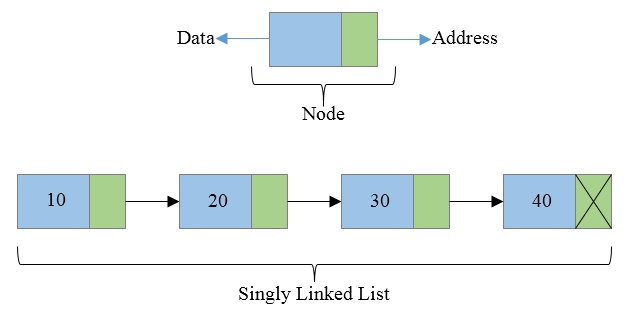
**Godavari College Of Engineering, Jalgaon.**

**Subject Name:** Data Structure. **Teacher Name:** Prof. S.S.Shete

**Practical No. :**  9 **Date:**

**Class:** S.E **Roll No:**

**Title:** Write programs to implement the following data structures: (a) Single linked list (b)

Double linked list.

**Aim:** To implement the following data structures: (a) Single linked list (b) Double linked list.

**Theory:**

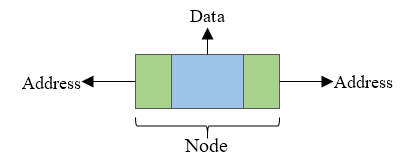
Linked List :- Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers.

**a) Singly Linked List :-** Singly linked list is a basic linked list type. Singly linked list is a collection of nodes linked together in a sequential way where each node of singly linked list contains a data field and an address field which contains the reference of the next node. Singly linked list can contain multiple data fields but should contain at least single address field pointing to its connected next node.

**Advantages of Singly Linked List :-**

* Insertion and deletion of element can be done easily.
* Insertion and deletion of elements doesn't requires movement of all elements when compared to an array.
* Requires less memory when compared to doubly, circular or doubly circular linked list.
* Can allocate or deallocate memory easily when required during its execution.
* It is one of most efficient data structure to implement when traversing in one direction is required.

**b) Doubly Linked List. :-** Doubly linked list is a collection of nodes linked together in a sequential way. Each node of the list contains two parts (as in singly linked list. ) data part and the reference or address part. Since doubly linked list allows the traversal of nodes in both direction hence we can keep track of both first and last nodes. The basic structure of node is shown in the below image:



## 

## **Advantages of Doubly linked list**

* Allows traversal of nodes in both direction which is not possible in singly linked list.
* Deletion of nodes is easy when compared to [singly linked list](https://codeforwin.org/2015/09/c-program-to-delete-middle-node-of-singly-linked-list.html), as in singly linked list deletion requires a pointer to the node and previous node to be deleted. Which is not in case of doubly linked list we only need the pointer which is to be deleted.
* Reversing the list is simple and straightforward.
* Can allocate or de-allocate memory easily when required during its execution.
* It is one of most efficient data structure to implement when traversing in both direction is required.

**Program:**

**1) Singly Linked List. :-**

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

struct node

{

int value;

struct node \*next;

};

void insert();

void display();

void del();

int count();

typedef struct node DATA\_NODE;

DATA\_NODE \*head\_node, \*first\_node, \*temp\_node = 0, \*prev\_node, next\_node;

int data;

int main()

{

int ch= 0;

printf("\n\n\t Singly Linked List Example - All Operations\n");

printf("\n\t Options\n");

printf("\n\t1 : Insert into Linked List \n");

printf("\t2 : Delete from Linked List \n");

printf("\t3 : Display Linked List\n");

printf("\t4 : Count Linked List\n");

printf("\t5 :Exit. \n");

rep:

printf("\n====================================================");

printf("\n\tEnter your option:- ");

scanf("%d", &ch);

switch (ch)

{

case 1:

insert();

break;

case 2:

del();

break;

case 3:

display();

break;

case 4:

count();

break;

case 5:

printf("\n\n\t Thanks For Visiting Program.");

printf("\n\t Now You Are Exiting From Program.");

break;

default:

break;

} // end switch

if(ch!=5)

{

goto rep;

}

return 0;

}

void insert()

{

printf("\nEnter Element for Insert Linked List : \n");

scanf("%d", &data);

temp\_node = (DATA\_NODE \*) malloc(sizeof (DATA\_NODE));

temp\_node->value = data;

if (first\_node == 0) {

first\_node = temp\_node;

} else {

head\_node->next = temp\_node;

}

temp\_node->next = 0;

head\_node = temp\_node;

fflush(stdin);

}

void del()

{

int countvalue, pos, i = 0;

countvalue = count();

temp\_node = first\_node;

printf("\nDisplay Linked List : \n");

printf("\nEnter Position for Delete Element : \n");

scanf("%d", &pos);

if (pos > 0 && pos <= countvalue)

{

if (pos == 1)

{

temp\_node = temp\_node -> next;

first\_node = temp\_node;

printf("\nDeleted Successfully \n\n");

}

else

{

while (temp\_node != 0)

{

if (i == (pos - 1))

{

prev\_node->next = temp\_node->next;

if(i == (countvalue - 1))

{

head\_node = prev\_node;

}

printf("\nDeleted Successfully \n\n");

break;

}

else

{

i++;

prev\_node = temp\_node;

temp\_node = temp\_node -> next;

}

}

}

} else

printf("\nInvalid Position \n\n");

}

void display()

{

int count = 0;

temp\_node = first\_node;

printf("\n\t Display Linked List : \n");

while (temp\_node != 0) {

printf(" %d ", temp\_node->value);

count++;

temp\_node = temp\_node -> next;

}

printf("\nNo Of Items In Linked List : %d\n", count);

}

int count() {

int count = 0;

temp\_node = first\_node;

while (temp\_node != 0) {

count++;

temp\_node = temp\_node -> next;

}

printf("\nNo Of Items In Linked List : %d\n", count);

return count;

}

**2) Doubly Linked List. :-**

#include<stdio.h>

#include<stdlib.h>

struct node

{

struct node \*prev;

struct node \*next;

int data;

};

struct node \*head;

void insertion\_beginning();

void insertion\_last();

void insertion\_specified();

void deletion\_beginning();

void deletion\_last();

void deletion\_specified();

void display();

void search();

int main ()

{

int choice =0;

printf("\n\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\n");

printf("\n===============================================\n");

printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random location\n4.Delete from Beginning. \n 5.Delete from last\n6.Delete the node after the given data\n7.Search\n8.Show\n9.Exit\n");

rep:

printf("\n==========================================================");

printf("\nEnter your choice :- ");

scanf("\n%d",&choice);

switch(choice)

{

case 1:

insertion\_beginning();

break;

case 2:

insertion\_last();

break;

case 3:

insertion\_specified();

break;

case 4:

deletion\_beginning();

break;

case 5:

deletion\_last();

break;

case 6:

deletion\_specified();

break;

case 7:

search();

break;

case 8:

display();

break;

case 9:

printf("\n\t Thanks For visiting Program.");

printf("\n\t NOw You Are Exiting From Program.");

exit(0);

break;

default:

printf("Please enter valid choice..");

}

if(choice!=9)

{

goto rep;

}

}

void insertion\_beginning()

{

struct node \*ptr;

int item;

ptr = (struct node \*)malloc(sizeof(struct node));

if(ptr == NULL)

{

printf("\n\t Memory is not Allocated for New NODE...");

}

else

{

printf("\n\t Enter Item value :- ");

scanf("%d",&item);

if(head==NULL)

{

ptr->next = NULL;

ptr->prev=NULL;

ptr->data=item;

head=ptr;

}

else

{

ptr->data=item;

ptr->prev=NULL;

ptr->next = head;

head->prev=ptr;

head=ptr;

}

printf("\n Node is inserted at Beginning of Doubly Linked List.");

}

}

void insertion\_last()

{

struct node \*ptr,\*temp;

int item;

ptr = (struct node \*) malloc(sizeof(struct node));

if(ptr == NULL)

{

printf("\n\t Memory is not Allocated for New NODE...");

}

else

{

printf("\n\t Enter value :- ");

scanf("%d",&item);

ptr->data=item;

if(head == NULL)

{

ptr->next = NULL;

ptr->prev = NULL;

head = ptr;

}

else

{

temp = head;

while(temp->next!=NULL)

{

temp = temp->next;

}

temp->next = ptr;

ptr ->prev=temp;

ptr->next = NULL;

}

}

printf("\n Node is inserted At Last Of Doubly Linked List.");

}

void insertion\_specified()

{

struct node \*ptr,\*temp;

int item,loc,i;

ptr = (struct node \*)malloc(sizeof(struct node));

if(ptr == NULL)

{

printf("\n\t Memory is not Allocated for New NODE...");

}

else

{

temp=head;

printf("\n\t Enter the location");

scanf("%d",&loc);

for(i=0;i<loc;i++)

{

temp = temp->next;

if(temp == NULL)

{

printf("\n\t There are less than %d elements", loc);

return;

}

}

printf("\n\t Enter value");

scanf("%d",&item);

ptr->data = item;

ptr->next = temp->next;

ptr -> prev = temp;

temp->next = ptr;

temp->next->prev=ptr;

printf("\n Node inserted at %d Location. ",loc);

}

}

void deletion\_beginning()

{

struct node \*ptr;

if(head == NULL)

{

printf("\n\t Memory is not Allocated for New NODE...");

}

else if(head->next == NULL)

{

head = NULL;

free(head);

printf("\n Node Deleted At Beginning of Linked List.");

}

else

{

ptr = head;

head = head -> next;

head -> prev = NULL;

free(ptr);

printf("\n Node Deleted At Bigging of Linked List.");

}

}

void deletion\_last()

{

struct node \*ptr;

if(head == NULL)

{

printf("\n\t Doubly Linked List is Empty.");

}

else if(head->next == NULL)

{

head = NULL;

free(head);

printf("\n Node Deleted From Last Of Linked List. ");

}

else

{

ptr = head;

while(ptr->next != NULL)

{

ptr = ptr -> next;

}

ptr->prev-> next= NULL;

free(ptr);

printf("\n Node Deleted From Last Of Linked List.");

}

}

void deletion\_specified()

{

struct node \*ptr, \*temp;

int val;

printf("\n Enter the data after which the node is to be deleted :- ");

scanf("%d", &val);

ptr = head;

while(ptr -> data != val)

ptr = ptr -> next;

if(ptr -> next == NULL)

{

printf("\n Node is Can't Deleted From Specific Location.");

}

else if(ptr -> next -> next == NULL)

{

ptr ->next = NULL;

printf("\n Node is Deleted From Specific Location.");

}

else

{

temp = ptr -> next;

ptr -> next = temp -> next;

temp -> next -> prev = ptr;

free(temp);

printf("\n Node is Deleted From Specific Location.");

}

}

void display()

{

struct node \*ptr;

printf("\n Doubly Linked List is :- \n");

ptr = head;

while(ptr != NULL)

{

printf("%d\n",ptr->data);

ptr=ptr->next;

}

}

void search()

{

struct node \*ptr;

int item,i=0,flag;

ptr = head;

if(ptr == NULL)

{

printf("\n\t Empty List\n");

}

else

{

printf("\n Enter item which you want to search?\n");

scanf("%d",&item);

while (ptr!=NULL)

{

if(ptr->data == item)

{

printf("\n Item found at location %d ",i+1);

flag=0;

break;

}

else

{

flag=1;

}

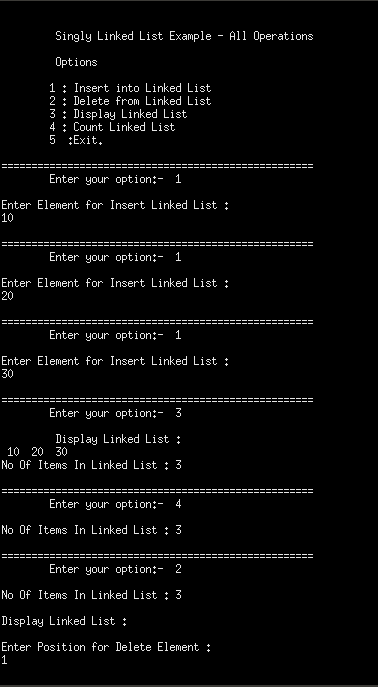
i++;

ptr = ptr -> next;

}

if(flag==1)

{

 printf("\nItem not found.");

} }

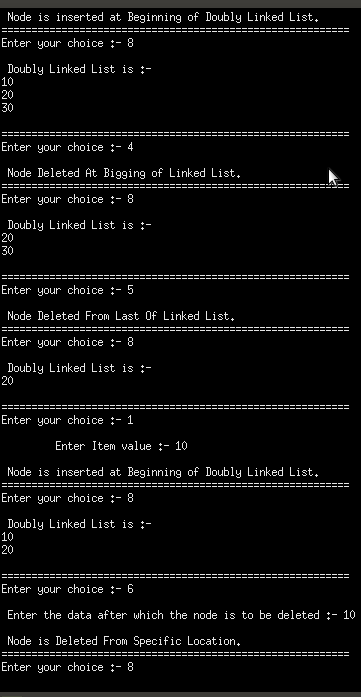
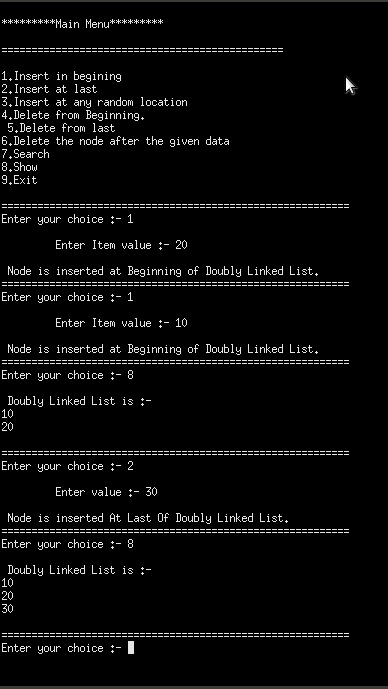
}

**Output**:

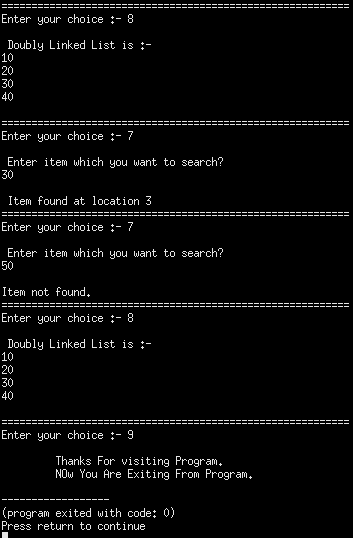
**1) Singly Linked List. :-**



**2) Doubly Linked List. :-**

****

**fig a) insert doubly linked list. Fig. b) delete from doubly linked list.**

****

**Fig c) Search in Doubly Linked List.**

**Conclusion:**