**EXPT NO:** 2  **DATE:**

**STACKS**

**AIM:**

**1) Write a C program to implement basic stack operations using an array.**

a) Push

b) Pop

c) Display Top element

d) Display all the elements

e) Reverse the stack using another stack.

**2) Write a C program to implement basic stack operations using Linked List**

a) Push

b) Pop

c) Display Top element

d) Display all the elements

e) Reverse the stack using another stack.

**3) Write a C program to perform the following**

a) Conversion of Infix to postfix expression and evaluation

b) Reversing a string using a stack

c) To check the nesting of parenthesis using stack.

**THEORY:**

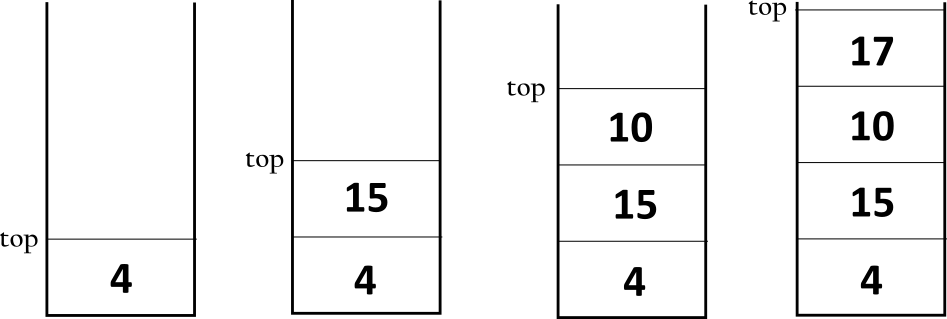
Stack is a linear list in which insertion and deletions are allowed only t one end, called the top of the stack. We can see the examples of stack in our daily life like stack of trays in a cafeteria, stack of books or stack of bills pierced on to a small metal rod in a hotel. In all this cases we can see that any object can be removed or added only at the top.

The insertion and deletions are given special names in the case of stacks. The push operation inserts an element onto the stack and the pop operation deletes the element which is on the top of the stack.

The behaviour of a stack is like LAST IN FIRST OUT, so it called as LIFO (Last In First Out) data structure.

Before pushing an element, we must check whether there is space in the stack or no. if there is no enough space on the stack then it is said to be in Overflow state and the new element cannot be pushed on to the stack. Similarly, when a pop operation is attempted on an empty stack then it is said to be in Underflow state.

**PUSH** 4,15,10,17



**POP OPERATION:** 17,10,15,4

**Array implementation of Stack.**

We take a one-dimensional array to hold the elements of the stack. In an array, elements can be added or deleted at any place but since we are implementing stack, we have to permit insertions and deletions at the top of the stack only. So, we take a variable Top which keeps a track of index value of the top variable.

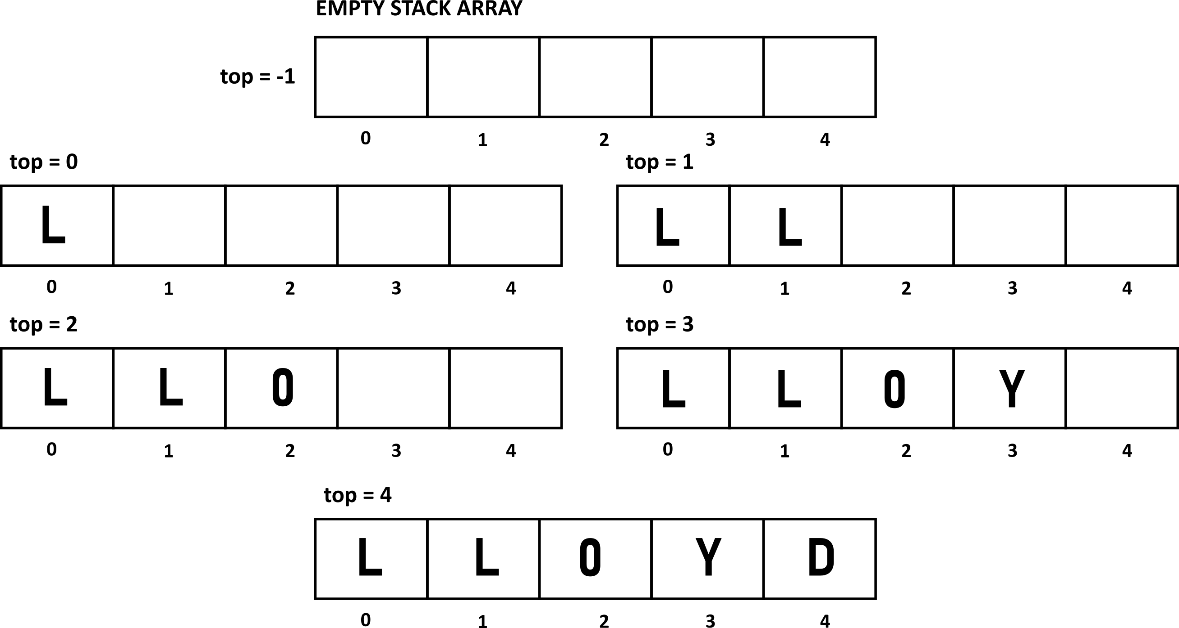
Initially when the stack is empty, value of the top is initialized to -1. For push operations, first the value of the top is increased by 1 and then the new element is pushed at the position. For the pop operation, the first element at the position of top is popped and then top is decreased by 1.

For the functions to push and pop we have to check the overflow and underflow. If MAX is the size of the array, then the stack will become full when the top becomes equal to MAX-1.

**push()** :- Pushes the items from the stack.

**pop()** :- Pops an item from the stack.

**PUSH:** L, L, O, Y, D MAX=5

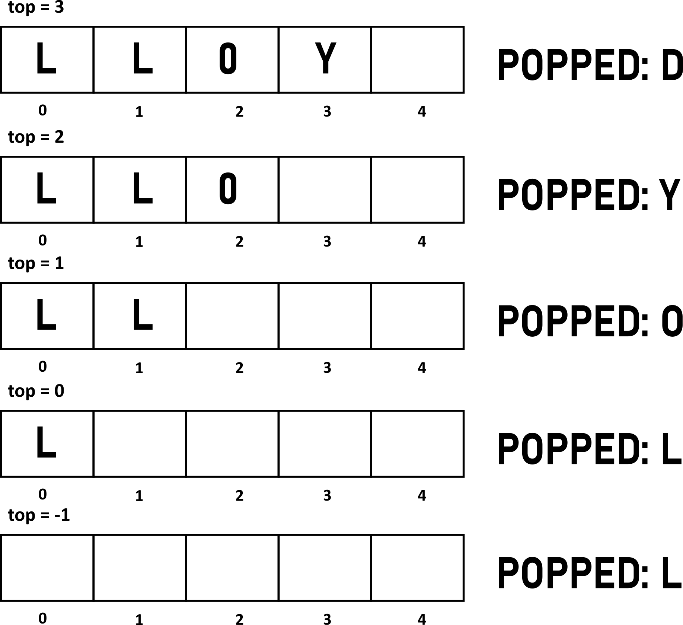






**POP:**



**peek() :-** Displays the top element of the stack.

**display() :-** Displays all the elements of the stack.

**reverse()** :- Used to read each element of stack 1 and pass it to push2() function

**push2()** :- Used to push elements into stack 2

**display2()** :- used to display all the elements of stack 2(reverse of stack1)

**Linked List implementation of Stack.**

When the size of the array is not known in advanced, it is better to implement it as a linked list. In this case stack will not overflow till there is space available for dynamic memory allocation.

struct node{

int info;

struct node \*link.

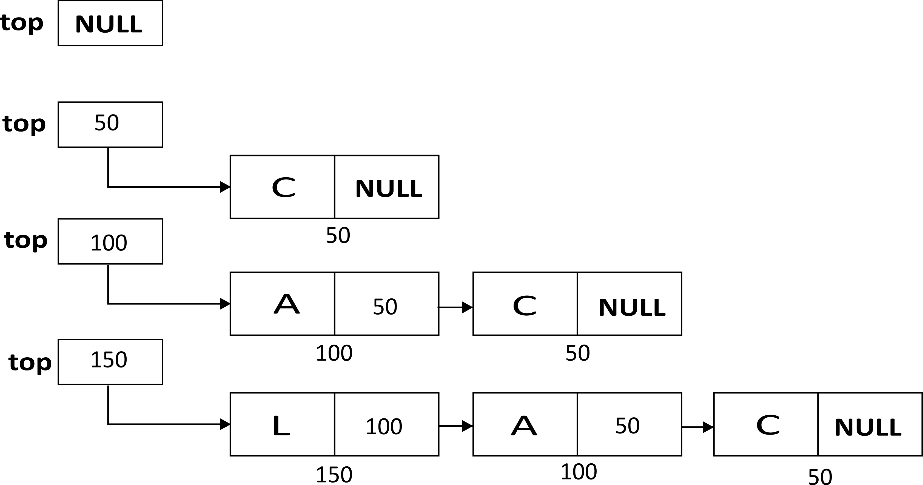
} ;

For push operation, a node is inserted at the beginning of the list. For pop operation, first node of the list will be deleted. A pointer Top is used to point the top node of the stack.

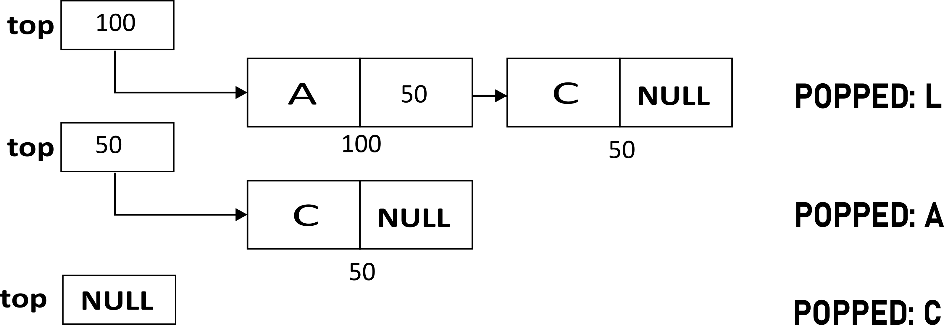
For pushing an element on stack we follow the procedure of inserting in the beginning of the linked list. The function push() would ne similar to the function addBeg() of singly linked list. The stack will overflow only when there is no space left for dynamic memory allocation and this and this case call to function malloc() will return NULL. So inside the function push(), we will check for this overflow condition.

For pop operation, we will delete the first element of the linked list. The underflow condition will arise when the linked list is empty, i.e. when top is equal to NULL. So inside the pop function we check for the underflow condition.

**PUSH:** C,A,L

****

**POP:**

****

**Steps involved to convert Infix expression to postfix expression.**

1) Scan the symbol of array infix one by one from left to right.

a) If symbol is an operand

add it to the array postfix

b) If the symbol is left parenthesis ‘(‘

Push it onto the stack

c) If the symbol is right parenthesis ‘)’

Pop all the operators from the stack upto the first left parenthesis and add this operators to array postfix. Discard both left and right parenthesis.

d) If symbol is operator

pop the operators which have precedence greater than or equal to the precedence of the symbol operator, and add those popped operators to array postfix. Push the scanned operators on the stack. (Assume that the left parenthesis has the least precedence).

2) After all the symbols of array infix have been scanned, pop all the operators remaining on the stack and add them in the array postfix.

|  |  |
| --- | --- |
| **Operator** | **Priority** |
| ^ Exponentiation | 3 |
| \*(Multiplication) and / Divisions | 2 |
| + Add and - Subtract | 1 |

Advantage of using Postfix Expressions

Whenever an infix expression consists of more than one operator, the precedence riles (BODMAS) should be applied to decide which operator and operands associated with that operator are evaluated first. As compared to postfix expressions, which is much easier to work with or evaluate. In a postfix expression, operands appear first, before the operators, there is no need for operator precedence and other rules

**Checking the validity of an expression containing nested parentheses**

An expression will be valid if it satisfies these two conditions

1) The total number of left parentheses should be equal to the total number of right parentheses in the expression.

2) For every right parenthesis there should be a left parenthesis of the same type

Examples

[A+B-C\*D}(C+D INVALID

(A+B)(C+D) VALID

Procedure to implement

1) Initially take an empty stack.

2) Scan the symbols of the expression from right to left.

3) Push all the parenthesis onto the stack

4) After scanning the entire expression, start popping out the elements one by one

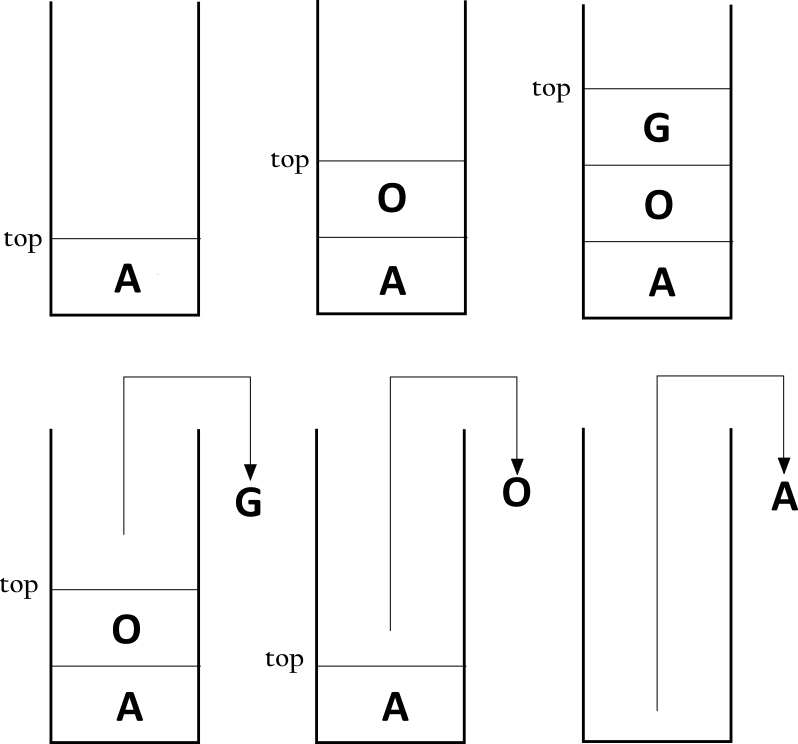
5) If left parenthesis is popped then increment the counter 1 and else if the right parenthesis is popped then increment the counter 2.

6) If count 1 is equal to count 2 then the expression is valid or else its an invalid statement.

**Reversing a string using a stack**

We can reverse a string by pushing each character of the string on the stack. After the whole string is pushed on the stack. We can start popping the characters from the stack and get the reversed string.

**PUSH:** A, O, G



**AFTER POPPPING:** G, O, A

**PSEUDOCODE:** Array implementation of Stack

**int isFull()**

1. if top ==MAX-1

1.return 1

2.else

1. return 0

**int isEmpty()**

1. if top == -1

1.return 1

2.else

1. return 0

**void push()**

1. declare int x

2. Input x

3. if(isFull())

    1. print “STACK OVERFLOW”

    2. return

4.  top=top+1

5. array[top]=x

**Void display()**

1. declare int i

2. if isEmpty()

1.print “STACK EMPTY”

2. return

3. i=top

4. if i >= 0

1. print array[i]

**Void peek()**

1. if(isEmpty())

    1.print “STACK UNDERFLOW”

    2. Return

2. Print array[top]

**Void pop()**

1. declare int x

2. if(isEmpty())

    1. print “STACK UNDERFLOW”

    2. return

3. x=array[top]

4. top=top-1

5. Print x

**Void reverse()**

1. declare int x

2. for(i=top;i>=0;i--)

    1. push2(array[i])

**Void push2(int x)**

1. top1++

2. array1[top1]=x

**Void display2()**

1. declare int i

2. if(isEmpty())

    1. print “STACK IS EMPTY”

    2. return

3. for(i=top1;i>=0;i--)

1.print array[i]

Linked List implementation of Stack

**Void display(int flag)**

1. if IsEmpty()

1. print “stack empty”

2. return

2. declare node \*p

3.if flag==0

1.p=top

4. else

1.p=top2

5.while p!=NULL

1.print p->data

2.p=p->link

.

**Void push(int x, int flag)**

1. declare node \*temp

2.tem0=(struct node\*)malloc(sizeof(struct node))

3. if temp==NULL

1.print “Stack Overflow”

2.Return

4. temp->data=x

5. if flag==0

1.Temp->link=top

2.top=temp

6. else

1.temp->link=top2

2.top2=temp

**Void pop()**

1. declare node \*temp

2. temp=top

3. if isEmpty()

1. print “stack underflow”

2.return

4. print top->data

5. top=top->link

6. free temp

.

**Int IsEmpty()**

1. if top == NULL

1. return 1

2. else

Return 0;

**Void reverse()**

1. declare node \*p

2. p=top

3. while p!=NULL

1. push(p->data,1)

2. p=p->link

4. if top2==NULL

Print “stack empty”

2.return

5.display(1)

.

Conversion of Infix expression to Postfix Expression

**Void push(long int symbol)**

1. if top==MAX-1

1.print “Stack Overflow”

2. exit(1)

2.stack[++top]=symbol

.

**Long int pop()**

1. if top==-1

1.print “Stack underflow”

2. exit(1)

2.return(stack[top--]);

.

**int isEmpty()**

1. if top==-1

1.return 1

2.else

1.return 0

.

**Int instack\_priority(char symbol)**

1. switch symbol

1. case ‘(‘

1.Return 0

2. case ‘+’

3. case ‘-‘

1. return 1

4. case ‘\*’

5. case ‘/’

6. case ‘%’

1.Return 2

7. case ‘^’

Return 3

.

**Int symbol\_priority(char symbol)**

1. switch symbol

1. case ‘(‘

1.return 0

2. case ‘+’

3. case ‘-‘

1. return 1

4. case ‘\*’

5. case ‘/’

6. case ‘%’

1.Return 2

7. case ‘^’

Return 4

.

**Int white\_space(char symbol)**

1. if(symbol==BLANK||symbol==TAB)

1. return 1

2. else

1.return 0 .

**Void infix\_to\_postfix()**

1. declare int i,p=0

2. declare char next,symbol

3. for(i=0; i<strlen(infix); i++)

1.symbol=infix[i]

2.if(!white\_space(symbol))

1.switch(symbl)

1. case ‘(‘

1.push(symbol)

2. break

2. case ‘)’

While(next=pop()!=’)’)

1.postfix[p++]=next

3. break

3. case ‘+’

4. case ‘-‘

5. case ‘\*’

6. case ‘/’

7. case ‘%’

8. case ‘^’

1. while((top!=-1)&&instack\_priority(stack[top]) >= symbol\_priority(symbol))

1.postfix[p++]=pop()

2.push(symbol)

3.break

9. default

1.postfix[p++]=symbol

4.while(!isEmpty())

1,postfix[p++]=pop()

5. postfix[p]=’\0’

**long eval\_post()**

1. declare long int a,b,temp,result

2. declare int i

3. for(i=0;i<strlen(postfix);i++)

1. if postfix[i]<=’9’&&postfix[i]>=’0’

1.push(postfix[i]-‘0’)

2. else

1. a=pop()

2. b=pop()

3. switch(postfix[i])

1. case ‘+’

1.Temp=b+a

2. break

2. case ‘-‘

1. temp=b-a

2.break

3. case ‘\*‘

1. temp=b\*a

2.break

4. case ‘/‘

1. temp=b/a

2.break

5. case ‘%‘

1. temp=b%a

2.break

6. case ‘^‘

1. temp=pow(b,a)

2.break

3. push(temp)

4. result=pop()

5. return result

Parenthesis Checking

**Void display()**

1. if(isEmpty())

    1. print “STACK EMPTYT”

    2. return

2. declare int count1=0,count2=0

3. declare and initialize p=top

4. while(p!=NULL)

    1.if(p->letter==’}’||p->letter==’)’)

    2. count1=count1+1

    3. if(p->letter==’{‘||p->letter==’(‘)

    4. count2=count2+1

    5. p=p->link

5. if(count1==count2)

6. print “VALID EXPRESSION”

7.else

8. print ”INVALID EXPRESSION”

**void StrINPUT()**

1. gets(temp)

.

**int isEmpty()**

1. declare struct node \*p

2. if(top==NULL)

    1. return 1

3. else

    1. return 0

**Void push(char x)**

1. declare struct node \*temp

2. temp=(struct node\*)malloc(sizeof(struct node))

3. if(temp==NULL)

    1. print “STACK OVERFLOW”

    2. return

4. temp->letter=x

5. temp->link=top

6. top=temp

**int main()**

1. declare int i

2. strInput()

3. for(i=0;temp[i]!='\0';i++)

1.push(temp[i]);

4.display();

Reversing a string

**Void push(char x)**

1. declare node\*temp

2. temp=(struct node\*)malloc(sizeof(struct node))

3. if(temp==NULL)

    1. print “STACK OVERFLOW”

    2. return

4. temp->letter=x

5. temp->link=top

6. top=temp

.

**void StrINPUT()**

1. gets(temp)

.

**Int isEmpty()**

1.declare struct node\*p

2. if(top==NULL)

    1. return 1

 3. else

    1. return 0

.

**int main()**

1. declare int i

2. strInput()

3. for(i=0;temp[i]!='\0';i++)

1.push(temp[i]);

4.display();

**Void display()**

1. if(isEmpty())

    1. print “STACK EMPTY”

    2. return

2. declare struct node\*p

3. p=top

4. while(p!=NULL)

    1. output p->letter

    2. p=p->link

**SOURCE CODE:**

Stack with Array Implementation

#include<stdio.h>

#define MAX 6

int array[MAX];

int array1[MAX];

int top=-1;

int top1=-1;

int isFull()

{

if(top==MAX-1)

return 1;

else

return 0;

}

int isEmpty()

{

if(top==-1)

return 1;

else

return 0;

}

void display()

{

int i;

if(isEmpty())

{

printf("STACK IS EMPTY\n");

return;

}

printf("STACK ELEMENTS:\n");

for(i=top;i>=0;i--)

printf("%d ",array[i]);

}

void push()

{

int x;

printf("ENTER ELEMENT: ");

scanf("%d",&x);

if(isFull())

{

printf("STACK OVERFLOW\n");

return;

}

top++;

array[top]=x;

}

void pop()

{

int x;

if(isEmpty())

{

printf("STACK UNDERFLOW\n");

return;

}

x=array[top];

top--;

printf("ITEM POPPED IS: %d\n",x);

}

void peek()

{

if(isEmpty())

{

printf("STACK UNDERFLOW\n");

return;

}

printf("TOP ELEMENT: %d\n",array[top]);

}

void display2()

{

int i;

if(isEmpty())

{

printf("STACK IS EMPTY\n");

return;

}

for(i=top1;i>=0;i--)

{

printf("%d ",array1[i]);

}

}

void push2(int x)

{

top1++;

array1[top1]=x;

}

void reverse()

{

int i;

for(i=top;i>=0;i--)

{

push2(array[i]);

}

}

int main()

{ int c;

while(1)

{

printf("\n\n\*\*\*\*\*\*\*\*\*\*\*\*STACKS\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("ENTER THE CORRESSPONDING CHOICE\n");

printf("PUSH...........................1\n");

printf("POP............................2\n");

printf("DISPLAY THE TOP ELEMENT........3\n");

printf("DISPLAY ALL ELEMENTS...........4\n");

printf("REVERSE THE STACK..............5\n");

printf("EXIT...........................0\n");

printf("ENTER YOUR CHOICE: ");

scanf("%d",&c);

printf("\n\n");

switch(c)

{

case 1:

push();

break;

case 2:

pop();

break;

case 3:

peek();

break;

case 4:

display();

break;

case 5:

printf("\nBEFORE REVERSING: \n");

display();

reverse();

printf("\nAFTER REVERSING: \n");

display2();

break;

case 0:

return 0;

break;

default:

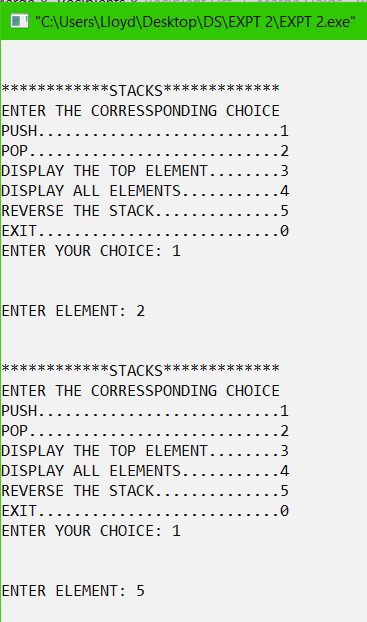
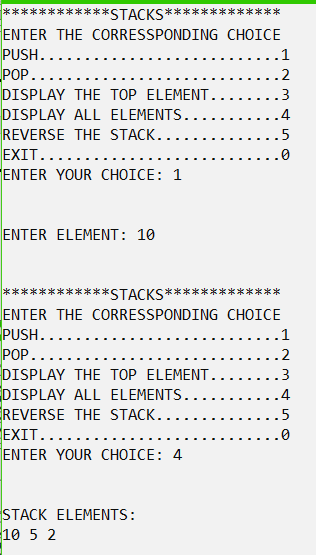
printf("INVALID CHOICE\n");

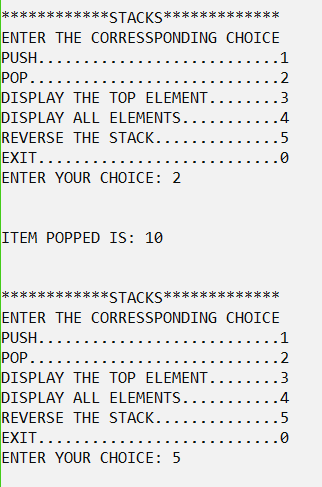
}

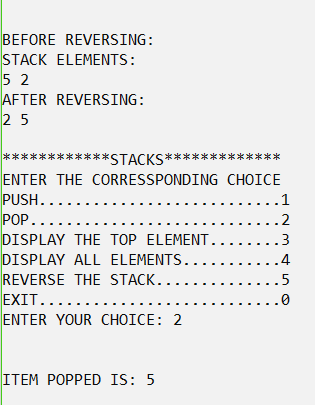
}

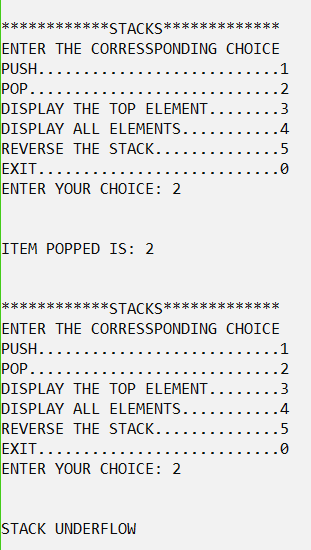
}

**OUTPUT:**

****

****

****

****

**SOURCE CODE:**

Stack with Linked List Implementation

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node \*link;

}\*top=NULL,\*top2=NULL;

int isEmpty(){

if(top==NULL)

return 1;

else

return 0;

}

void push(int x, int flag){

struct node \*temp;

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

if(temp==NULL)

{

printf("STACK OVERFLOW\n");

return;

}

temp->data=x;

if(flag==0)

{

temp->link=top;

top=temp;

}

else

{

temp->link=top2;

top2=temp;

}

}

void pop(){

struct node \*temp;

temp=top;

if(isEmpty())

{

printf("STACK UNDERFLOW\n");

return;

}

printf("POPPED: %d",top->data);

top=top->link;

free(temp);

}

void display(int flag){

if(isEmpty())

{

printf("STACK EMPTY\n");

return;

}

struct node \*p;

if(flag==0)

p=top;

else

p=top2;

while(p!=NULL)

{

printf("%d ",p->data);

p=p->link;

}

}

void reverse()

{

struct node \*p=top;

while(p!=NULL)

{

push(p->data,1);

p=p->link;

}

if(top2==NULL)

{

printf("STACK EMPTY\n");

return;

}

display(1);

}

int main()

{

int choice,e;

while(1)

{

printf("\n\nENTER THE CORRESPONDING NUMBER TO ACCESS IT:\n");

printf("1: PUSH\n");

printf("2: POP\n");

printf("3: DISPLAY TOP ELEMENT\n");

printf("4: DISPLAY ALL ELEMENTS\n");

printf("5: REVERSE THE STACK\n");

printf("\nENTER YOU CHOICE: ");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("ENTER THE ELEMENT\n");

scanf("%d",&e);

push(e,0);

break;

case 2:

pop();

break;

case 3:

printf("TOP ITEM: %d\n",top->data);

break;

case 4:

display(0);

break;

case 5:

reverse();

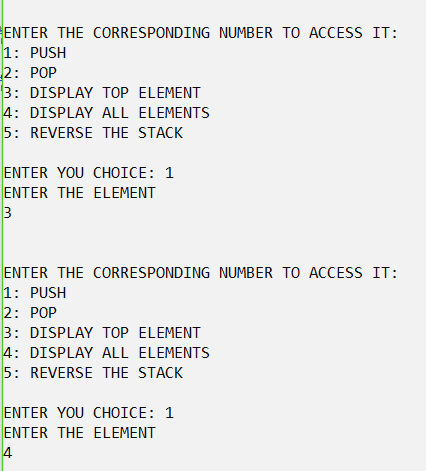
break;

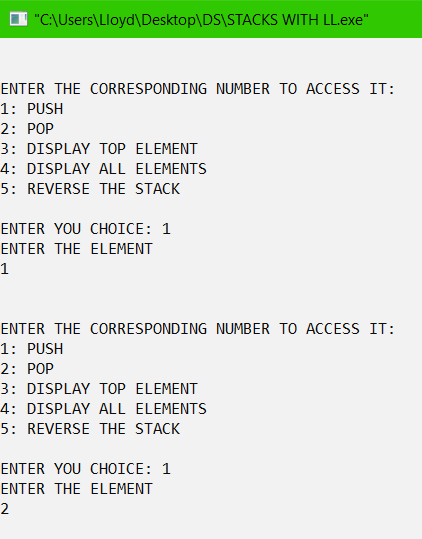
}

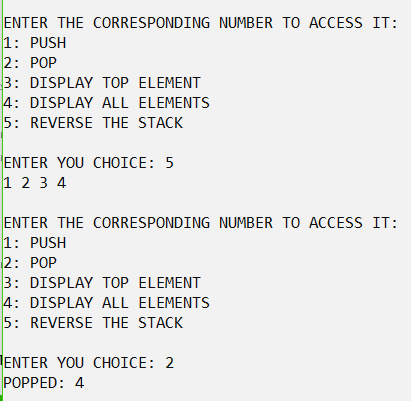
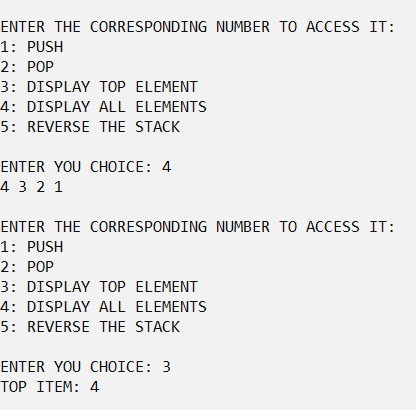
}

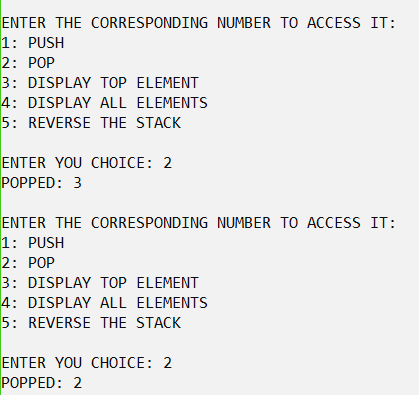
}

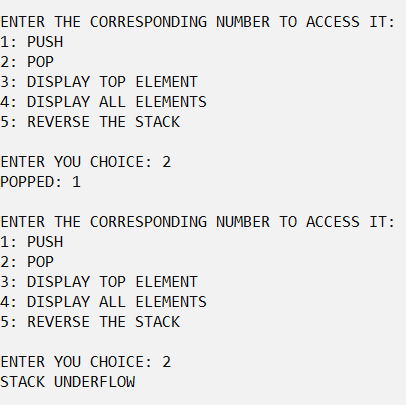
**OUTPUT:**

****

****

****

****

****

Infix to Postfix and its evaluation

#include<stdio.h>

#include<string.h>

#include<math.h>

#include<stdlib.h>

#define BLANK ' '

#define TAB '\t'

#define MAX 50

void push(long int symbol);

long int pop();

void infix\_to\_postfix();

long int eval\_post();

int instack\_priority(char symbol);

int symbol\_priority(char symbol);

int white\_space(char symbol);

char infix[MAX], postfix[MAX];

long int stack[MAX];

int top=-1;

int main()

long int value;

printf("ENTER THE INFIX EXPRESSION\n");

gets(infix);

infix\_to\_postfix();

printf("Postfix %s\n",postfix);

value=eval\_post();

printf("Value of expression : %ld\n",value);

return 0;

}

void push(long int symbol)

{

if(top==MAX-1)

{

printf("Stack Overflow\n");

exit(1);

}

stack[++top]=symbol;

}

long int pop()

{

if(top==-1)

{

printf("Stack Underflow\n");

exit(1);

}

return(stack[top--]);

}

int isEmpty()

{

if(top==-1)

return 1;

else

return 0;

}

int instack\_priority(char symbol)

{

switch(symbol)

{

case '(':

return 0;

case '+':

case '-':

return 1;

case '\*':

case '/':

case '%':

return 2;

case '^':

return 3;

}

}

int symbol\_priority(char symbol)

{

switch(symbol)

{

case '(':

return 0;

case '+':

case '-':

return 1;

case '\*':

case '/':

case '%':

return 2;

case '^':

return 4;

}

}

void infix\_to\_postfix()

{

int i,p=0;

char next;

char symbol;

for(i=0;i<strlen(infix);i++)

{

symbol=infix[i];

if(!white\_space(symbol))

{

switch(symbol)

{

case '(':

push(symbol);

break;

case ')':

while((next=pop())!='(')

postfix[p++]=next;

break;

case '+':

case '-':

case '\*':

case '/':

case '%':

case '^':

while((top!=-1)&&instack\_priority(stack[top])>=symbol\_priority(symbol))

postfix[p++]=pop();

push(symbol);

break;

default:

postfix[p++]=symbol;

}

}

}

while(!isEmpty())

postfix[p++]=pop();

postfix[p]='\0';

}

int white\_space(char symbol)

{

if(symbol==BLANK||symbol==TAB)

return 1;

else

return 0;

}

long int eval\_post()

{

long int a,b,temp,result;

int i;

for(i=0;i<strlen(postfix);i++)

{

if(postfix[i]<='9'&&postfix[i]>='0')

push(postfix[i]-'0');

else{

a=pop();

b=pop();

switch(postfix[i])

{

case '+':

temp=b+a;

break;

case '-':

temp=b-a;

break;

case '\*':

temp=b\*a;

break;

case '/':

temp=b/a;

break;

case '%':

temp=b%a;

break;

case '^':

temp=pow(b,a);

}

push(temp);

}

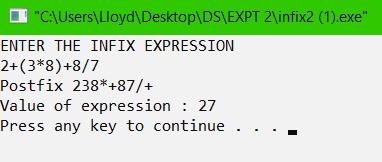
}

result=pop();

return result;

}

**OUTPUT:**

****

Reversing a String using a stack

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

char temp[50];

struct node{

char letter;

struct node \*link;

}\*top=NULL;

void StrINPUT()

{

printf("ENTER A STRING\n");

gets(temp);

}

int isEmpty(){

struct node \*p;

if(top==NULL)

return 1;

else

return 0;

}

void push(char x){

struct node \*temp;

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

if(temp==NULL)

{

printf("STACK OVERFLOW\n");

return;

}

temp->letter=x;

temp->link=top;

top=temp;

}

void display(){

if(isEmpty())

{

printf("STACK EMPTY\n");

return;

}

struct node \*p;

p=top;

while(p!=NULL)

{

printf("%c ",p->letter);

p=p->link;

}

}

int main()

{

int i;

StrINPUT();

for(i=0;temp[i]!='\0';i++)

{

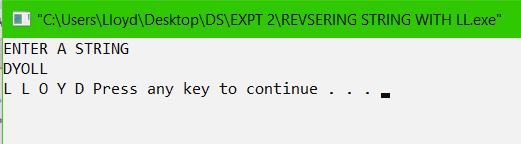
push(temp[i]);

}

display();

}

**OUTPUT:**

****

Parenthesis checking using Linked List

Parenthesis Check using stack

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

char temp[50];

struct node{

char letter;

struct node \*link;

}\*top=NULL;

void StrINPUT()

{

printf("ENTER A EXPRESSION\n");

gets(temp);

}

int isEmpty(){

struct node \*p;

if(top==NULL)

return 1;

else

return 0;

}

void push(char x){

struct node \*temp;

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

if(temp==NULL)

{

printf("STACK OVERFLOW\n");

return;

}

temp->letter=x;

temp->link=top;

top=temp;

}

void display(){

if(isEmpty())

{

printf("STACK EMPTY\n");

return;

}

struct node \*p;

int count1=0,count2=0;

p=top;

while(p!=NULL)

{

if(p->letter=='}'||p->letter==')')

count1++;

if(p->letter=='{'||p->letter=='(')

count2++;

p=p->link;

}

if(count1==count2)

printf("VALID EXPRESSION\n");

else

printf("INVALID EXPRESSION\n");

}

int main()

{

int i;

StrINPUT();

for(i=0;temp[i]!='\0';i++)

{

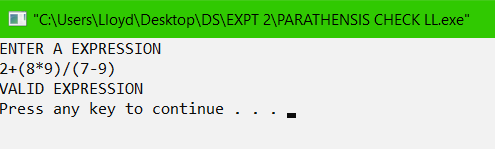
push(temp[i]);

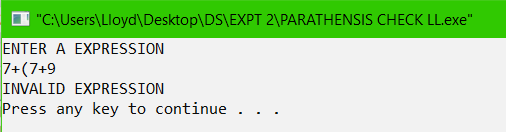
}

display();

}

**OUTPUT:**



****

**CONCLUSIONS:**

The given problem statements were successfully compiled and executed.

A stack is one of the simplest data structure which can be easily implemented in any program. It’s Last in First Out (LIFO) approach makes it useful in many applications.

**Findings**

Linked List implementation of stacks may be an optimum choice over array implementation since array is static in nature and hence has a fixed size.

**Compilation Time**

|  |  |
| --- | --- |
| Array Implementation of stack | 0.17 secs |
| Linked List Implementation of stack | 0.18 secs |
| Infix to Postfix | 0.22 secs |
| String Reversal | 0.17 secs |
| Parenthesis Checking | 0.17 secs |