**PROGRAM-6**

***Q. Design and implement a dynamic list (Singly linked list / Doubly linked list) to store any information which needs a linear data structure.***

**“Doubly Linked List”**

* **Theory:**

Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence. Therefore, in a doubly linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer), pointer to the previous node (previous pointer).

Both the previous and subsequent parts of the first and last nodes will always be null, signifying an end in both directions.

Because each node in a singly linked list has the address of the next node and lacks a record of its previous nodes, we can only traverse the list in one direction. Nevertheless, doubly linked lists get over this single linked list restriction. Each node in the list has the address of the node that came before it, therefore using the prior address stored in the previous part of each node, we can also obtain all the information about the previous node.

**Different operations performed on Doubly Linked list:**

1. Insertion at the beginning
2. Insertion at the end
3. Insertion at specified position
4. Deletion from the beginning
5. Deletion from the end
6. Deletion from specified position
7. Searching the element
8. Display the elements



//------------------------------------------------------------

* **Algorithm:**
* **Node creation:**

struct node

{

int data;

struct node \*next;

struct node \*prev;

};

struct node \*head, \*ptr;

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

* **Insertion:**

Step-1: Create nodes based on the type of insertion being done.

Step-2: Allocate the memory dynamically using malloc () function.

Step-3: Check for exceptions like overflow and underflow being done.

Step-4: Accept the data from the user.

Step-5: Update the data parts and link part.

Step-6: Insert the data into the linked list.

Step-7: END

* **Deletion:**

Step-1: First of all, check whether the list is empty.

Step-2: If list is empty, print deletion is not possible.

Step-3: delete the nodes based on the type of deletion being done.

Step-4: Free the nodes using free () function to avoid memory wastage.

//--------------------------------------------------------------

* **Code:**

#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();

void main ()

{

int choice =0;

while(choice != 9)

{

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

printf("\nChoose one option from the following list ...\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");

printf("\nEnter your choice?\n");

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:

exit(0);

break;

default:

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

}

}

}

void insertion\_beginning()

{

struct node \*ptr;

int item;

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

if(ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter 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("\nNode inserted\n");

}

}

void insertion\_last()

{

struct node \*ptr,\*temp;

int item;

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

if(ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter 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("\nnode inserted\n");

}

void insertion\_specified()

{

struct node \*ptr,\*temp;

int item,loc,i;

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

if(ptr == NULL)

{

printf("\n OVERFLOW");

}

else

{

temp=head;

printf("Enter the location");

scanf("%d",&loc);

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

{

temp = temp->next;

if(temp == NULL)

{

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

return;

}

}

printf("Enter value");

scanf("%d",&item);

ptr->data = item;

ptr->next = temp->next;

ptr -> prev = temp;

temp->next = ptr;

temp->next->prev=ptr;

printf("\nnode inserted\n");

}

}

void deletion\_beginning()

{

struct node \*ptr;

if(head == NULL)

{

printf("\n UNDERFLOW");

}

else if(head->next == NULL)

{

head = NULL;

free(head);

printf("\nnode deleted\n");

}

else

{

ptr = head;

head = head -> next;

head -> prev = NULL;

free(ptr);

printf("\nnode deleted\n");

}

}

void deletion\_last()

{

struct node \*ptr;

if(head == NULL)

{

printf("\n UNDERFLOW");

}

else if(head->next == NULL)

{

head = NULL;

free(head);

printf("\nnode deleted\n");

}

else

{

ptr = head;

if(ptr->next != NULL)

{

ptr = ptr -> next;

}

ptr -> prev -> next = NULL;

free(ptr);

printf("\nnode deleted\n");

}

}

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("\nCan't delete\n");

}

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

{

ptr ->next = NULL;

}

else

{

temp = ptr -> next;

ptr -> next = temp -> next;

temp -> next -> prev = ptr;

free(temp);

printf("\nnode deleted\n");

}

}

void display()

{

struct node \*ptr;

printf("\n printing values...\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("\nEmpty List\n");

}

else

{

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

scanf("%d",&item);

while (ptr!=NULL)

{

if(ptr->data == item)

{

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

flag=0;

break;

}

else

{

flag=1;

}

i++;

ptr = ptr -> next;

}

if(flag==1)

{

printf("\nItem not found\n");

}

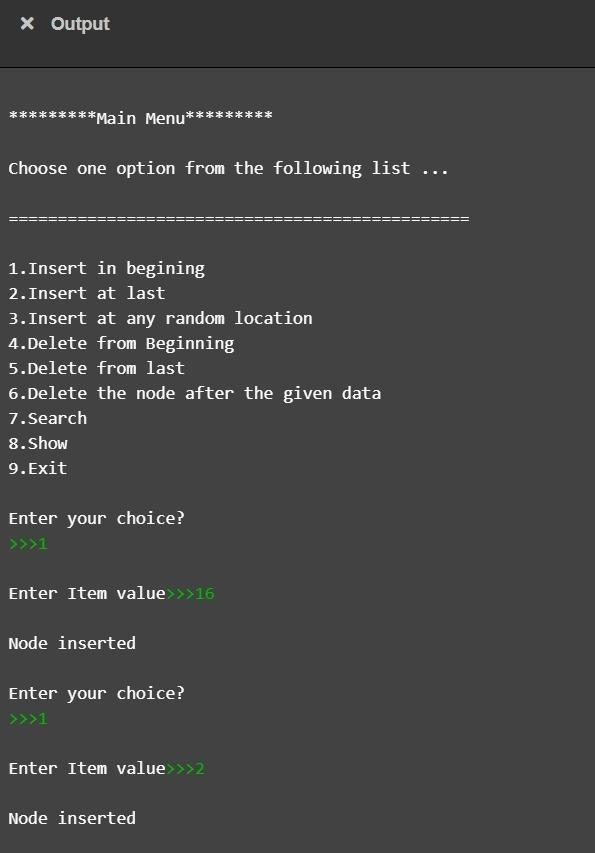
}

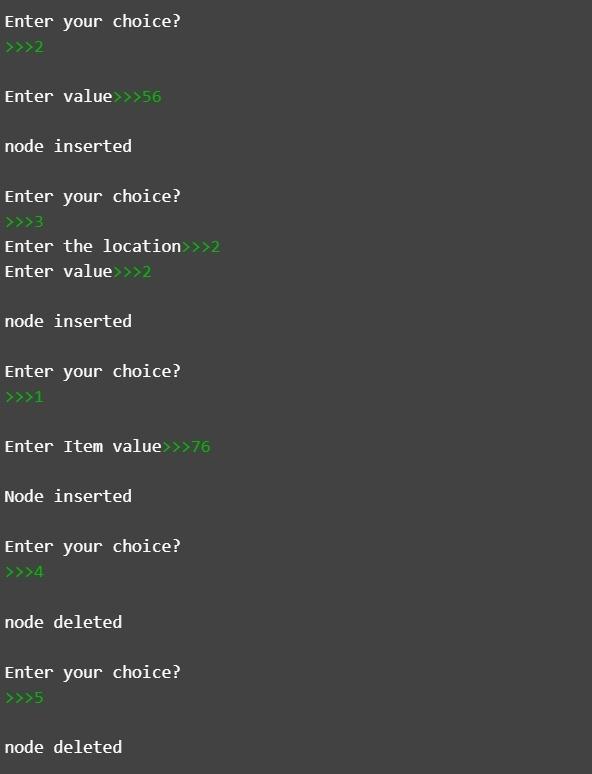
}

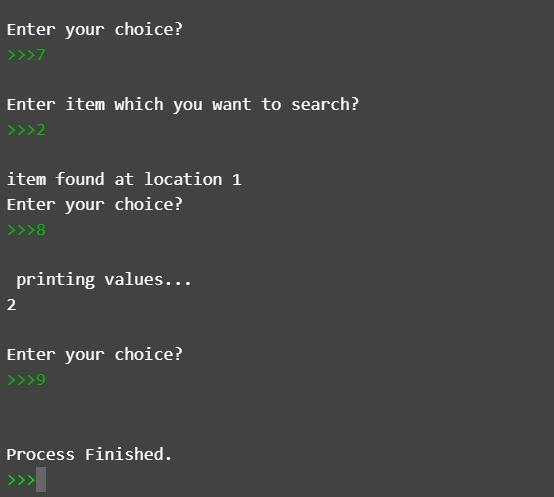
//END OF THE PROGRAM

//-----------------------------------------------------------------

**“SCREENSHOTS OF OUTPUT”**

****

****

****