**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.***

**“Singly Linked List”**

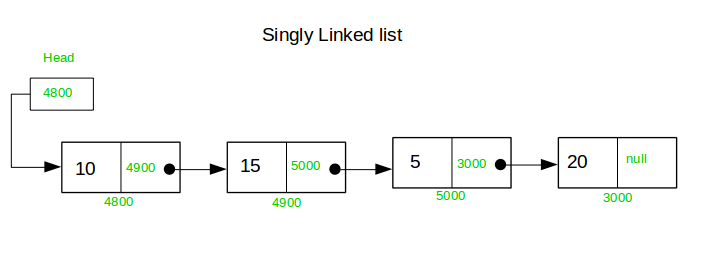
* **Theory:**

A single linked list is a collection of an ordered set of elements. Depending on the requirements of the programme, the number of pieces may change. Data and link components make up a node in a single linked list. While the link portion of the node contains the address of its immediate successor, the data portion of the node keeps the actual information that will be represented by the node.

A single-linked list or a one-way chain can only be traversed in one direction. In other words, since each node only carries a pointer to the next item, we cannot traverse the list backwards.

**Different operations performed on singly 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:**

**|||||||||**

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

* **Code:**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*head;

void beginsert ();

void lastinsert ();

void randominsert();

void begin\_delete();

void last\_delete();

void random\_delete();

void display();

void search();

void main ()

{

int choice =0;

while(choice != 9)

{

printf("\n\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 node after specified location\n7.Search for an element\n8.Show\n9.Exit\n");

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

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

switch(choice)

{

case 1:

beginsert();

break;

case 2:

lastinsert();

break;

case 3:

randominsert();

break;

case 4:

begin\_delete();

break;

case 5:

last\_delete();

break;

case 6:

random\_delete();

break;

case 7:

search();

break;

case 8:

display();

break;

case 9:

exit(0);

break;

default:

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

}

}

}

void beginsert()

{

struct node \*ptr;

int item;

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

if(ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter value\n");

scanf("%d",&item);

ptr->data = item;

ptr->next = head;

head = ptr;

printf("\nNode inserted");

}

}

void lastinsert()

{

struct node \*ptr,\*temp;

int item;

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

if(ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter value?\n");

scanf("%d",&item);

ptr->data = item;

if(head == NULL)

{

ptr -> next = NULL;

head = ptr;

printf("\nNode inserted");

}

else

{

temp = head;

while (temp -> next != NULL)

{

temp = temp -> next;

}

temp->next = ptr;

ptr->next = NULL;

printf("\nNode inserted");

}

}

}

void randominsert()

{

int i,loc,item;

struct node \*ptr, \*temp;

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

if(ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter element value");

scanf("%d",&item);

ptr->data = item;

printf("\nEnter the location after which you want to insert ");

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

temp=head;

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

{

temp = temp->next;

if(temp == NULL)

{

printf("\ncan't insert\n");

return;

}

}

ptr ->next = temp ->next;

temp ->next = ptr;

printf("\nNode inserted");

}

}

void begin\_delete()

{

struct node \*ptr;

if(head == NULL)

{

printf("\nList is empty\n");

}

else

{

ptr = head;

head = ptr->next;

free(ptr);

printf("\nNode deleted from the begining ...\n");

}

}

void last\_delete()

{

struct node \*ptr,\*ptr1;

if(head == NULL)

{

printf("\nlist is empty");

}

else if(head -> next == NULL)

{

head = NULL;

free(head);

printf("\nOnly node of the list deleted ...\n");

}

else

{

ptr = head;

while(ptr->next != NULL)

{

ptr1 = ptr;

ptr = ptr ->next;

}

ptr1->next = NULL;

free(ptr);

printf("\nDeleted Node from the last ...\n");

}

}

void random\_delete()

{

struct node \*ptr,\*ptr1;

int loc,i;

printf("\n Enter the location of the node after which you want to perform deletion \n");

scanf("%d",&loc);

ptr=head;

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

{

ptr1 = ptr;

ptr = ptr->next;

if(ptr == NULL)

{

printf("\nCan't delete");

return;

}

}

ptr1 ->next = ptr ->next;

free(ptr);

printf("\nDeleted node %d ",loc+1);

}

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("item found at location %d ",i+1);

flag=0;

}

else

{

flag=1;

}

i++;

ptr = ptr -> next;

}

if(flag==1)

{

printf("Item not found\n");

}

}

}

void display()

{

struct node \*ptr;

ptr = head;

if(ptr == NULL)

{

printf("Nothing to print");

}

else

{

printf("\nprinting values . . . . .\n");

while (ptr!=NULL)

{

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

ptr = ptr -> next;

}

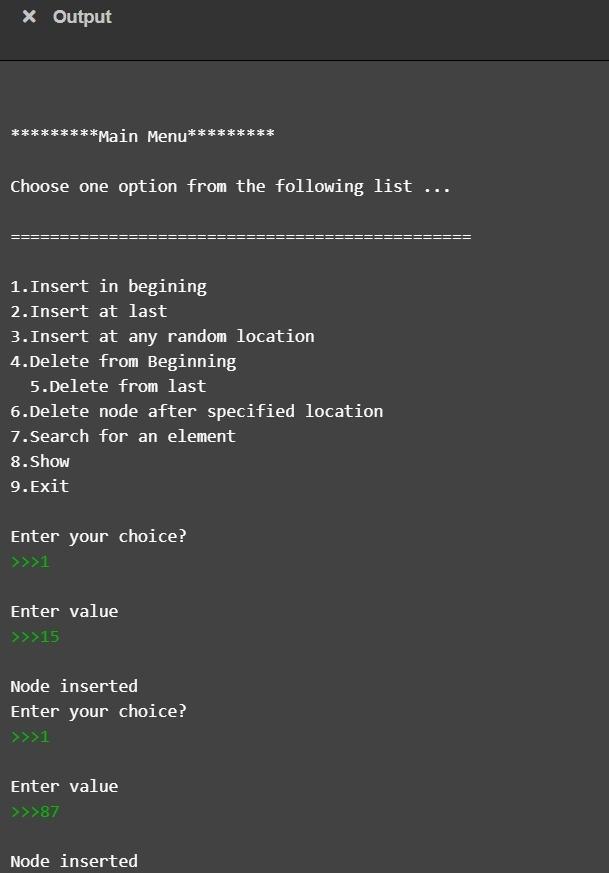
}

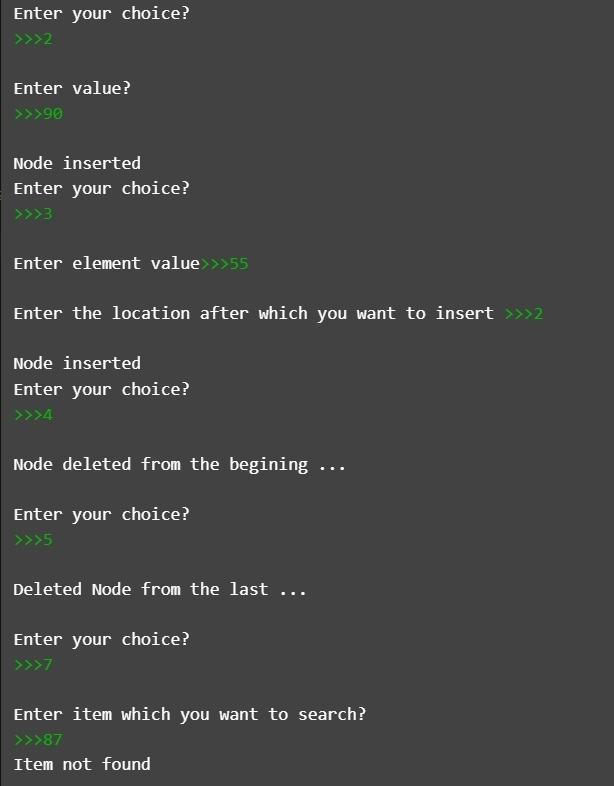
}

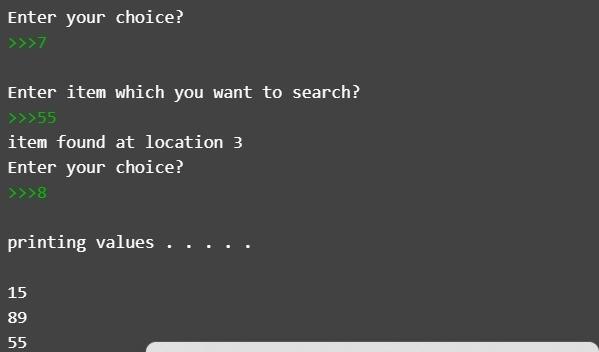
//END OF THE PROGRAM

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

**“SCREENSHOTS OF OUTPUT”**

****

****

****