Data Structuers and algorithms (CS09203)

Lab Report

Name:	Irtaza Kashir Raja
Registration #:	SEU-F16-127
Lab Report #:	07
Dated:	21-05-2018
Submitted To:	Mr. Usman Ahmed

The University of Lahore, Islamabad Campus
Department of Computer Science & Information Technology

Experiment # 6
DOUBLE LINK LIST

Objective
The objective of this session is to understand the various operations on graphs in C++. **Software Tool**
1. Code Blocks with GCC compiler.

1 Theory

In computer science, a doubly linked list is a linked data structure that consists of a set of sequentially linked records called nodes. Each node contains two fields, called links, that are references to the previous and to the next node in the sequence of nodes. The beginning and ending nodes' previous and next links, respectively, point to some kind of terminator, typically a sentinel node or null, to facilitate traversal of the list. If there is only one sentinel node, then the list is circularly linked via the sentinel node. It can be conceptualized as two singly linked lists formed from the same data items, but in opposite sequential orders.

2 Task

2.1 Task 1

Write a C++ code using functions for the following operations. 1. Create the doubly link list. 2. Traversing a Linked list.

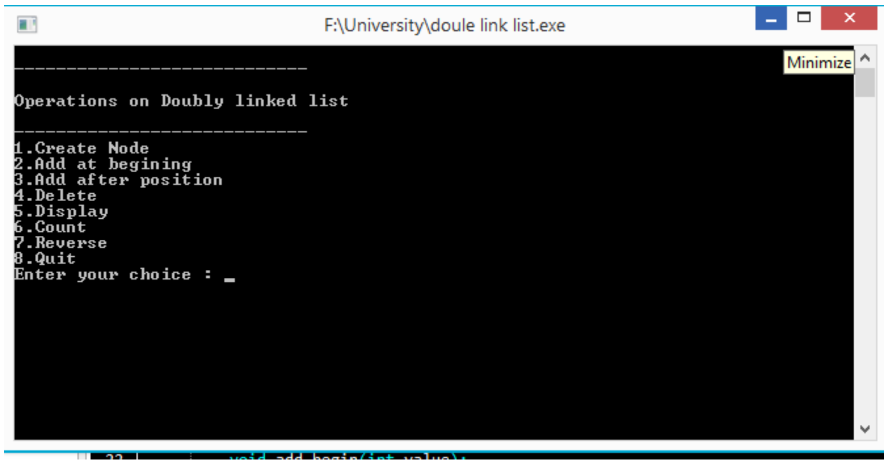
2.2 Procedure: Task 1

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
/*
```

```
 * Node Declaration
 */
using namespace std;
struct node
{
    int info;
    struct node *next;
    struct node *prev;
}*start;

/*
 Class Declaration
 */
class double_llist
{
    public:
        void create_list(int value);
        void add_begin(int value);
        void add_after(int value, int position);
        void delete_element(int value);
        void search_element(int value);
        void display_dlist();
        void count();
        void reverse();
        double_llist()
        {
            start = NULL;
        }
};

/*
 * Main: Conatins Menu
 */
int main()
{
    int choice, element, position;
    double_llist dl;
    while (1)
    {
        cout<<endl<<"-----------------------------"<<endl;
```

```
cout<<endl<<"Operations on Doubly linked list"<<endl;
cout<<endl<<"————————————————————————————"<<endl;
cout<<"1.Create Node"<<endl;
cout<<"2.Add at begining"<<endl;
cout<<"3.Add after position"<<endl;
cout<<"4.Delete"<<endl;
cout<<"5.Display"<<endl;
cout<<"6.Count"<<endl;
cout<<"7.Reverse"<<endl;
cout<<"8.Quit"<<endl;
cout<<"Enter your choice : ";
cin>>choice;
switch ( choice )
{
case 1:
    cout<<"Enter the element: ";
    cin>>element;
    dl.create_list(element);
    cout<<endl;
    break;
case 2:
    cout<<"Enter the element: ";
    cin>>element;
    dl.add_begin(element);
    cout<<endl;
    break;
case 3:
    cout<<"Enter the element: ";
    cin>>element;
    cout<<"Insert Element after postion: ";
    cin>>position;
    dl.add_after(element, position);
    cout<<endl;
    break;
case 4:
    if (start == NULL)
    {
        cout<<"List empty,nothing to delete"<<endl;
        break;
    }
```

```
                cout<<"Enter the element for deletion: ";
                cin>>element;
                dl.delete_element(element);
                cout<<endl;
                break;
            case 5:
                dl.display_dlist();
                cout<<endl;
                break;
            case 6:
                dl.count();
                break;
            case 7:
                if (start == NULL)
                {
                    cout<<"List empty,nothing to reverse"<<endl;
                    break;
                }
                dl.reverse();
                cout<<endl;
                break;
            case 8:
                exit(1);
            default:
                cout<<"Wrong choice"<<endl;
            }
        }
        return 0;
}

/*
 * Create Double Link List
 */
void double_llist::create_list(int value)
{
    struct node *s, *temp;
    temp = new(struct node);
    temp->info = value;
    temp->next = NULL;
    if (start == NULL)
        {
            temp->prev = NULL;
            start = temp;
        }
        else
        {
            s = start;
            while (s->next != NULL)
                s = s->next;
            s->next = temp;
            temp->prev = s;
        }
}

/*
 * Insertion at the beginning
 */
void double_llist::add_begin(int value)
{
        if (start == NULL)
        {
            cout<<"First Create the list."<<endl;
            return;
        }
        struct node *temp;
        temp = new(struct node);
        temp->prev = NULL;
        temp->info = value;
        temp->next = start;
        start->prev = temp;
        start = temp;
        cout<<"Element Inserted"<<endl;
}

/*
 * Insertion of element at a particular position
 */
void double_llist::add_after(int value, int pos)
{
        if (start == NULL)
        {
            cout<<"First Create the list."<<endl;
            return;
        }
        struct node *tmp, *q;
        int i;
        q = start;
        for (i = 0;i < pos - 1;i++)
        {
            q = q->next;
            if (q == NULL)
            {
                cout<<"There are less than ";
                cout<<pos<<" elements."<<endl;
                return;
            }
        }
        tmp = new(struct node);
        tmp->info = value;
        if (q->next == NULL)
        {
            q->next = tmp;
            tmp->next = NULL;
            tmp->prev = q;
        }
        else
        {
            tmp->next = q->next;
            tmp->next->prev = tmp;
            q->next = tmp;
            tmp->prev = q;
        }
        cout<<"Element Inserted"<<endl;
}

/*
 * Deletion of element from the list
 */
void double_llist::delete_element(int value)
{
        struct node *tmp, *q;
         /*first element deletion*/
        if (start->info == value)
        {
            tmp = start;
            start = start->next;
            start->prev = NULL;
            cout<<"Element Deleted"<<endl;
            free(tmp);
            return;
        }
        q = start;
        while (q->next->next != NULL)
        {
            /*Element deleted in between*/
            if (q->next->info == value)
            {
                tmp = q->next;
                q->next = tmp->next;
                tmp->next->prev = q;
                cout<<"Element Deleted"<<endl;
                free(tmp);
                return;
            }
            q = q->next;
        }
         /*last element deleted*/
        if (q->next->info == value)
        {
            tmp = q->next;
            free(tmp);
            q->next = NULL;
            cout<<"Element Deleted"<<endl;
            return;
        }
        cout<<"Element "<<value<<" not found"<<endl;
}

/*
 * Display elements of Doubly Link List
 */
void double_llist::display_dlist()
{
        struct node *q;
        if (start == NULL)
        {
            cout<<"List empty,nothing to display"<<endl;
            return;
        }
        q = start;
        cout<<"The Doubly Link List is :"<<endl;
        while (q != NULL)
        {
            cout<<q->info<<" <-> ";
            q = q->next;
        }
        cout<<"NULL"<<endl;
}

/*
 * Number of elements in Doubly Link List
 */
void double_llist::count()
{
        struct node *q = start;
        int cnt = 0;
        while (q != NULL)
        {
            q = q->next;
            cnt++;
        }
        cout<<"Number of elements are: "<<cnt<<endl;
}

/*
 * Reverse Doubly Link List
 */
void double_llist::reverse()
{
        struct node *p1, *p2;
```



```
---------------------------

Operations on Doubly linked list

---------------------------
1.Create Node
2.Add at begining
3.Add after position
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : _
```

Figure 1: output

```
    p1 = start;
    p2 = p1->next;
    p1->next = NULL;
    p1->prev = p2;
    while (p2 != NULL)
    {
        p2->prev = p2->next;
        p2->next = p1;
        p1 = p2;
        p2 = p2->prev;
    }
    start = p1;
    cout<<”List Reversed”<<endl;
}
```