KATHMANDU UNIVERSITY

Department of Computer Science and Engineering
Dulikhel, Kavre

LAB EXERCISE 1
[Code No: COMP 206]

Submitted By:

Rajaram Karki (Roll No: 23)

Submitted To:

Department of Computer Science and Engineering
Rajani Chulyadyo (Ma'am)

Linked List

Linkedlist.h

```
#pragma once

class Node{
    public:
        int data;
        Node * next;

        Node() : next(nullptr) {}
        Node(int data, Node *next): data(data), next(next){}
};


class linkedlist{
    public:
        linkedlist();
        bool isEmpty();
        void addtohead(int data);
        void addtotail(int data);
        void traverse(char separator = ' ');
        bool search(int data);
        int removefromhead();
        int removefromtail();
        void remove(int data);
        void add(int data, Node *predescessor);
        bool retrieve(int data, Node *outputNodePointer);
        Node *getheadptr(){return headptr;}
        Node *gettailptr(){return tailptr;}
    private:
        Node * headptr;
        Node * tailptr;
};
```

Linkedlist.cpp

```
#include"linkedlist.h"
#include<iostream>
using namespace std;


linkedlist::linkedlist() {
    headptr = nullptr;
    tailptr = nullptr;
}


void linkedlist::traverse(char separator)
{
    if(isEmpty())
    {
        cout<<"The list is empty."<<endl;
    }
    else{
        Node *temp = headptr;

        while(temp != nullptr)
        {
            cout<< temp -> data << separator;
            temp = temp->next;
        }
        cout<<endl;
    }
}


bool linkedlist::isEmpty() {
    return headptr == nullptr && tailptr == nullptr;
}
```

```
void linkedlist::addtohead(int data) {
    Node *newNode = new Node();
    newNode->data = data;
    newNode->next = headptr;
    headptr = newNode;

    if(tailptr == nullptr){
        tailptr = headptr;
    }
}


void linkedlist::addtotail(int data)
{
    Node *newNode = new Node();
    newNode->data = data;
    newNode->next = NULL;

    if(headptr == nullptr)
        headptr = tailptr = newNode;

    else
    {
        tailptr -> next = newNode;
        tailptr = tailptr -> next;
    }
}


void linkedlist::add(int data, Node *predescessor)
{
    Node *newNode = new Node(data, predescessor->next);
    // newNode->data = data;
    // newNode->next = predescessor->next;
    predescessor->next = newNode;
}
```

```
int linkedlist::removefromhead()
{
    if(!isEmpty())
    {
        Node *nodetodelete = headptr;
        headptr = nodetodelete -> next;
        int info = nodetodelete->data;
        delete nodetodelete;

        if(headptr == nullptr)
        {
            tailptr == nullptr;
        }


        return info;
    }
    else{
        return isEmpty();
    }
}


int linkedlist::removefromtail()
{
    if(!isEmpty())
    {
        Node *nodetodelete = tailptr;

        if(headptr == tailptr)
        {
            headptr = tailptr = nullptr;
        }
        else{
            Node *pred = headptr;
            while(pred -> next != tailptr)
            {
                pred = pred->next;
            }
            tailptr = pred;
            pred->next = nullptr;
        }
        int info = nodetodelete->data;
        delete nodetodelete;

        return info;
    }
    else{
        return isEmpty();
    }
}


void linkedlist::remove(int data)
{
    if(!isEmpty())
    {
        if(headptr->data == data)
            removefromhead();
        else
        {
            Node *temp = headptr->next;
            Node *prev = headptr;

            while(temp!=NULL)
            {
                if(temp->data == data)
                    break;

                else
                {
                    prev = prev->next;
                    temp = temp->next;
                }
            }
            if(temp!=NULL)
            {
                prev->next = temp->next;
                delete temp;
                if(prev->next==NULL)
                {
                    tailptr = prev;
                }
            }
        }
    }
}


bool linkedlist::retrieve(int data, Node *outputNodePointer)
{
    Node * temp = headptr;
    while(temp!=nullptr && temp->data != data)
    {
        temp = temp->next;
    }

    if(temp==nullptr)
    {
        cout<<data<<"doesn't exist in the list"<<endl;
        return false;

    }

    else
    {
        outputNodePointer = temp;
        cout<<data<<" found"<<endl;
        return true;
    }
}


bool linkedlist::search(int data)
{
    Node *temp = headptr;
    while (temp != nullptr && temp->data != data)
    {
        temp = temp->next;
    }
    if (temp == nullptr)
    {
        cout << data <<" element is not in the list" << endl;
        return false;
    }
    else
    {
        cout << data << " is in the list" << endl;
        return true;
    }
}
```

Queue.h

```
#include<iostream>
#include "linkedlist.h"
using namespace std;

class Queue{
    linkedlist list;

    public:
        bool enqueue(const int &data);
        bool dequeue(int &data);
        bool front(int &data);
        bool rear(int &data);
};


bool Queue::enqueue(const int &data)
{
    list.addtotail(data);
    cout<< data << " added to queue" << endl;
    return true;
}

bool Queue::dequeue(int &data)
{
    if(!list.isEmpty())
    {
        data = list.removefromhead();
        cout<< data << " removed from queue" << endl;
        return true;
    }
     else
     {
         cout<< "Queue is empty"<<endl;
         return false;
     }
}

bool Queue::front(int &data)
{
    if(!list.isEmpty())
    {
        data = list.getheadptr()->data;
        cout<< data << " is in the front" << endl;
        return true;
    }
    else
    {
        cout<< "Queue is empty"<<endl;
        return false;
    }
}

bool Queue::rear(int &data)
{
     if(!list.isEmpty())
    {
        data = list.gettailptr()->data;
        cout<< data << " is in the rear" << endl;
        return true;
    }
    else
    {
        cout<< "Queue is empty"<<endl;
        return false;
    }
}
```

Stack.h

```
#include<iostream>
#include "linkedlist.h"
using namespace std;

class Stack{

    linkedlist list;

    public:
        bool push(const int &data);
        bool pop(int &data);
        bool top(int & data);
};

bool Stack::push(const int &data)
{
     list.addtohead(data);
     cout << data << " pushed" << endl;
     return true;
}

bool Stack::pop(int &data)
{
    if(!list.isEmpty())
    {
        data = list.removefromhead();
        cout<< data << " popped" <<endl;
        return true;
    }
    else
        cout<<"The list is empty!"<<endl;
        return false;
}
```

```
bool Stack::top(int &data)
{
    if(!list.isEmpty())
    {
        data = list.getheadptr()->data;
        cout<<"Top is "<< data <<endl;
        return true;
    }
    else
        cout << "The stack is empty."<< endl;
        return false;
}
```

Main.cpp

```
#include"linkedlist.h"
#include"Stack.h"
#include"queue.h"
#include<iostream>

using namespace std;


int main()
{
    cout<<"For linkedlist:"<<endl;
    linkedlist list;
    Node *n = nullptr;
    int i;

    list.traverse();
    list.addtohead(5);
    list.addtohead(10);
    list.addtohead(15);
    list.addtotail(0);
    list.traverse();
    list.removefromhead();
    list.traverse();
    list.removefromtail();
    list.traverse();
    list.addtotail(50);
    list.traverse();
    list.remove(50);
    list.traverse();
    list.search(5);
    list.retrieve(10,n);
    list.traverse();
    cout<<endl;
    cout<<"For Stack:"<<endl;
    int j;
    Stack s;

    s.top(j);
    s.push(10);
    s.top(j);
    s.push(15);
    s.top(j);
    s.pop(j);
    s.top(j);

    cout<<endl;
    cout<<"For Queue:"<<endl;
    int k;
    Queue a;

    a.front(k);
    a.enqueue(15);
    a.enqueue(10);
    a.enqueue(5);
    a.enqueue(2);
    a.front(k);
    a.rear(k);
    a.dequeue(k);

}
```

Output

For linkedlist:

```
The list is empty.
15 10 5 0
10 5 0
10 5
10 5 50
10 5
5 is in the list
10 found
10 5
```

For Stack:

```
The stack is empty.
10 pushed
Top is 10
15 pushed
Top is 15
15 popped
Top is 10
```

For Queue:

```
Queue is empty
15 added to queue
10 added to queue
5 added to queue
2 added to queue
15 is in the front
2 is in the rear
15 removed from queue
```