::Node<type> head1)
{
    dsa::Node<type> *temp1 = this->head;
    dsa::Node<type> *temp2 = &head1;
    int l1=0,l2=0;
    while(temp1!=NULL){
        l1++;
        temp1=temp1->next;
    }
    while(temp2!=NULL){
        l2++;
        temp2=temp2->next;
    }
    temp1=this->head;
    temp2=&head1;
    int len;
    if (l1 < l2)
    {
        len = l2 - l1;
        while (len--)
        {
```

```
            temp2 = temp2->next;
        }
    }
    else
    {
        len = l1 - l2;
        while (len>0)
        {
            temp1 = temp1->next;
            len--;
        }
    }
while (temp1 != NULL && temp2 != NULL)
    {
        if (temp1 == temp2)
        {
            return true;
        }
        temp1 = temp1->next;
        temp2 = temp2->next;
    }return false;
```

```
}

// Sort the linked list

template <typename type>

void dsa::LinkedList<type>::sort(){

  dsa::Node<type> *current = this->head, *index = NULL;

  int temp;

  if (this->head == NULL) {

    return;

  } else {

    while (current != NULL) {

      // index points to the node next to current

      index = current->next;

           while (index != NULL) {

        if (current->data > index->data) {

          temp = current->data;

          current->data = index->data;

          index->data = temp;

            }

            index = index->next;

          }

          current = current->next; }  } }
```

CHAPTER 5

RESULTS AND DISCUSSIONS

5.1 WORKING PROTOTYPE IMAGE

index.cpp > main()

```cpp
#include<bits/stdc++.h>
#include "./linkedList.hpp"
using namespace std;

int main(){
    dsa::LinkedList<int> List1;
    dsa::LinkedList<int> List2;
    List1.beginInsert(1);
    List1.beginInsert(2);
    List1.beginInsert(3);
    List1.beginInsert(4);
    List1.beginInsert(5);
    List2.beginInsert(6);
    List2.beginInsert(7);
    List1.display();
    List2.display();
    cout<<List1.Size()<<"\n";
    cout<<List2.Size();
    return 0;
}
```

PROBLEMS OUTPUT TERMINAL JUPYTER DEBUG CONSOLE

```
Microsoft Windows [Version 10.0.22621.674]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Balla Ganesh\Desktop\MiniTrial>cd "c:\Users
5 -> 4 -> 3 -> 2 -> 1 -> NULL
7 -> 6 -> NULL
5
2
c:\Users\Balla Ganesh\Desktop\MiniTrial>
```

5.2 SCREENSHOTS OF EXECUTION:

Function	Execution
index.cpp > main() 1 #include<bits/stdc++.h> 2 #include "./linkedList.hpp" 3 using namespace std; 4 5 int main(){ 6 dsa::LinkedList<int> List1; 7 List1.beginInsert(5); 8 List1.display(); 9 List1.beginInsert(10); 10 List1.display(); 11 return 0; 12 }	C:\Users\Balla Ganesh\Desktop\MiniTrial>cd 5 -> NULL 10 -> 5 -> NULL
Fig 5.1 (a) beginInsert() 5 and 10	**Fig 5.1 (b)** beginInsert() execution
index.cpp > main() 1 #include<bits/stdc++.h> 2 #include "./linkedList.hpp" 3 using namespace std; 4 5 int main(){ 6 dsa::LinkedList<int> List1; 7 List1.lastInsert(5); 8 List1.display(); 9 List1.lastInsert(10); 10 List1.display(); 11 return 0; 12 }	C:\Users\Balla Ganesh\Desktop\MiniTrial>cd 5 -> NULL 5 -> 10 -> NULL
Fig 5.2 (a) lastInsert() 5 and 10	**Fig 5.2 (b)** lastInsert( ) execution

index.cpp > main()

```cpp
#include<bits/stdc++.h>
#include "./linkedList.hpp"
using namespace std;

int main(){
    dsa::LinkedList<int> List1;
    List1.locInsert(5,3);
    List1.display();
    List1.beginInsert(7);
    List1.beginInsert(8);
    List1.locInsert(10,4);
    List1.display();
    List1.locInsert(15,3);
    List1.display();
    return 0;
}
```

Fig 5.3 (a)

locInsert() 5 at position 3

locInsert() 10 at position 4

```
C:\Users\Balla Ganesh\Desktop\MiniTrial>cd
5 -> NULL
8 -> 7 -> 5 -> 10 -> NULL
8 -> 7 -> 15 -> 5 -> 10 -> NULL
```

Fig 5.3 (b)

locInsert() Execution

index.cpp > main()

```cpp
#include<bits/stdc++.h>
#include "./linkedList.hpp"
using namespace std;

int main(){
    dsa::LinkedList<int> List1;
    List1.beginDelete();
    List1.display();
    List1.lastInsert(10);
    List1.beginInsert(5);
    List1.display();
    List1.beginDelete();
    List1.display();
    return 0;
}
```

Fig 5.4 (a)

beginDelete() 5 and 10 from Linked list

```
C:\Users\Balla Ganesh\Desktop\MiniTrial>cd
List is empty
NULL
5 -> 10 -> NULL
10 -> NULL
```

Fig 5.4 (b)

beginDelete() Execution

index.cpp > main()

```cpp
#include<bits/stdc++.h>
#include "./linkedList.hpp"
using namespace std;

int main(){
    dsa::LinkedList<int> List1;
    List1.lastDelete();
    List1.display();
    List1.lastInsert(10);
    List1.beginInsert(5);
    List1.display();
    List1.lastDelete();
    List1.display();
    return 0;
}
```

Fig 5.5 (a)

lastDelete() 5 and 10 from list

```
C:\Users\Balla Ganesh\Desktop\MiniTrial>cd
List is empty
NULL
5 -> 10 -> NULL
5 -> NULL
```

Fig 5.5 (b)
lastDelete() Execution

index.cpp > main()

```cpp
    dsa::LinkedList<int> List1;
    List1.beginInsert(7);
    List1.beginInsert(8);
    List1.locInsert(5,3);
    List1.locInsert(10,4);
    List1.locInsert(15,3);
    List1.display();
    List1.locDelete(0);
    List1.display();
    List1.locDelete(3);
    List1.display();
    List1.locDelete(6);
    List1.display();
    return 0;
}
```

Fig 5.6 (a)

locDelete() from 3rd and 6th location

```
C:\Users\Balla Ganesh\Desktop\MiniTrial>cd
8 -> 7 -> 15 -> 5 -> 10 -> NULL
8 -> 7 -> 15 -> 5 -> 10 -> NULL
8 -> 7 -> 5 -> 10 -> NULL
8 -> 7 -> 5 -> NULL
```

Fig 5.6 (b)
locDelete() Execution

```cpp
#include "./linkedList.hpp"
using namespace std;

int main(){
    dsa::LinkedList<int> List1;
    List1.beginInsert(5);
    List1.beginInsert(4);
    List1.beginInsert(3);
    List1.beginInsert(2);
    List1.beginInsert(1);
    List1.display();
    cout<<List1.isCycle()<<"\n";
    List1.makeCycle(3);
    List1.display();
    return 0;
}
```

Fig 5.7 (a)
makeCycle() at 3rd node

```
C:\Users\Balla Ganesh\Desktop\MiniTrial>cd "c:\Users\Balla
1 -> 2 -> 3 -> 4 -> 5 -> NULL
0
1 -> 2 -> 3 -> 4 -> 5 -> 3 -> 4 -> 5 -> 3 -> 4 -> 5 -> 3 ->
-> 4 -> 5 -> 3 -> 4 -> 5 -> 3 -> 4 -> 5 -> 3 -> 4 -> 5 -> 3
 3 -> 4 -> 5 -> 3 -> 4 -> 5 -> 3 -> 4 -> 5 -> 3 -> 4 -> 5 -
```

Fig 5.7 (b)
makeCycle() Execution

index.cpp > main()

```cpp
#include "./linkedList.hpp"
using namespace std;

int main(){
    dsa::LinkedList<int> List1;
    List1.beginInsert(5);
    List1.beginInsert(4);
    List1.beginInsert(3);
    List1.beginInsert(2);
    List1.beginInsert(1);
    List1.display();
    cout<<List1.isCycle()<<"\n";
    List1.makeCycle(3);
    cout<<List1.isCycle()<<"\n";
    List1.removeCycle();
    cout<<List1.isCycle()<<"\n";
    List1.display();
    return 0;
}
```

Fig 5.8 (a)
removeCycle() from the given Linked List

```
C:\Users\Balla Ganesh\Desktop\MiniTrial>cd
1 -> 2 -> 3 -> 4 -> 5 -> NULL
0
1
0
1 -> 2 -> 3 -> 4 -> 5 -> NULL
```

Fig 5.8 (b)
removeCycle() Execution

index.cpp > main()

```cpp
#include<bits/stdc++.h>
#include "./linkedList.hpp"
using namespace std;

int main(){
    dsa::LinkedList<int> List1;
    dsa::LinkedList<int> List2;
    List1.beginInsert(1);
    List1.beginInsert(2);
    List1.beginInsert(3);
    List1.beginInsert(4);
    List1.beginInsert(5);
    List2.beginInsert(6);
    List2.beginInsert(7);
    List1.display();
    List2.display();
    cout<<List1.Size()<<"\n";
    cout<<List2.Size();
    return 0;
}
```

Fig 5.9 (a)

Size() of Linked List

```
C:\Users\Balla Ganesh\Desktop\MiniTrial>cd
5 -> 4 -> 3 -> 2 -> 1 -> NULL
7 -> 6 -> NULL
5
2
```

Fig 5.9 (b)

Size() Execution

index.cpp > main()

```cpp
#include "./linkedList.hpp"
using namespace std;

int main(){
    dsa::LinkedList<int> List1;
    List1.beginInsert(5);
    List1.beginInsert(4);
    List1.beginInsert(3);
    List1.beginInsert(2);
    List1.beginInsert(1);
    List1.display();
    List1.reverse();
    List1.display();
    return 0;
}
```

Fig 5.10 (a)

Reverse() the Linked List

```
C:\Users\Balla Ganesh\Desktop\MiniTrial>cd
1 -> 2 -> 3 -> 4 -> 5 -> NULL
5 -> 4 -> 3 -> 2 -> 1 -> NULL
```

Fig 5.10 (b)

Reverse() execution

index.cpp > main()

```cpp
#include "./linkedList.hpp"
using namespace std;

int main(){
    dsa::LinkedList<int> List1;
    List1.beginInsert(5);
    List1.beginInsert(4);
    List1.beginInsert(3);
    List1.beginInsert(2);
    List1.beginInsert(1);
    List1.display();
    cout<<List1.isKey(10)<<"\n";
    cout<<List1.isKey(4);
    return 0;
}
```

Fig 5.11 (a)

isKey() to check if 10 and 4 are present in the Linked List

```
C:\Users\Balla Ganesh\Desktop\MiniTrial>cd
1 -> 2 -> 3 -> 4 -> 5 -> NULL
0
1
```

Fig 5.11 (b)
isKey() execution

index.cpp > main()

```cpp
using namespace std;

int main(){
    dsa::LinkedList<int> List1;
    List1.beginInsert(5);
    List1.beginInsert(4);
    List1.beginInsert(3);
    List1.beginInsert(2);
    List1.beginInsert(1);
    List1.display();
    cout<<List1.isSorted()<<"\n";
    List1.reverse();
    List1.display();
    cout<<List1.isSorted()<<"\n";
    return 0;
}
```

Fig 5.12 (a)

isSorted() checks whether a linked list is sorted or not. Be it Ascending or Descending

```
C:\Users\Balla Ganesh\Desktop\MiniTrial>cd
1 -> 2 -> 3 -> 4 -> 5 -> NULL
1
5 -> 4 -> 3 -> 2 -> 1 -> NULL
1
```

Fig 5.12 (b)
isSorted() execution

index.cpp > main()

```cpp
using namespace std;

int main(){
    dsa::LinkedList<int> List1;
    List1.beginInsert(2);
    List1.beginInsert(4);
    List1.beginInsert(5);
    List1.beginInsert(3);
    List1.beginInsert(1);
    List1.display();
    cout<<List1.isSorted()<<"\n";
    List1.sort();
    List1.display();
    cout<<List1.isSorted()<<"\n";
    return 0;
}
```

Fig 5.13 (a)

Sort() is used to sort the unsorted Linked List

```
C:\Users\Balla Ganesh\Desktop\MiniTrial>cd "
1 -> 3 -> 5 -> 4 -> 2 -> NULL
0
1 -> 2 -> 3 -> 4 -> 5 -> NULL
1
```

Fig 5.13 (b)

Sort() execution

index.cpp > main()

```cpp
#include<bits/stdc++.h>
#include "./linkedList.hpp"
using namespace std;

int main(){
    dsa::LinkedList<int> List1;
    dsa::LinkedList<int> List2;
    List1.beginInsert(1);
    List1.beginInsert(2);
    List1.beginInsert(3);
    List1.beginInsert(4);
    List1.beginInsert(5);
    List2.beginInsert(6);
    List2.beginInsert(7);
    List1.display();
    List2.display();
    cout<<List1.isConnected(List2.getHead())<<"\n";
    List1.intersect(List2.getHead(),3);
    List1.display();
    List2.display();
    cout<<List1.isConnected(List2.getHead())<<"\n";
    return 0;
}
```

Fig 5.14 (a)

isConnected() to determine whether two Linked Lists are connected or not

```
C:\Users\Balla Ganesh\Desktop\MiniTrial>cd "
5 -> 4 -> 3 -> 2 -> 1 -> NULL
7 -> 6 -> NULL
0
5 -> 4 -> 3 -> 2 -> 1 -> NULL
7 -> 6 -> 3 -> 2 -> 1 -> NULL
1
```

Fig 5.14 (b)

isConnected() execution

5.3 PERFORMANCE MEASURE:

Functions	Best Case	Average Case	Worst Case
beginInsert	O (1)	O (1)	O (1)
lastInsert	O (1)	O (n)	O (n)
locInsert	O (1)	O (n)	O (n)
beginDelete	O (1)	O (1)	O (1)
lastDelete	O (1)	O (n)	O (n)
locDelete	O (1)	O (n)	O (n)
display	O (1)	O (n)	O (n)
IsCycle	O (1)	O (n)	O (n)
removeCycle	O (1)	O (n)	O (n)
Size	O (1)	O (1)	O (1)
isKey	O (1)	O (n)	O (n)
isSorted	O (1)	O (n)	O (n)
isConnected	O (1)	O (n)	O (n)
sort	O (1)	O (n)	O (n)

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 CONCLUSION:

- Our project has reached the goals which we expected it to reach, it decreased the redundancy of code by using the concepts of functions.
- The project has reached its goal of increasing the reusability of code to perform some most frequently performed operations on the Singly Linked Lists. We generally make use of the functions which are declared/defined in the body of code.
- But the function's scope is only local to that particular file. And it cannot be accessed among other files. So, to address this issue we can make use of Header Files.
- By making use of Header Files we can globally access the function that are defined in the header file. This therefore increases the code reusability and increase the efficiency by making use of the predefined functions in the applications/problems that are solved by using the Linked Lists concept.
- We have written the functions in a User Defined Header File. The functions we wrote inside the header file are optimal and efficient. We used the concept of OOPS by making use of Classes and Objects.

6.2 FUTURE WORK:

The future work is as follows: -

- We primarily focused on Singly Linked Lists in this project as it is frequently used in competitive coding and other aspects. But, we can increase the scope of this project in the future by implementing this concept of custom header files to other types of linked lists i.e. Doubly Linked List and Circular Linked Lists.

- We included most of the frequently used functions in the Custom Header File. There are some other functions we can add in the future. For example, there is a possibility to add a merge and sort function to the existing model which we have worked on.

CHAPTER 7

REFERENCES

REFERENCES:

1) https://learn.microsoft.com/en-us/cpp/cpp/header-files-cpp?view=msvc-170
2) https://gcc.gnu.org/onlinedocs/gcc-12.2.0/libstdc++/manual/