EXPERIMENT NO. 8

IMPLEMENTATION OF STACK USING LINKED LIST

Aim: Write A Program to Implement Stack Using Linked List.

Theory: A ***Linked List*** is a linear collection of data elements, called ***nodes***, each pointing to the next node by means of a Pointer. It is a Data Structure consisting of a group of nodes which together represent a sequence. Under the simplest form, each node is composed of data and a Reference **(in other words, a link)** to the next node in the sequence.

The **Principal Benefit** of a Linked List over a Conventional Array is that the list elements can easily be inserted or removed without reallocation or reorganization of the entire structure because the data items need not be stored contiguously in memory.

A **Stack** Is an Important Data Structure Which Is Extensively Used in Computer Application**. Implementation of Stack Can Be Done by Using Array as Well as Linked Lists.** It Stores the Element in A Well Ordered Manner. A Stack Is a Linear Data Structure in Which Elements Can Be Added & Removed Only from One End Which Is Called The **TOP.** Hence, A Stack Is Called A **LIFO (Last-In-First-Out)** Data Structure, As The Element That Was Inserted Last Is the First One to Be Taken Out.

Algorithm:

FOR COUNT OPERATION:

1. Start
2. Declare C =0
3. if (Q= =NULL)
4. Display “Stack Does Not Exists”
5. Else (Step 6 To Step 9)
6. Repeat till Step 8 while (Q! =NULL)
7. Increment C by 1
8. Q=Q’s Link Section
9. Display Value of C
10. Stop

FOR DISPLAY OPERATION:

1. Start
2. if (Q= =NULL)
3. Display “Stack Does Not Exists”
4. Else (Step 5 To Step 10)
5. Display “Contents”
6. Move to A New Line
7. Repeat till Step 10 while (Q! =NULL)
8. Display Value of (Q’s Data Section)
9. Move to A New Line
10. Q=Q’s Link Section
11. Stop

FOR POP OPERATION:

1. Start
2. if (Q= =NULL)
3. Display “Stack Empty”
4. Else (Step 5 To Step 6)
5. P=Q’s Link Section
6. Copy P in TOP
7. Stop

FOR PUSH OPERATION:

1. Start
2. Declare Structure Pointer Temp
3. if (Q= =NULL) (Step 4 To Step 9)
4. Display “Creating Stack”.
5. Create A Structure of Memory Requirement Node and Store the Starting Address in Temp
6. Place The Data (Num) in Temp’s Data Section
7. Place “NULL” In Temp’s Link Section
8. Copy Temp in P
9. Copy P in TOP
10. Else (Step 11 To Step 15)

11. Create A Structure of Memory Requirement Node and Store the Starting Address in Temp
12. Place The Data (Num) in Temp's Data Section
13. Place TOP in Temp's Link Section
14. Copy Temp in Q
15. Copy P in TOP
16. Stop

Program:

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

struct node
{
  int data;
  struct node *link;
};

struct node *p;
struct node *top;

void push(struct node *q,int num)
{
  struct node *temp;
  if(q==NULL)
  {
    printf("Creating stack\n");
    temp=(struct node *)malloc(sizeof(struct node));
    temp->data=num;
    temp->link=NULL;
    p=temp;
    top=p;
  }
  else
  {
    temp=(struct node *)malloc(sizeof(struct node));
    temp->data=num;
    temp->link=p;
    p=temp;
    top=p;
  }
 }
```

```
void pop(struct node *q)
{
  if(q==NULL)
  printf("stack does not exist");
  else
  {
    p=q->link;
    top=p;
  }
}

void count(struct node *q)
{
  int c;
  c=0;
  if(q==NULL)
  printf("Link List Does Not Exists");
  else
  {
    while(q!=NULL)
    {
      c++;
      q=q->link;
    }
    printf("Number Of Nodes %d\n",c);
   }
}

void display(struct node *q)
{
  if(q==NULL)
  printf("Link List Does Not Exists\n");
  else
  {
    printf("Contents:\n");
    while(q!=NULL)
    {
      printf("%d\n" ,q->data);
      q=q->link;
    }
  }
}
```

```
int main()
{
  int n,c;
  p=NULL;
  do
  {
    printf("Enter Your Choice:\n");
    printf("1.Create/push\t");
    printf("\t2.pop\t");
    printf("\t3.Count\t");
    printf("\t4.Display\t");
    printf("\t5.Exit\n");
    scanf("%d",&c);

    switch(c)
    {
      case 1:
      printf("Enter A Value:\n");
      scanf("%d",&n);
      push(p,n);
      break;
      case 2:
      pop(p);
      break;
      case 3:
      count(p);
      break;
      case 4:
      display(p);
      break;
    }
  }
  while(c!=5);
  return 0;
}
```

Output:-

```
Enter Your Choice:
1.Create/push          2.pop          3.Count        4.Display
5.Exit
3
Link List Does Not ExistsEnter Your Choice:
1.Create/push          2.pop          3.Count        4.Display
5.Exit
1
Enter A Value:
17
Creating stack
Enter Your Choice:
1.Create/push          2.pop          3.Count        4.Display
5.Exit
1
Enter A Value:
22
Enter Your Choice:
1.Create/push          2.pop          3.Count        4.Display
5.Exit
4_
```

```
1.Create/push          2.pop          3.Count        4.Display
5.Exit
4
Contents:
22
17
Enter Your Choice:
1.Create/push          2.pop          3.Count        4.Display
5.Exit
2
Enter Your Choice:
1.Create/push          2.pop          3.Count        4.Display
5.Exit
1
Enter A Value:
56
Enter Your Choice:
1.Create/push          2.pop          3.Count        4.Display
5.Exit
3
Number Of Nodes 2
Enter Your Choice:
1.Create/push          2.pop          3.Count        4.Display
5.Exit
4_
```

```
1.Create/push          2.pop          3.Count          4.Display
5.Exit
2
Enter Your Choice:
1.Create/push          2.pop          3.Count          4.Display
5.Exit
1
Enter A Value:
56
Enter Your Choice:
1.Create/push          2.pop          3.Count          4.Display
5.Exit
3
Number Of Nodes 2
Enter Your Choice:
1.Create/push          2.pop          3.Count          4.Display
5.Exit
4
Contents:
56
17
Enter Your Choice:
1.Create/push          2.pop          3.Count          4.Display
5.Exit
5_
```

Conclusion:- The experiment successfully demonstrates the implementation of a stack using a linked list, illustrating the Last-In-First-Out (LIFO) principle. It effectively showcases the fundamental operations of push and pop, highlighting how elements are added and removed from the top. This implementation enhances understanding of stack data structure functionality and its dynamic memory management using linked lists.