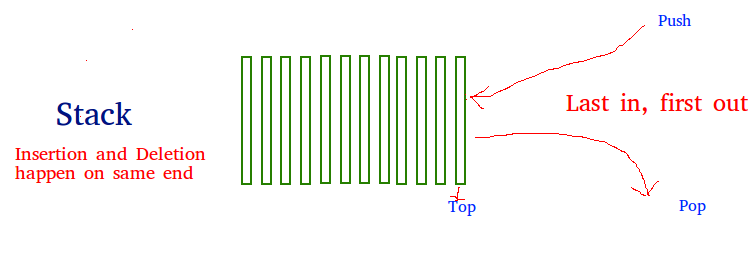
**STACK USING LINKED LISTS**

**Aim:** To implement stack using Linked list.

**Theorem:** Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).



There are many real-life examples of a stack. Consider an example of plates stacked over one another in the canteen. The plate which is at the top is the first one to be removed, i.e. the plate which has been placed at the bottommost position remains in the stack for the longest period of time. So, it can be simply seen to follow LIFO(Last In First Out)/FILO(First In Last Out) order.

**Basic Operations**

Stack operations may involve initializing the stack, using it and then de-initializing it. Apart from these basic stuffs, a stack is used for the following two primary operations −

* **push()** − Pushing (storing) an element on the stack.
* **pop()** − Removing (accessing) an element from the stack.

When data is PUSHed onto stack.

To use a stack efficiently, we need to check the status of stack as well. For the same purpose, the following functionality is added to stacks −

* **peek()** − get the top data element of the stack, without removing it.
* **isFull()** − check if stack is full.
* **isEmpty()** − check if stack is empty.

At all times, we maintain a pointer to the last PUSHed data on the stack. As this pointer always represents the top of the stack, hence named **top**. The **top** pointer provides top value of the stack without actually removing it.

**Program:**

#include<stdio.h> //preprocessor directives

#include<stdlib.h>

void push();

void pop(); //function declaration

void display();

struct node{ //creating new node using structures

int val;

struct node \*next;

};

struct node \*head; //creating head node

void push() //function for push operation

{

int val;

struct node \*ptr=(struct node \*)malloc(sizeof(struct node)); //dynamic memory allocation for node

if(ptr==NULL)

printf("cant push");

else

{

printf("enter the val");

scanf("%d",&val);

if(head==NULL)

{

ptr->val=val; //here we asign value to the data part of node

ptr->next=NULL; //we declare the address part of the node as null

head=ptr; //asigning ptr to head

}

else

{

ptr->val=val;

ptr->next=head;

head=ptr;

}

printf("the item is pushed");

}

}

void pop() //function for pop operation

{

int item;

struct node \*ptr;

if(head==NULL)

printf("stack is empty");

else

{

item=head->val; //here we tell the compiler that item is the data part of node

ptr=head; //declaring head to ptr

head=head->next; //asigning head as its link

free(ptr); //deleting ptr

printf("item popped");

}

}

void display()

{

int i;

struct node \*ptr;

ptr=head;

if(ptr==NULL)

printf("stack is empty");

else

{

printf("printing elements");

while(ptr!=NULL) //while loop to display all elements

{

printf("%d",ptr->val); //prints data part of the node

ptr=ptr->next; //assignining next node to ptr

}

}

}

int main()

{

int choice;

while(1)

{

printf("1.Push\n2.Pop\n3.display\n4.exit\n");

scanf("%d",&choice);

switch(choice) //switch statement

{

case 1:push();

break;

case 2:pop();

break;

case 3:display();

break;

case 4:exit(0);

}

}

}

**Algorithm:**

**Push Operation**

STEP 1:I nitialise a node

STEP 2: Update the value of that node by data i.e. **node->data = data**

STEP 3: Now link this node to the top of the linked list

STEP 4: And update top pointer to the current node

**Pop Operation**

STEP 1: First Check whether there is any node present in the linked list or not, if not then return

STEP 2: Otherwise make pointer let say **temp** to the top node and move forward the top node by 1 step

STEP 3:Now free this temp node

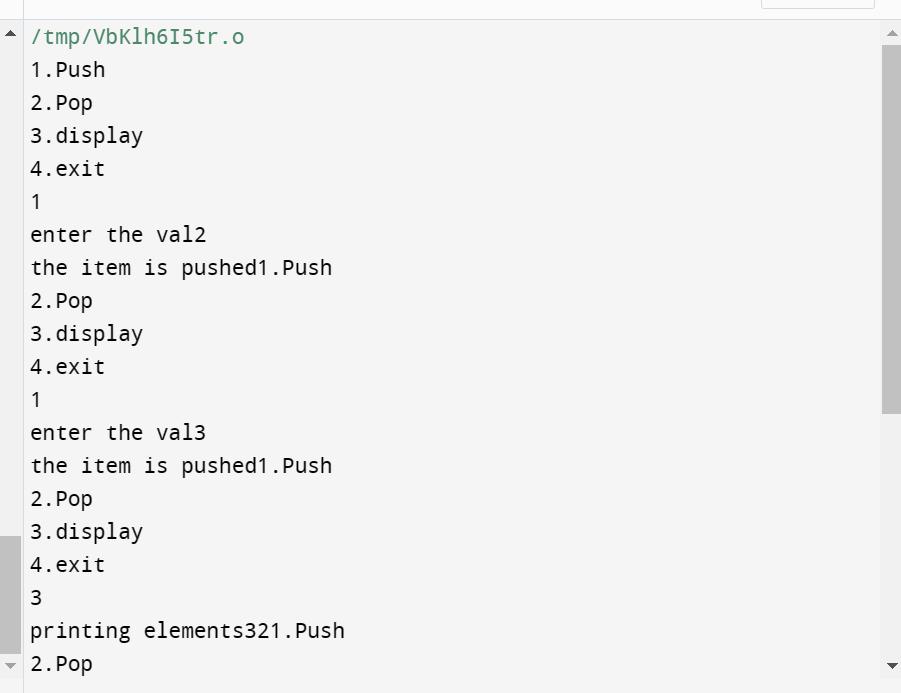
**Display operation**

STEP 1: Take a **temp**node and initialize it with top pointer

STEP 2:Now start traversing temp till it encounters NULL

STEP 3: Simultaneously print the value of the temp node

**OUTPUT:**

****

