

AIM Implementation of Linked List Data Structure

DOUBLY LINKED LIST

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>

struct node {
    int data;
    int key;

    struct node *next;
    struct node *prev;
};

//this link always point to first Link
struct node *head = NULL;

//this link always point to last Link
struct node *last = NULL;

struct node *current = NULL;

//is list empty
bool isEmpty() {
    return head == NULL;
}

int length() {
    int length = 0;
    struct node *current;
```

```
for(current = head; current != NULL; current = current->next){  
    length++;  
}  
  
return length;  
}
```

```
//display the list in from first to last  
void displayForward() {
```

```
    //start from the beginning  
    struct node *ptr = head;
```

```
    //navigate till the end of the list  
    printf("\n[ ");
```

```
    while(ptr != NULL) {  
        printf("(%d,%d) ",ptr->key,ptr->data);  
        ptr = ptr->next;  
    }
```

```
    printf(" ]");  
}
```

```
//display the list from last to first  
void displayBackward() {
```

```
    //start from the last  
    struct node *ptr = last;
```

```
    //navigate till the start of the list  
    printf("\n[ ");
```

```

while(ptr != NULL) {

    //print data
    printf("(%d,%d) ",ptr->key,ptr->data);

    //move to next item
    ptr = ptr ->prev;

}

}

//insert link at the first location
void insertFirst(int key, int data) {

    //create a link
    struct node *link = (struct node*) malloc(sizeof(struct node));
    link->key = key;
    link->data = data;

    if(isEmpty()) {
        //make it the last link
        last = link;
    } else {
        //update first prev link
        head->prev = link;
    }

    //point it to old first link
    link->next = head;

    //point first to new first link

```

```
    head = link;
}
```

```
//insert link at the last location
```

```
void insertLast(int key, int data) {
```

```
    //create a link
```

```
    struct node *link = (struct node*) malloc(sizeof(struct node));
```

```
    link->key = key;
```

```
    link->data = data;
```

```
    if(isEmpty()) {
```

```
        //make it the last link
```

```
        last = link;
```

```
    } else {
```

```
        //make link a new last link
```

```
        last->next = link;
```

```
        //mark old last node as prev of new link
```

```
        link->prev = last;
```

```
    }
```

```
//point last to new last node
```

```
last = link;
```

```
}
```

```
//delete first item
```

```
struct node* deleteFirst() {
```

```
    //save reference to first link
```

```
    struct node *tempLink = head;
```

```
    //if only one link
```

```
if(head->next == NULL){
    last = NULL;
} else {
    head->next->prev = NULL;
}

head = head->next;
//return the deleted link
return tempLink;
}
```

//delete link at the last location

```
struct node* deleteLast() {
    //save reference to last link
    struct node *tempLink = last;

    //if only one link
    if(head->next == NULL) {
        head = NULL;
    } else {
        last->prev->next = NULL;
    }

    last = last->prev;

    //return the deleted link
    return tempLink;
}
```

//delete a link with given key

```
struct node* delete(int key) {
```

```

//start from the first link
struct node* current = head;
struct node* previous = NULL;

//if list is empty
if(head == NULL) {
    return NULL;
}

//navigate through list
while(current->key != key) {
    //if it is last node

    if(current->next == NULL) {
        return NULL;
    } else {
        //store reference to current link
        previous = current;

        //move to next link
        current = current->next;
    }
}

//found a match, update the link
if(current == head) {
    //change first to point to next link
    head = head->next;
} else {
    //bypass the current link
    current->prev->next = current->next;
}

```

```

if(current == last) {
    //change last to point to prev link
    last = current->prev;
} else {
    current->next->prev = current->prev;
}

return current;
}

bool insertAfter(int key, int newKey, int data) {
    //start from the first link
    struct node *current = head;

    //if list is empty
    if(head == NULL) {
        return false;
    }

    //navigate through list
    while(current->key != key) {

        //if it is last node
        if(current->next == NULL) {
            return false;
        } else {
            //move to next link
            current = current->next;
        }
    }

    //create a link

```

```
struct node *newLink = (struct node*) malloc(sizeof(struct node));
newLink->key = newKey;
newLink->data = data;

if(current == last) {
    newLink->next = NULL;
    last = newLink;
} else {
    newLink->next = current->next;
    current->next->prev = newLink;
}

newLink->prev = current;
current->next = newLink;
return true;
}
```

```
void main() {
    insertFirst(1,10);
    insertFirst(2,20);
    insertFirst(3,30);
    insertFirst(4,1);
    insertFirst(5,40);
    insertFirst(6,56);

    printf("\nList (First to Last): ");
    displayForward();

    printf("\n");
    printf("\nList (Last to first): ");
    displayBackward();

    printf("\nList , after deleting first record: ");
}
```



```

deleteFirst();
displayForward();

printf("\nList , after deleting last record: ");
deleteLast();
displayForward();

printf("\nList , insert after key(4) : ");
insertAfter(4,7, 13);
displayForward();

printf("\nList , after delete key(4) : ");
delete(4);
displayForward();
}

```

OUTPUT

```

List (First to Last):
[ (6,56) (5,40) (4,1) (3,30) (2,20) (1,10) ]

List (Last to first):
[ (1,10) (2,20) (3,30) (4,1) (5,40) (6,56) ]
List , after deleting first record:
[ (5,40) (4,1) (3,30) (2,20) (1,10) ]
List , after deleting last record:
[ (5,40) (4,1) (3,30) (2,20) ]
List , insert after key(4) :
[ (5,40) (4,1) (7,13) (3,30) (2,20) ]
List , after delete key(4) :
[ (5,40) (4,13) (3,30) (2,20) ]

```

Algo written in comments too

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.

CIRCULAR LINKED LIST USING C

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
```

```
struct node {
    int data;
    int key;

    struct node *next;
};
```

```
struct node *head = NULL;
struct node *current = NULL;
```

```
bool isEmpty() {
    return head == NULL;
}
```

```
int length() {
    int length = 0;
```

```
    if(head == NULL) {
        return 0;
    }
```

```
    current = head->next;
```

```

while(current != head) {
    length++;
    current = current->next;
}

return length;
}

//insert link at the first location
void insertFirst(int key, int data) {

    //create a link
    struct node *link = (struct node*) malloc(sizeof(struct node));
    link->key = key;
    link->data = data;

    if (isEmpty()) {
        head = link;
        head->next = head;
    } else {
        //point it to old first node
        link->next = head;

        //point first to new first node
        head = link;
    }
}

```

```
//delete first item
struct node * deleteFirst() {

    //save reference to first link
    struct node *tempLink = head;

    if(head->next == head) {
        head = NULL;
        return tempLink;
    }

    //mark next to first link as first
    head = head->next;

    //return the deleted link
    return tempLink;
}

//display the list
void printList() {

    struct node *ptr = head;
    printf("\n[ ");

    //start from the beginning
    if(head != NULL) {
```

```

    while(ptr->next != ptr) {
        printf("(%d,%d) ",ptr->key,ptr->data);
        ptr = ptr->next;
    }
}

printf(" ]");
}

int main() {
    insertFirst(1,11);
    insertFirst(2,21);
    insertFirst(3,51);
    insertFirst(4,111);
    insertFirst(5,151);
    insertFirst(6,501);

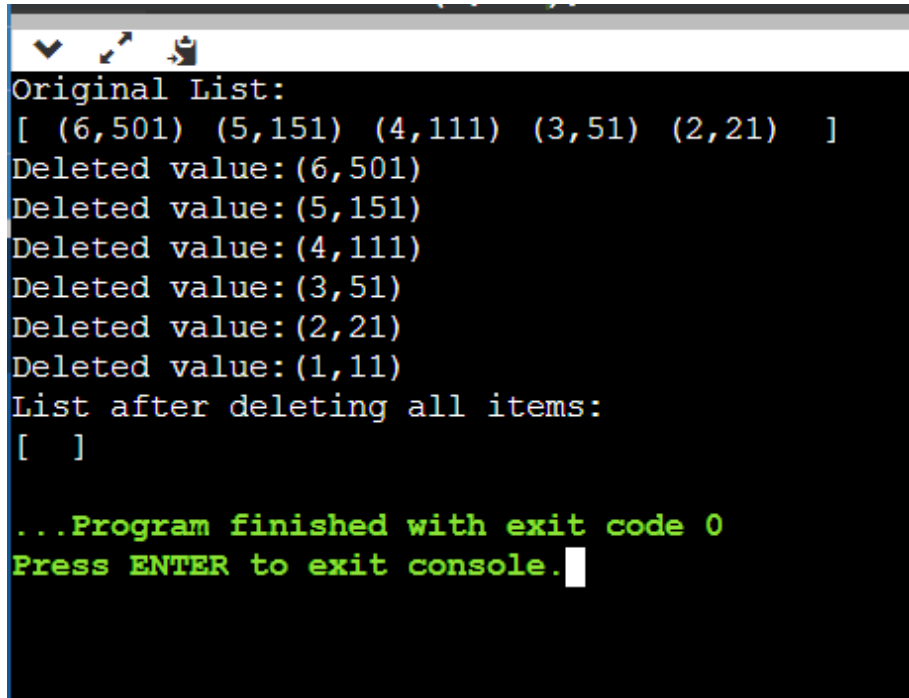
    printf("Original List: ");

    //print list
    printList();

    while(!isEmpty()) {
        struct node *temp = deleteFirst();
        printf("\nDeleted value:");
        printf("(%d,%d) ",temp->key,temp->data);
    }
}

```

```
printf("\nList after deleting all items: ");  
printList();  
}
```



The screenshot shows a console window with a dark background and light-colored text. At the top, there are three small icons: a checkmark, a magnifying glass, and a document. The main text in the console is as follows:

```
Original List:  
[ (6,501) (5,151) (4,111) (3,51) (2,21) ]  
Deleted value:(6,501)  
Deleted value:(5,151)  
Deleted value:(4,111)  
Deleted value:(3,51)  
Deleted value:(2,21)  
Deleted value:(1,11)  
List after deleting all items:  
[ ]  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

DOUBLY CIRCULAR LINKED LIST

```
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int val;
    struct node *next;
    struct node *prev;
};
typedef struct node n;

n* create_node(int);
void add_node();
void insert_at_first();
void insert_at_end();
void insert_at_position();
void delete_node_position();
void sort_list();
void update();
void search();
void display_from_beg();
void display_in_rev();

n *new, *ptr, *prev;
n *first = NULL, *last = NULL;
int number = 0;

void main()
{
```



```
int ch;
```

```
printf("\n linked list\n");
```

```
printf("1.insert at beginning \n 2.insert at end\n 3.insert at position\n4.sort  
linked list\n 5.delete node at position\n 6.updatenodevalue\n7.search element  
\n8.displaylist from beginning\n9.display list from end\n10.exit ");
```

```
while (1)
```

```
{
```

```
    printf("\n enter your choice:");
```

```
    scanf("%d", &ch);
```

```
    switch (ch)
```

```
    {
```

```
        case 1 :
```

```
            insert_at_first();
```

```
            break;
```

```
        case 2 :
```

```
            insert_at_end();
```

```
            break;
```

```
        case 3 :
```

```
            insert_at_position();
```

```
            break;
```

```
        case 4 :
```

```
            sort_list();
```

```
            break;
```

```
        case 5 :
```

```
            delete_node_position();
```

```
            break;
```

```
        case 6 :
```

```
            update();
```

```
            break;
```

```
        case 7 :
```

```

        search();
        break;
    case 8 :
        display_from_beg();
        break;
    case 9 :
        display_in_rev();
        break;
    case 10 :
        exit(0);
    case 11 :
        add_node();
        break;
    default:
        printf("\ninvalid choice");
    }
}
}
/*
*MEMORY ALLOCATED FOR NODE DYNAMICALLY
*/
n* create_node(int info)
{
    number++;
    new = (n *)malloc(sizeof(n));
    new->val = info;
    new->next = NULL;
    new->prev = NULL;
    return new;
}
/*
*ADDS NEW NODE
*/

```

```

void add_node()
{

    int info;

    printf("\nenter the value you would like to add:");
    scanf("%d", &info);
    new = create_node(info);

    if (first == last && first == NULL)
    {

        first = last = new;
        first->next = last->next = NULL;
        first->prev = last->prev = NULL;
    }
    else
    {
        last->next = new;
        new->prev = last;
        last = new;
        last->next = first;
        first->prev = last;
    }
}
/*
*INSERTS ELEMENT AT FIRST
*/
void insert_at_first()
{

    int info;

```

```

printf("\nenter the value to be inserted at first:");
scanf("%d",&info);
new = create_node(info);

if (first == last && first == NULL)
{
    printf("\ninitially it is empty linked list later insertion is done");
    first = last = new;
    first->next = last->next = NULL;
    first->prev = last->prev = NULL;
}
else
{
    new->next = first;
    first->prev = new;
    first = new;
    first->prev = last;
    last->next = first;
    printf("\n the value is inserted at begining");
}
}
/*
*INSERTS ELEMNET AT END
*/
void insert_at_end()
{

    int info;

    printf("\nenter the value that has to be inserted at last:");
    scanf("%d", &info);
    new = create_node(info);

```

```

    if (first == last && first == NULL)
    {
        printf("\ninitially the list is empty and now new node is inserted but at
first");
        first = last = new;
        first->next = last->next = NULL;
        first->prev = last->prev = NULL;
    }
    else
    {
        last->next = new;
        new->prev = last;
        last = new;
        first->prev = last;
        last->next = first;
    }
}
/*
*INSERTS THE ELEMENT AT GIVEN POSITION
*/
void insert_at_position()
{
    int info, pos, len = 0, i;
    n *prevnode;

    printf("\n enter the value that you would like to insert:");
    scanf("%d", &info);
    printf("\n enter the position where you have to enter:");
    scanf("%d", &pos);
    new = create_node(info);

    if (first == last && first == NULL)
    {

```

```

    if (pos == 1)
    {
        first = last = new;
        first->next = last->next = NULL;
        first->prev = last->prev = NULL;
    }
    else
        printf("\n empty linked list you cant insert at that particular position");
}
else
{
    if (number < pos)
        printf("\n node cant be inserted as position is exceeding the linkedlist
length");

    else
    {
        for (ptr = first, i = 1; i <= number; i++)
        {
            prevnode = ptr;
            ptr = ptr->next;
            if (i == pos-1)
            {
                prevnode->next = new;
                new->prev = prevnode;
                new->next = ptr;
                ptr->prev = new;
                printf("\ninserted at position %d succesfully", pos);
                break;
            }
        }
    }
}
}

```

```

}
/*
*SORTING IS DONE OF ONLY NUMBERS NOT LINKS
*/
void sort_list()
{
    n *temp;
    int tempval, i, j;

    if (first == last && first == NULL)
        printf("\nlinked list is empty no elements to sort");
    else
    {
        for (ptr = first, i = 0; i < number; ptr = ptr->next, i++)
        {
            for (temp = ptr->next, j = i; j < number; j++)
            {
                if (ptr->val > temp->val)
                {
                    tempval = ptr->val;
                    ptr->val = temp->val;
                    temp->val = tempval;
                }
            }
        }
        for (ptr = first, i = 0; i < number; ptr = ptr->next, i++)
            printf("\n%d", ptr->val);
    }
}
/*
*DELETION IS DONE
*/
void delete_node_position()

```

```

{
    int pos, count = 0, i;
    n *temp, *prevnode;

    printf("\n enter the position which u wanted to delete:");
    scanf("%d", &pos);

    if (first == last && first == NULL)
        printf("\n empty linked list you cant delete");

    else
    {
        if (number < pos)
            printf("\n node cant be deleted at position as it is exceeding the linkedlist
length");

        else
        {
            for (ptr = first, i = 1; i <= number; i++)
            {
                prevnode = ptr;
                ptr = ptr->next;
                if (pos == 1)
                {
                    number--;
                    last->next = prevnode->next;
                    ptr->prev = prevnode->prev;
                    first = ptr;
                    printf("%d is deleted", prevnode->val);
                    free(prevnode);
                    break;
                }
                else if (i == pos - 1)

```



```

        {
            number--;
            prevnode->next = ptr->next;
            ptr->next->prev = prevnode;
            printf("%d is deleted", ptr->val);
            free(ptr);
            break;
        }
    }
}
}
}
}
/*
*UPDATION IS DONE FRO GIVEN OLD VAL
*/
void update()
{
    int oldval, newval, i, f = 0;
    printf("\n enter the value old value:");
    scanf("%d", &oldval);
    printf("\n enter the value new value:");
    scanf("%d", &newval);
    if (first == last && first == NULL)
        printf("\n list is empty no elemnts for updation");
    else
    {
        for (ptr = first, i = 0; i < number; ptr = ptr->next, i++)
        {
            if (ptr->val == oldval)
            {
                ptr->val = newval;
                printf("value is updated to %d", ptr->val);
                f = 1;
            }
        }
    }
}

```

```

    }
}
if (f == 0)
    printf("\n no such old value to be get updated");
}
}
/*
*SEARCHING USING SINGLE KEY
*/
void search()
{
    int count = 0, key, i, f = 0;

    printf("\n enter the value to be searched:");
    scanf("%d", &key);

    if (first == last && first == NULL)
        printf("\n list is empty no elemnets in list to search");
    else
    {
        for (ptr = first, i = 0; i < number; i++, ptr = ptr->next)
        {
            count++;
            if (ptr->val == key)
            {
                printf("\n the value is found at position at %d", count);
                f = 1;
            }
        }
        if (f == 0)
            printf("\n the value is not found in linkedlist");
    }
}

```

```

/*
*DISPLAYING IN BEGINNING
*/
void display_from_beg()
{
    int i;
    if (first == last && first == NULL)
        printf("\nlist is empty no elemnts to print");
    else
    {
        printf("\n%d number of nodes are there", number);
        for (ptr = first, i = 0; i < number; i++, ptr = ptr->next)
            printf("\n %d", ptr->val);
    }
}
/*
* DISPLAYING IN REVERSE
*/
void display_in_rev()
{
    int i;
    if (first == last && first == NULL)
        printf("\nlist is empty there are no elments");
    else
    {
        for (ptr = last, i = 0; i < number; i++, ptr = ptr->prev)
        {
            printf("\n%d", ptr->val);
        }
    }
}

```

Output

linked list

1.insert at beginning

2.insert at end

3.insert at position

4.sort linked list

5.delete node at position

6.updatenodevalue

7.search element

8.displaylist from beginning

9.display list from end

10.exit

enter your choice:8

list is empty no elemnts to print

enter your choice:5

enter the position which u wanted to delete:2

empty linked list you cant delete

enter your choice:6

enter the value old value:6

enter the value new value:8

list is empty no elemnts for updation

enter your choice:7

enter the value to be searched:57

list is empty no elements in list to search

enter your choice:1

enter the value to be inserted at first:11

initially it is empty linked list later insertion is done

enter your choice:3

enter the value that you would like to insert:5

enter the position where you have to enter:5

node can't be inserted as position is exceeding the linkedlist length

enter your choice:1

enter the value to be inserted at first:56

the value is inserted at beginning

enter your choice:1

enter the value to be inserted at first:89

the value is inserted at beginning

enter your choice:2

enter the value that has to be inserted at last:89

enter your choice:2

enter the value that has to be inserted at last:45

enter your choice:

6 number of nodes are there

```
89
56
11
89
45
89
enter your choice:4
```

```
11
89
89
45
56
11
enter your choice:10
```

Algorithm(circular doubly)

- Step 1: IF PTR = NULL.
- Step 2: SET NEW_NODE = PTR.
- Step 3: SET PTR = PTR -> NEXT.
- Step 4: SET NEW_NODE -> DATA = VAL.
- Step 5: SET TEMP = HEAD.
- Step 6: Repeat Step 7 while TEMP -> NEXT != HEAD.
- Step 7: SET TEMP = TEMP -> NEXT.
- Step 8: SET TEMP -> NEXT = NEW_NODE.

Insertion in doubly linked list

To insert a node at the end of the list, follow these steps:

1. Create a node, say T.
2. Make T -> next = last -> next;
3. last -> next = T.
4. last = T.

