**STACK**

Stack is a linear data structure in which insertions and deletions are performed using one end called top end. Stack follows LIFO (Last In First Out) or FILO (First In Last Out) order.

It is named stack as it behaves like a real-world stack, for example: a deck of cards or a pile of plates in cafeteria.







**Basic Operations on Stack :**

* **push()** − Inserting (storing) an element into the stack.
* **pop()** − Removing (deleting) an element from the stack.
* **peek()** − Get the top data element of the stack, without removing it.
* **isFull()** − Check if stack is full.
* **isEmpty()** − Check if stack is empty.

**Note:**

* push() & pop() operations can be performed from only one end called the ‘top end’.

Because other end is closed.

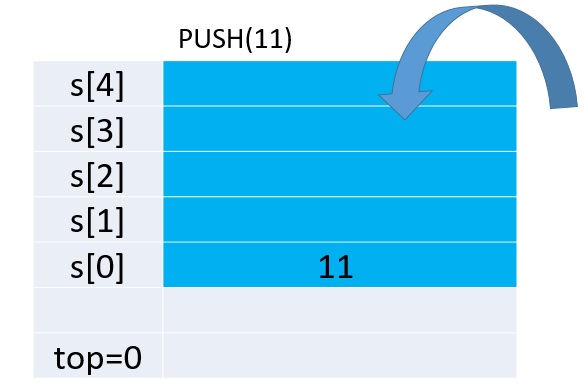
* For each insert / push operation stack top is incremented (by one)
* For each delete / pop operation stack top is decremented (by one)

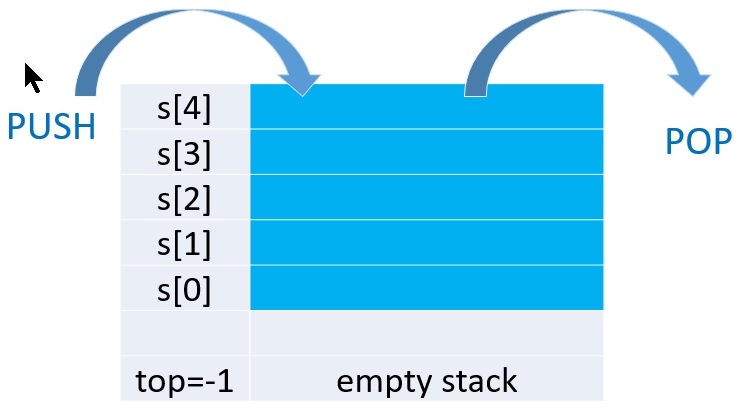
**Stack Implementation:**

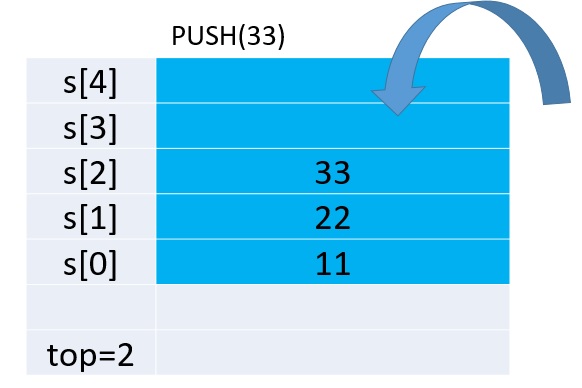
Stack can be implemented in two ways

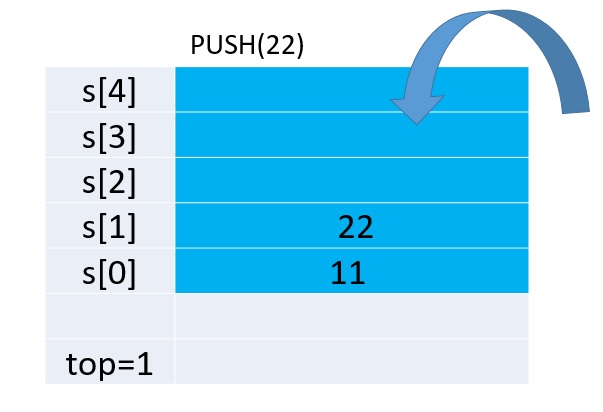
1. Array method 2) Linked list method

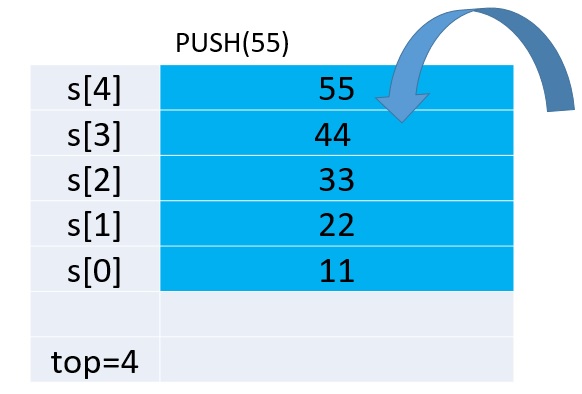
The below diagrams represent push & pop operations on stack using **Arrays** and the position of stack pointer (stack top)

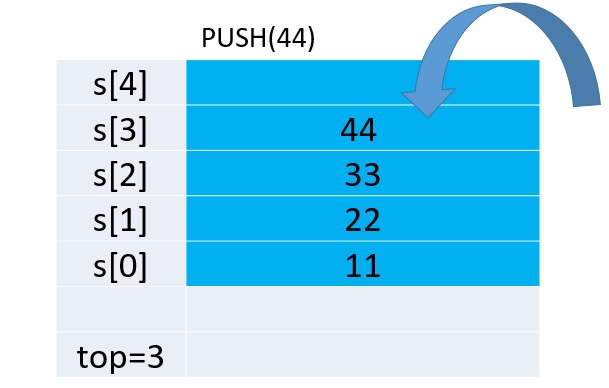


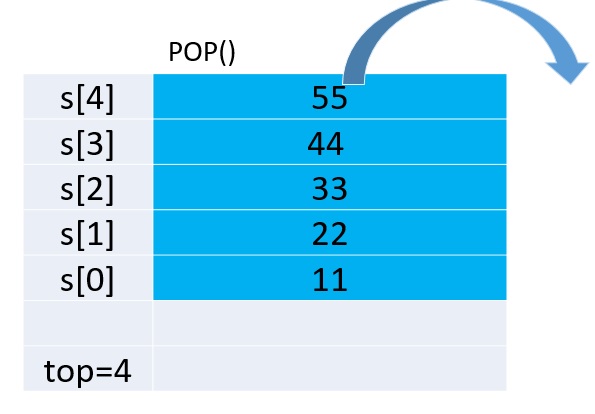


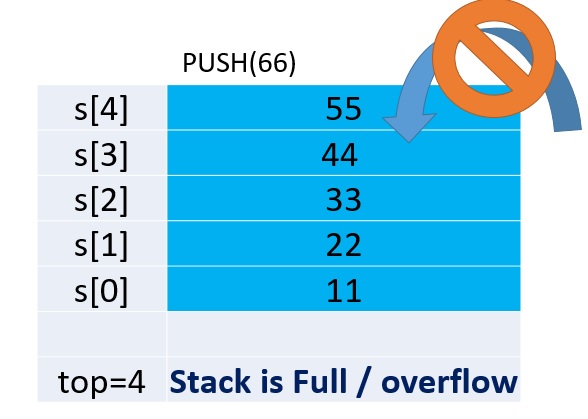


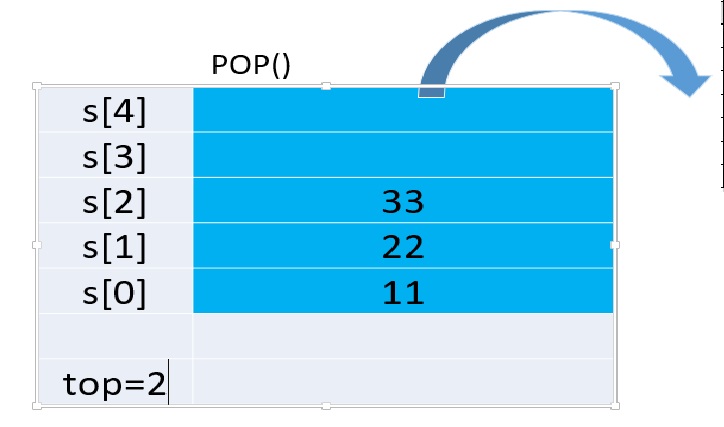


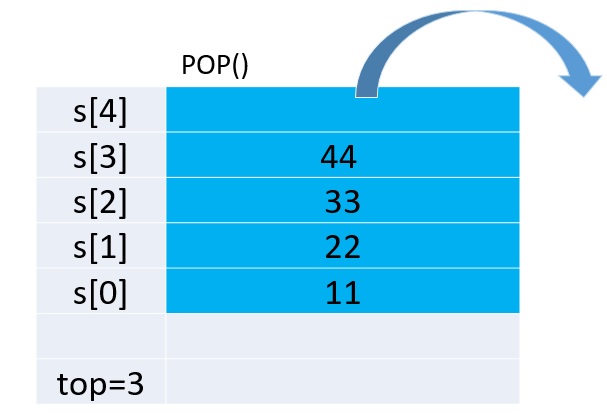


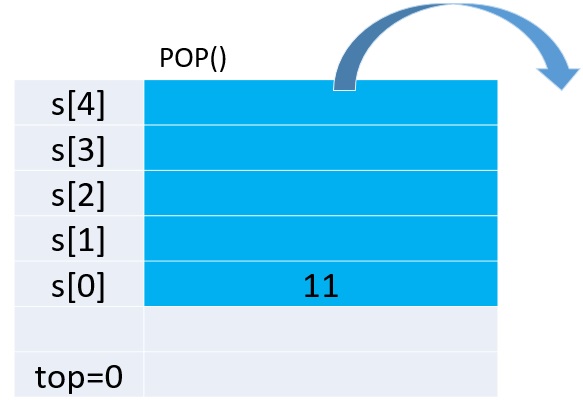


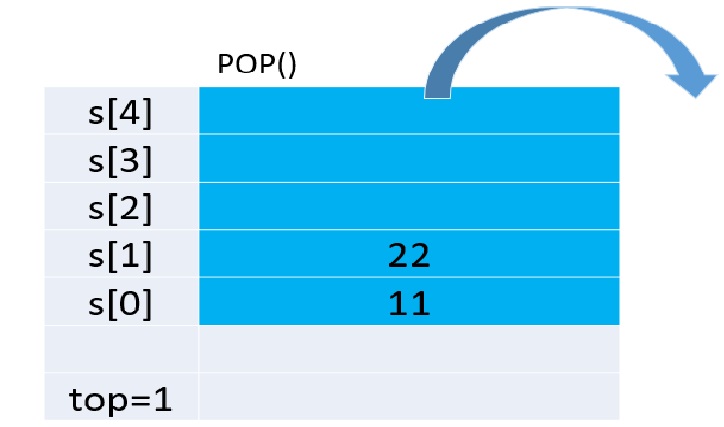


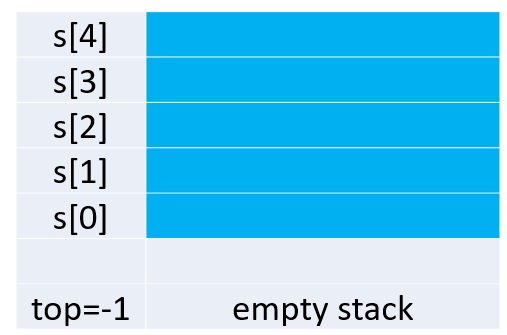


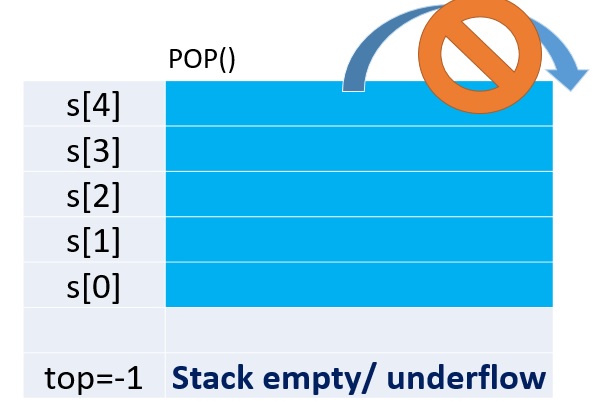












**Applications of Stack :**

* Recursion / Function calls (Stack is used to store activation records of the functions)
* Undo / Redo operations in text editor (browser)
* Page-visited history in a Web browser
* String reversal (Stack can be used to reverse a string)
* **Parenthesis Checker:** Compilers verify whether source code contains balanced parenthesis or not (If source code contains imbalanced parenthesis, then compiler will throw you an error. This can be checked by the stack.)

Ex) Enter any expression

(a+(b-c)) (a+b)\*c)

valid expression invalid expression

* **Expression Conversion**

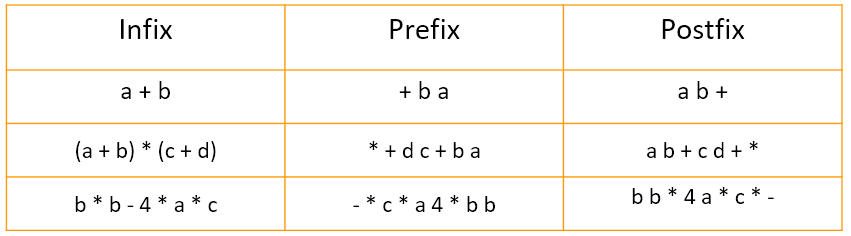
Converting one form of expressions to another is one of the important applications of stacks.

An expression can be represented in prefix, postfix or infix notation. Stack can be used to convert one form of expression to another. (see the below table)

* **Expression Evaluation**

Stack is used to evaluate prefix, postfix and infix expressions.

* **Backtracking** is a recursive algorithm which is used for solving the optimization problem. (game playing, finding paths, exhaustive searching)
* **Memory Management** is the important function of the Operating System. Stack also plays the main role when it comes to Memory Management.



**Programs:**

**1**. **Implement a stack using array to perform push(), pop() and display() operations based on user's choice: To be written in record**

#include<stdio.h>

#include<stdlib.h>

#define SIZE 10

void push(int value);

void pop();

void display();

int s[SIZE], top = -1;

void main()

{

int value, choice;

while(1)

    {

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

      printf("1. Push\n2. Pop\n3. Display\n4. Exit");

      printf("\nEnter your choice: ");

      scanf("%d",&choice);

      switch(choice)

      {

case 1: printf("Enter the value to be insert: ");

scanf("%d",&value);

push(value);

break;

case 2: pop();

break;

case 3: display();

break;

case 4: printf("\*\*\*\*EXIT\*\*\*\*\n");

                                exit(0);

default: printf("\nInvalid choice!!! Try again!!!");

      }

    }

}

void push(int value)

{

if(top == SIZE-1)

      printf("\nStack is Full!!! Insertion is not possible!!!");

    else

  {

      top++;

      s[top] = value;

      printf("\n%d element is inserted to stack", s[top]);

    }

}

void pop()

{

if(top == -1)

      printf("\nStack is Empty!!! Deletion is not possible!!!");

  else

  {

      printf("\nDeleted element : %d", s[top]);

      top--;

    }

}

void display()

{

if(top == -1)

      printf("\nStack is Empty!!!");

    else

  {

      int i;

      printf("\nStack elements are\n");

      for(i=0; i<=top; i++)

printf("%d\n",s[i]);

    }

}

**2. Check for balanced parentheses in an expression using Stack: To be written in record**

#include<stdio.h>

#include<stdlib.h>

#define max 50

void main()

{

char stk[max], exp[100];

int top, i;

top = -1;

printf("\nEnter an infix expression\n");

gets(exp);

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

{

if (exp[i]=='(' || exp[i] =='[' || exp[i] == '{' )

{

top++;

stk[top]= exp[i];

}

if (exp[i] == ')')

{

if (stk[top] == '(')

{

top--;

}

else

{

printf("Unbalanced expression\n");

exit(0);

}

}

if(exp[i] == ']')

{

if(stk[top] == '[')

{

top--;

}

else

{

printf("Unbalanced expression\n");

exit(0);

}

}

if(exp[i] == '}')

{

if(stk[top] == '{')

{

top--;

}

else

{

printf("Unbalanced expression\n");

exit(0);

}

}

}

if(top == -1)

printf("Expression is balanced\n");

else

printf("Exp is not balanced");

}

Output:

Enter an infix expression

((a/b) + (b\*c))

Expression is balanced

**3. Stack program to evaluate postfix expression: Self-Study**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<ctype.h>

#define STACK\_SIZE 20

void push(int item, int \*top, int a[])

{

\*top = \*top + 1; //a[++\*top] = item;

a[\*top] = item;

}

int pop(int \*top, int a[])

{

int item;

item = a[\*top];

\*top = \*top - 1;

return item;

}

int calculate(int op1, int op2, char symbol)

{

switch(symbol)

{

case '+': return (op1 + op2);

case '-': return (op1 - op2);

case '\*': return (op1 \* op2);

case '/': return (op1 / op2);

}

}

int main()

{

int a[100], i, top = -1, op1, op2, res;

char exp[20], symbol;

printf("Enter postfix expression \n");

scanf("%s",exp);

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

{

symbol = exp[i];

if (isdigit(symbol))

{

push(symbol-'0', &top, a);

}

else

{

op2 = pop(&top, a);

op1 = pop(&top, a);

res = calculate(op1, op2, symbol);

push(res, &top, a);

}

}

printf("Result of evaluation of expression is %d\n",a[top]);

}

This is DS lab

Welcome to Presidency University

Thank you all.

Have a great day