# ***LAB REPORT***

***Group members:***

- Laiba Batool
- Mariyam Muzammil
- Hira Arif

***Course***: DSA

***Lab Number:*** 02

***Lab Title:*** Doubly Linked List

***Date:*** 02/10/24

***Objective:***

- Practice Abstract Data Types.
- Understand the Basic Doubly Linked List.
- Implement Doubly Linked List ADT named as DList.

***Conclusion:***

The main function is working well. All functions are good to go. We learnt a lot about inheritance and its types and abstract classes.

***HEADER FILE:***

## DLIST.h

```
#ifndef DLIST_H
#define DLIST_H


// Node Structure
struct Node{
public:
     int data;
     Node* next;
     Node* prev;
};
// Defining Node Pointer Type
typedef Node* Nodeptr;


// List Class Declaration
class DList
{
   public:
      // constructor
      DList();
      // destructor
      virtual ~DList();
      // copy constructor
      DList(const DList& other);


      // boolean function
      bool empty() const;
```

```
    // access head element
    int headElement() const;
    // access tail element
    int tailElement() const;
    // access element at specific index
    int getAt(int idx);
    // add to the head
    void addHead(int newdata);
    // delete the head
    void delHead();
    // add to the head
    void addTail(int newdata);
    // delete the head
    void delTail();
    // add to the head
    void addAt(int idx, int newdata);
    // delete the head
    void delAt(int idx);
    // Clear the list
    void Clear();

    // utility function to get length of list
    int length() const;
    // display the list
    void print() const;

private:
    void createDummyHead();
    //Go to specific index and return poiter to node at that position
```

```
        Nodeptr goToIndex(int idx);
        //head pointer, pointing to dummy noce actually
        Nodeptr head;
    };

    #endif // DLIST_H
```

## ***.CPP FILES :***

### Class DLIST

```
#include "DList.h"
#include <iostream>
using namespace std;
DList::DList()
{
   createDummyHead();
}

DList::~DList()
{
   //Clear The list
   Clear();
   //Delete Dummy Node
   delete head;
}

// copy constructor
DList::DList(const DList& other)
```

```
{
   //Initialize current list
   createDummyHead();
   //Check if other list is empty (if empty do nothing)
   if(other.empty())
      return;
   //Iterate through all the nodes of other list
   //and add all data elements to current list
   Nodeptr other_curr = other.head->next;
   Nodeptr other_head = other.head;

   while(other_curr != other_head)
   {
      addTail(other_curr->data);
      other_curr = other_curr->next;
   }
}

// boolean function
bool DList::empty() const
{
   return head->next == head;
}

// access head element
```

```
int DList::headElement() const
{
   if(!empty())
      return head->next->data; //since list is empty,so dummy->next->data
   cerr<<"List is Empty";
}

// access tail element
int DList::tailElement() const
{
   if(!empty())
      return head->prev->data;
   cerr<<"List is Empty";
}

// access element at specific index
int DList::getAt(int idx)
{
   Nodeptr pos = goToIndex(idx);
   if(pos != NULL)
   {
      return pos->data;
   }
}
```

```
// add to the head
void DList::addHead(int newdata)
{
   //Location to insert Head Node,
   //Between DummyHead and Actual First Node
   Nodeptr curr = head->next;

   //Create New Node
   Nodeptr newnode = new Node;

   //Populate the new created node
   newnode->data = newdata;

   //Link the new created node
   newnode->next = curr;
   newnode->prev = head;
   head->next->prev = newnode;
   head->next = newnode;
}

// delete the head
void DList::delHead()
{
   //Check if list is empty ? Do nothing
   if(empty())
```

```
        return;
    //Location to delete Head Node,
    //Just after DummyHead
    Nodeptr curr = head->next;
    //Update references
    head->next = curr->next;
    curr->next->prev = head;

    //Free Node Memory on Heap
    delete curr;

}

// add to the tail
void DList::addTail(int newdata)
{
    //Location to insert Head Node,
    //Between DummyHead and Actual Last Node
    Nodeptr curr = head;
    //Create New Node
    Nodeptr newnode = new Node;
    //Populate the new created node
    newnode->data = newdata;

    //Link the new created node
```

```
    newnode->next = head;
    newnode->prev = head->prev;
    head->prev->next = newnode;
    head->prev = newnode;
}

// delete the tail
void DList::delTail()
{
    //Check if list is empty ? Do nothing
    if(empty())
        return;
    //Location to delete Tail Node,
    //Just Before DummyHead
    Nodeptr curr = head->prev;
    //Update references
    head->prev = curr->prev;
    curr->prev->next = head;

    //Free Node Memory on Heap
    delete curr;

}

// add to the specific indx
```

```
void DList::addAt(int idx, int newdata)
{
   //Get node at current position
   Nodeptr curr = goToIndex(idx);
   if(curr == NULL)    //Index exceed size
      return;

   //Create New Node
   Nodeptr newnode = new Node;

   //Populate the new created node
   newnode->data = newdata;

   //Link the new created node
   newnode->next = curr;
   newnode->prev = curr->prev;
   curr->prev->next = newnode;
   curr->prev = newnode;
}

// delete at index
void DList::delAt(int idx)
{
   //Get node at current position
   Nodeptr curr = goToIndex(idx);
```

```
    if(curr == NULL)    //Index exceed size
        return;

    //Update references
    curr->prev->next = curr->next;
    curr->next->prev = curr->prev;

    //Free Node Memory on Heap
    delete curr;

}

// utility function to get length of list
int DList::length() const
{
    int count = 0;
    Nodeptr curr = head->next;
    while(curr!=head)
    {
        count++;
        curr = curr->next;
    }
    return count;
}
```

```
// display the list
void DList::print() const
{
   //Set the starting point of list
   Nodeptr curr = head->next;
   cout << "[";
   //Iterate and display list.
   //Make sure to handle comma ',' seperation is correct
   if(!empty()){
      cout << curr->data;
      curr = curr->next;
   }
   while(curr != head){
      cout << ", " << curr->data;
      curr = curr -> next;
   }
   cout << "]" << endl;
}

// Add dummy Head and populate
void DList::createDummyHead()
{
   head = new Node;
   head->next = head;
   head->prev = head;
```

```
}

// Clear The List
void DList::Clear()
{
   while(!empty())
      delHead();
}

//Go to specific index and return poiter to node at that position
//Indexing is zero based
Nodeptr DList::goToIndex(int idx)
{
   if(idx > length())
   {
      cerr<<"Error! Given index exceed the size of list";
      return NULL;
   }

   //Iterate uptill given index
   Nodeptr curr = head->next;
   for(int i=0; i<idx; i++)
      curr = curr->next;

   return curr;
```

}

## ***DIRECTORY STRUCTURE :***