

# DATA STRUCTURE AND ALOGRITHUM

**Lab Report**

Name: SAMI ULLAH

Registration #: SEU-S17-030

Lab Report #: 06

Dated: 5-21-2018

Submitted To: Mr. Usman Ahmed

The University of Lahore, Islamabad Campus

Department of Computer Science & Information Technology

**Experiment # 1**

**DOUBLE LINK LIST**

**Objective**

To understand the meaninig and implementation of double link list.

**Software Tool**

1.

DEV C++

# Theory

Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, either forward and backward easily as compared to Single Linked List. Following are the important terms to understand the concept of doubly linked list. Link Each link of a linked list can store a data called an element. Next Each link of a linked list contains a link to the next link called Next. Prev Each link of a linked list contains a link to the previous link called Prev. LinkedList A Linked List contains the connection link to the first link called First and to the last link called Last. Doubly Linked List Representation

As per the above illustration, following are the important points to be considered. Doubly Linked List contains a link element called first and last. Each link carries a data field(s) and two link fields called next and prev. Each link is linked with its next link using its next link. Each link is linked with its previous link using its previous link. The last link carries a link as null to mark the end of the list. Basic Operations Following are the basic operations supported by a list. 1. CREATE NEW NODE 2. ADD AT BEGINNING 3. ADD AFTER POSITIO 4. DELETE 5. DISPLAY 6.

COUNT 7. REVERSE 8. QUIT

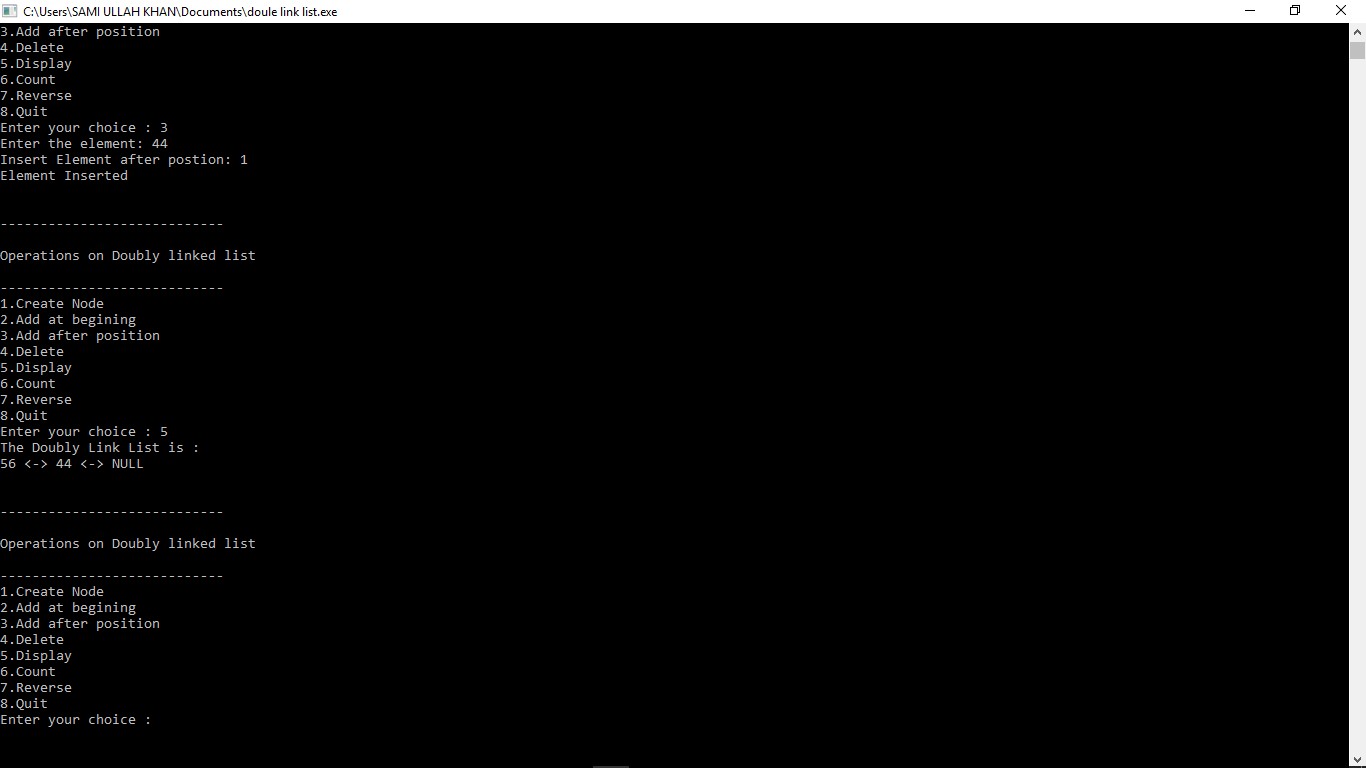


Figure 1: Time Independent Feature Set

# Task

**2.1 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 l i s t ”*<<*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 l i s t . ”*<<*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 l i s t . ”*<<*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 ;

/∗ f i r s t 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*<<*”TheDoubly 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 ;

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 ; }

# Conclusion

in this lab we perform the basics function of double link list insertion deletion insertion at any n postion display reverse etc