V Stack

UNIT V Stack

Revision

- Basic concept of stack
- Stack as an ADT
- Representation of stack using Array
- Application of Stack
- Infix to Postfix

Revision of Last Lecture

- Application of Stack
- Postfix Evaluation
- Linked Stack and Operations
- Recursion and Variants

UNIT V Stack

Unit V	Stack	(07 Hours)
Basic concept, stack Abstract Data Type, Representation of Stacks Using Sequential Organization, stack operations, Multiple Stacks, Applications of Stack- Expression Evaluation and Conversion, Polish notation and expression conversion, Need for prefix and postfix expressions, Postfix expression evaluation, Linked Stack and Operations. **Recursion-** concept, variants of recursion- direct, indirect, tail and tree, backtracking algorithmic strategy, use of stack in backtracking.		
#Exemplar/Case Studies	Android- multiple tasks/multiple activities and back-stack, Tower of Hanoi, 4 Queens problem.	
***Mapping of Course Outcomes for Unit V**	CO1, CO2, CO3, CO5, CO6	

Data

Data Structure

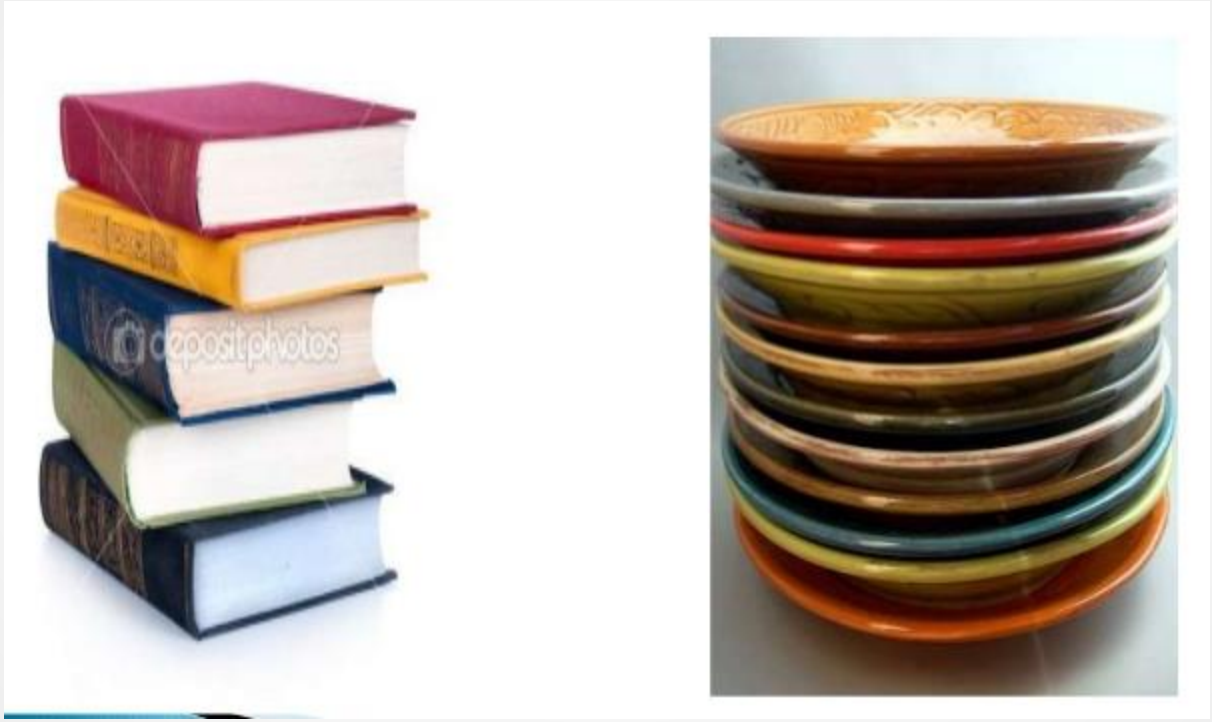
What is Data Structures?

- Data structure is a representation of data and the operations allowed on that data.

  - A data structure is a way to store and organize data in order to facilitate the access and modifications.

  - Data Structures are the method of representing of logical relationships between individual data elements related to the solution of a given problem.

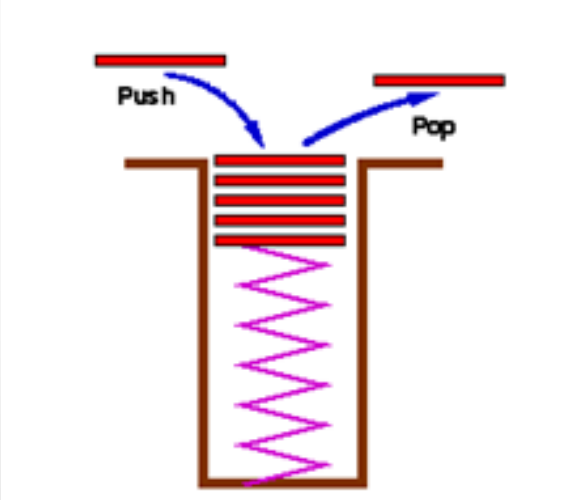
Stack Data Structures



Stack Data Structures

Stack – Basic Concept

- Stack is a LIFO (last in first out) structure.
- It is an ordered list of the same type of elements.
- A stack is a linear list where all insertions and deletions are permitted only at one end of the list.
- List of operations -
  - Initialize
  - Push
  - Pop
  - IsEmpty
  - IsFull



Stack – Basic Concept

•Push: Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.

•Pop: Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.

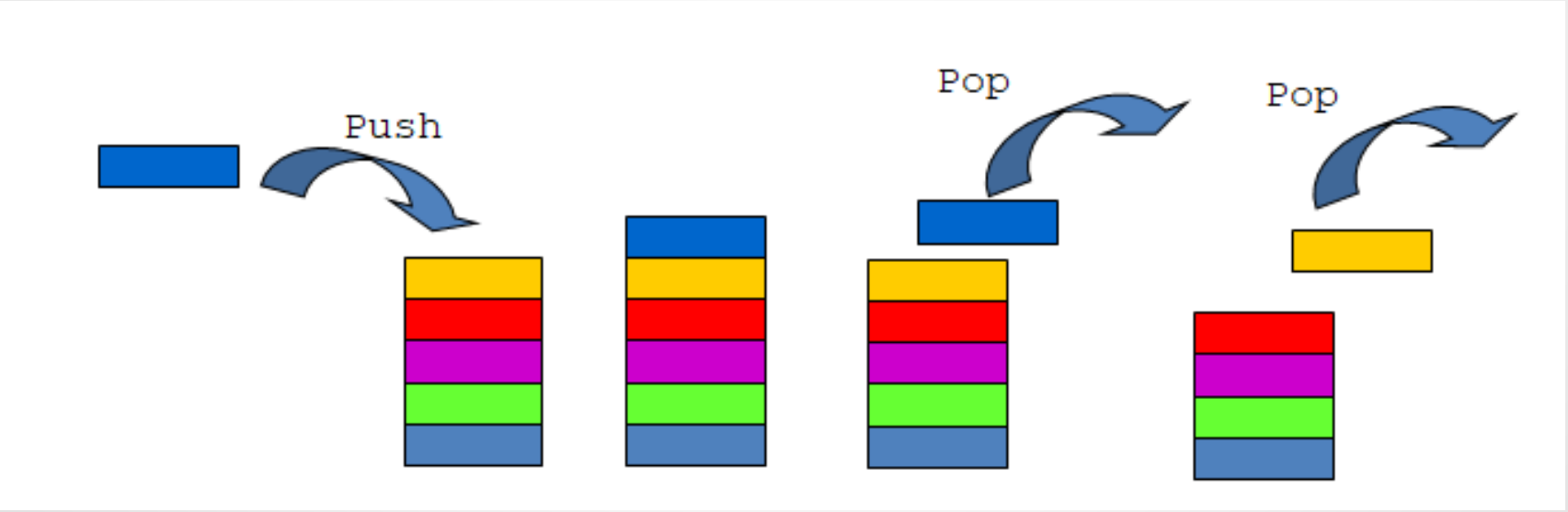
•Peek or Top: Returns top element of stack.

•isEmpty: Returns true if stack is empty, else false.

•isFull: Returns true if stack is full, else false.

Stack – Basic Concept

- Push: Adds an item in the stack.
- Pop: Removes an item from the stack.



Stack – Example

- Describe the output of the following series of stack operations
  - Push(8)
  - Push(3)
  - Pop()
  - Push(2)
  - Push(5)
  - Pop()
  - Pop()
  - Push(9)
  - Push(1)

Stack – Example

```
#include<iostream>
#include<ctype.h>
using namespace std;
class stack
{
int top;
char data[20];
public:
stack() {top=-1;}
}
```

Stack – Example

```
void push(int x)
{
 top++;
data[top]=x;
 }

 int pop()
 {
 char x;
 x=data[top];
 top--;
 return x;
}
```

Stack

- Stack Operation Complexity for Different

	Array Fixed-Size	Array Expandable (doubling strategy)	List Singly-Linked
Pop()	O(1)	O(1)	O(1)
Push(o)	O(1)	O(n) Worst Case O(1) Best Case O(1) Average Case	O(1)
Top()	O(1)	O(1)	O(1)
Size(), isEmpty()	O(1)	O(1)	O(1)

Applications of Stack

- Infix to Postfix /Prefix conversion
- Reversing a string
- Balancing of symbols
- Redo-undo features at many places like editors, photoshop.
- Forward and backward feature in web browsers
- Used in many algorithms like Tower of Hanoi, tree traversals,
- stock span problem, histogram problem.
- Other applications can be Backtracking, Knight tour problem, rat
- in a maze, N queen problem and sudoku solver
- In Graph Algorithms like Topological Sorting and

DSL Group - D

26	In any language program mostly syntax error occurs due to unbalancing delimiter such as (),{},[]. Write C++ program using stack to check whether given expression is well parenthesized or not.
27	Implement C++ program for expression conversion as infix to postfix and its evaluation using stack based on given conditions: 1. Operands and operator, both must be single character. 2. Input Postfix expression must be in a desired format. 3. Only '+', '-', '*' and '/ ' operators are expected.

Infix to Postfix Conversion

- Read Infix expression left to right one character at a time
- If input symbol operand then place it in postfix expression
- If input Symbol (- Push onto the stack
- If input Symbol operator - Push onto the stack
  - Check if Priority of operator in stack is grater than the priority of incoming operator then place it in postfix expression. Repeat step till ….
  - Otherwise Push operator onto the stack
  - If) – pop all operator till (
- Finally pop all element
- Print postfix expression as a result

Infix to Postfix Conversion

Infix Expression - A*B+C

Input	Action	Stack	Output/ Postfix
-	-	Empty	-
A	Print A	-	A
*	Push *	*	A
B	Print B	*	AB
+	Push +	+	AB*
C	Print C	+	AB*C+

Infix to Postfix Conversion

Infix Expression –
(A+B)*(C-D)

Input	Action	Stack	Output/ Postfix
-	-	Empty	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-

Infix to Postfix Conversion

Infix Expression

(a+b)*d+e(f+a*d)+c

(a+(b*c/d)-e)

Infix to Postfix Conversion

Infix Expression

(a+b)*d+e(f+a*d)+c
ANS - ab+d*e+fad*+c+

(a+(b*c/d)-e)
ANS –abc*d/+e-

Infix to Postfix Conversion

Infix Expression

(a+b)*d+e(f+a*d)+c

Input	Action	Stack	Output/ Postfix
(	Push (	(	(
a	Print a	(	a
+	Push +	(+	a
b	Print b	(+	ab
)	Pop all until (& print all the content	-	ab+
*	Push *	*	ab+
d	Print d	*	ab+d
+	Push + & Pop	+	ab+d*
e	Print e	+	ab+d*e
(	Push (	(+	ab+d*e
f	Print f	(+	ab+d*ef
+	Push +	(++	ab+d*ef
a	Print a	(++	ab+d*efa
*	Push *	Push *	Push *

Infix to Postfix Conversion

Post fix expression is:
ab+d*efad*++c+

Infix to Postfix Conversion Group - D

27	Implement C++ program for expression conversion as infix to postfix and its evaluation using stack based on given conditions: 1. Operands and operator, both must be single character. 2. Input Postfix expression must be in a desired format. 3. Only '+', '-', '*' and '/ ' operators are expected.

Infix to Postfix Conversion

```
int main()
{
stack s;
char infix[20],postfix[20];
cout<<"Enter the  infix expression";
cin>>infix;
s.conversion(infix,postfix);
cout<<"Post fix expression is:  "<<postfix;
return 0;
}
```

Infix to Postfix Conversion

```
#include<iostream>
#include<ctype.h>
using namespace std;
class stack
{
int top;
char data[20];
public:
stack()
{
top=-1;
}
```

Infix to Postfix Conversion

```
void  conversion(char infix[20],char postfix[20])
{
 int j=0,i;
 char e,token,x;
 for(i=0;infix[i]!='\0';i++)
 {
 token=infix[i];
 if(isalnum(token))
 {
 postfix[j]=token;
 j++;
 }
```

Infix to Postfix Conversion

```
else
{
if(token=='(')
push(token);
else if(token==')')
{
while((x=pop())!='(')

{
postfix[j]=x;
      j++;
         }
}
```

Infix to Postfix Conversion

```
else{
 e=topmost();
 while(precedence(token)<=precedence(e) && !empty())
                        {
                         x=pop();
                         postfix[j]=x;
                         j++;
                        }
 push(token);
 }} }
 while(!empty())
 {
 x=pop();
 postfix[j]=x;
 j++;
 }
postfix[j]='\0';}
```

Infix to Postfix Conversion

```
 void push(int x)
 {
 top++;
 data[top]=x;
 }
 int pop()
 { char x;
 x=data[top];
 top--;
 return x;
 }
char topmost()
 {
char e;
 e=data[top];
 return e;
 }
```

Infix to Postfix Conversion

```
int precedence(char x)
 {
 if(x=='(')
 return 0;
 if(x=='+'|| x=='-')
 return 1;
 if(x=='*'|| x=='/' ||x=='%')
              return 2;
 else
return 3;
 }
 int empty()
 {
 if(top==-1)
 return 1;
 else
 return 0;
 }
};
```

UNIT V Stack

Unit V	Stack	(07 Hours)
Basic concept, stack Abstract Data Type, Representation of Stacks Using Sequential Organization, stack operations, Multiple Stacks, Applications of Stack- Expression Evaluation and Conversion, Polish notation and expression conversion, Need for prefix and postfix expressions, Postfix expression evaluation, Linked Stack and Operations. **Recursion-** concept, variants of recursion- direct, indirect, tail and tree, backtracking algorithmic strategy, use of stack in backtracking.		
#Exemplar/Case Studies	Android- multiple tasks/multiple activities and back-stack, Tower of Hanoi, 4 Queens problem.	
***Mapping of Course Outcomes for Unit V**	CO1, CO2, CO3, CO5, CO6	

Infix to Postfix Conversion

(A + B * C / D - E + F / G / (H + I))

Infix to Postfix Conversion

(A + B * C / D - E + F / G / (H + I))

Input Symbol	Content of stack	Reverse polish
	(	
(	((	
A	((	
+	((+	A
B	((+ B	A
*	((+ *	A B
C	((+ * C	A B
/	((+ /	A B C *
D	((+ / D	A B C *
-	((-	A B C * D / +
E	((- E	A B C * D / +
+	((+	A B C * D / + E -
F	((+ F	A B C * D / + E -
/	((+ /	A B C * D / + E – F
G	((+ / G	A B C * D / + E – F
/	((+ /	A B C * D / + E – F G /
(	((+ / (	A B C * D / + E – F G /
H	((+ / (H	A B C * D / + E – F G /
+	((+ / (+	A B C * D / + E – F G / H
I	((+ / (+ I	A B C * D / + E – F G / H
)	((+ /	A B C * D / + E – F G / H I +
)	(	A B C * D / + E – F G / H I + / +
)		A B C * D / + E – F G / H I + / +

Postfix expression is: **A B C * D / + E – F G / H I + / +**

Infix to Postfix Conversion

(A + B * C / D - E + F / G / (H + I))

Infix to Postfix Conversion

(A + B) * C + D / (B + A * C) + D

Input Symbol	Content of stack	Reverse polish
	(	
(	((	
A	((A	
+	((+	A
B	((+ B	A
)	(	A B +
*	(*	A B +
C	(* C	A B +
+	(+	A B + C *
D	(+ D	A B + C *
/	(+ /	A B + C * D
(	(+ / (	A B + C * D
B	(+ / (B	A B + C * D
+	(+ / (+	A B + C * D B
A	(+ / (+ A	A B + C * D B
*	(+ / (+ *	A B + C * D B A
C	(+ / (+ * C	A B + C * D B A
)	(+ /	A B + C * D B A C * +
+	(+	A B + C * D B A C * + / +
D	(+ D	A B + C * D B A C * + / +
)		A B + C * D B A C * + / + D +

Postfix expression is: **A B + C * D B A C * + / + D +**

Fully Parenthesized Expression

- A FPE has exactly one set of Parentheses enclosing each operator and its operands.
- Which is fully parenthesized?

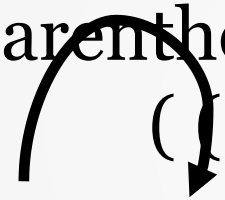
(A + B) * C

((A + B) * C)

((A + B) * (C))

Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:



((A + B) * (C + D))

Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

(+ A B * (C + D))

Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

* + A B (C + D)

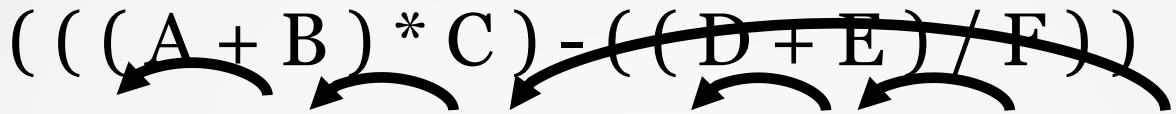
Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

* + A B + C D

Order of operands does not change!

Infix to Postfix



A B + C * D E + F / -

- Operand order does not change!
- Operators are in order of evaluation!

Computer Algorithm
FPE Infix To Postfix

•Assumptions:

1.Space delimited list of tokens represents a FPE infix expression

2.Operands are single characters.

3. Operators +,-,*,/

FPE Infix To Postfix

- Initialize a Stack for operators, output list
- Split the input into a list of tokens.
- for each token (left to right):
    - if it is operand: append to output
    - if it is '(': push onto Stack
    - if it is ')': pop & append till '('

FPE Infix to Postfix

(((A + B) * (C - E)) / (F + G))

- stack: <empty>
- output: []

FPE Infix to Postfix

((A + B) * (C - E)) / (F + G))

- stack: (
- output: []

FPE Infix to Postfix

(A + B) * (C - E)) / (F + G))

- stack: ((
- output: []

FPE Infix to Postfix

A + B) * (C - E)) / (F + G))

- stack: (((
- output: []

FPE Infix to Postfix

+ B) * (C - E)) / (F + G))

- stack: (((
- output: [A]

FPE Infix to Postfix

B) * (C - E)) / (F + G))

- stack: (((+
- output: [A]

FPE Infix to Postfix

) * (C - E)) / (F + G))

- stack: (((+
- output: [A B]

FPE Infix to Postfix

* (C - E)) / (F + G))

- stack: ((
- output: [A B +]

FPE Infix to Postfix

(C - E)) / (F + G))

- stack: ((*
- output: [A B +]

FPE Infix to Postfix

C - E)) / (F + G))

- stack: ((* (
- output: [A B +]

FPE Infix to Postfix

- E)) / (F + G))

- stack: ((* (
- output: [A B + C]

FPE Infix to Postfix

E)) / (F + G))

- stack: ((* (-
- output: [A B + C]

FPE Infix to Postfix

)) / (F + G))

- stack: ((* (-
- output: [A B + C E]

FPE Infix to Postfix

) / (F + G))

- stack: ((*
- output: [A B + C E -]

FPE Infix to Postfix

/ (F + G))

- stack: (
- output: [A B + C E - *]

FPE Infix to Postfix

(F + G))

- stack: (/
- output: [A B + C E - *]

FPE Infix to Postfix

F + G))

- stack: (/ (
- output: [A B + C E - *]

FPE Infix to Postfix

+ G))

- stack: (/ (
- output: [A B + C E - * F]

FPE Infix to Postfix

G))

- stack: (/ (+
- output: [A B + C E - * F]

FPE Infix to Postfix

))

- stack: (/ (+
- output: [A B + C E - * F G]

FPE Infix to Postfix

)

- stack: (/
- output: [A B + C E - * F G +]

FPE Infix to Postfix

- stack: <empty>
- output: [A B + C E - * F G + /]

Infix to Postfix

- Initialize a Stack for operators, output list
- Split the input into a list of tokens.
- for each token (left to right):

```
        if it is operand:  append to output
        if it is '(': push onto Stack
        if it is ')': pop & append till '('
        if it in '+-*/':
            while peek has precedence ≥ it:
                pop & append
            push onto Stack
    pop and append the rest of the Stack.
```

Conversion

- Infix to Postfix Conversion
- Postfix to Infix Conversion
- Postfix to Prefix Conversion
- Infix to Prefix Conversion
- Prefix to Infix Conversion
- Prefix to Postfix Conversion

Infix to Prefix

Convert the following string into prefix: A-B/(C*D^E)

Step-1 : reverse infix expression

) E ^) D * C ((/ B - A

Step-2 : convert '(' to ')' and ')' to '(' and append extra ')' at last

(E ^ (D * C)) / B - A

Step-3 : Now convert this string to postfix

Input Symbol	Content of stack	Reverse polish	Rank
	(		0
(	((		0
E	((E		0
^	((^	E	1
(	((^ (	E	1
D	((^ (D	E	1
*	((^ (*	E D	2
C	((^ (* C	E D	2
)	((^	E D C *	2
)	(	E D C * ^	1
/	(/	E D C * ^	1
B	(/ B	E D C * ^	1
-	(-	E D C * ^ B /	1
A	(- A	E D C * ^ B /	1
)		E D C * ^ B / A -	1

Step 4 : Reverse this postfix expression

- A / B ^ * C D E

Infix to Prefix

(a + b ^ c ^ d) * (e + f / d)

Infix to Prefix

$$(a + b \wedge c \wedge d) * (e + f / d)$$

Step-1 : reverse infix expression

) d / f + e (*) d ^ c ^ b + a (

Step-2 : convert '(' to ')' and ')' to '(' and append extra ')' at last

(d / f + e) * (d ^ c ^ b + a))

Step-3 : Now convert this string to postfix

Infix to Prefix

(a + b ^ c ^ d) * (e + f / d)

(d / f + e) * (d ^ c ^ b + a))

Step-3 : Now convert this string to postfix

Input symbol	Content of stack	Reverse polish
	(	
(	((	
d	((d	
/	((/	d
f	((/f	d
+	((+	df/
e	((+e	df/
)	(	d f/e+
*	(+	df/e+
(	(*(	df/e+
d	(*(d	df/e+
^	(*(^	df/e+d
c	(*(^c	df/e+d
^	(*(^^	df/e+dc
b	(*(^^b	df/e+dc
+	(*(+	df/e+dcb^^
a	(*(+a	df/e+dcb^^
)	(*	df/e+dcb^^a+
)		df/e+dcb^^a+*

Step 4 : Reverse this postfix expression

*+a^^bcd+e/fd

Infix to Prefix and postfix

Examples of infix to prefix and post fix

Infix	PostFix	Prefix
A+B	AB+	+AB
(A+B) * (C + D)	AB+CD+*	*+AB+CD
A-B/(C*D^E)	ABCDE^*/-	-A/B*C^DE

Infix to Prefix and postfix

Expression No	Infix Notation	Prefix Notation	Postfix Notation
1	a + b	+ a b	a b +
2	(a + b) * c	* + a b c	a b + c *
3	a * (b + c)	* a + b c	a b c + *
4	a / b + c / d	+ / a b / c d	a b / c d / +
5	(a + b) * (c + d)	* + a b + c d	a b + c d + *
6	((a + b) * c) - d	- * + a b c d	a b + c * d -

Postfix to Infix

- pqr-+st-xy-z+/*

Postfix to Prefix

- pqr-+st-xy-z+/*

Prefix to Infix

- *+p-qr/-st+-uvw

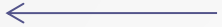
Prefix to Infix

- *+p-qr/-st+-uvw

- (p+q-r)*(s-t)/(u-v+w)

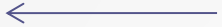
Prefix to Postfix

- *+a-bc/-de+-fgh



Prefix to Postfix

- *+a-bc/-de+-fgh



- abc-+de-fg-h+/*

UNIT V Stack

Unit V	Stack	(07 Hours)
Basic concept, stack Abstract Data Type, Representation of Stacks Using Sequential Organization, stack operations, Multiple Stacks, Applications of Stack- Expression Evaluation and Conversion, Polish notation and expression conversion, Need for prefix and postfix expressions, Postfix expression evaluation, Linked Stack and Operations. **Recursion-** concept, variants of recursion- direct, indirect, tail and tree, backtracking algorithmic strategy, use of stack in backtracking.		
#Exemplar/Case Studies	Android- multiple tasks/multiple activities and back-stack, Tower of Hanoi, 4 Queens problem.	
***Mapping of Course Outcomes for Unit V**	CO1, CO2, CO3, CO5, CO6	

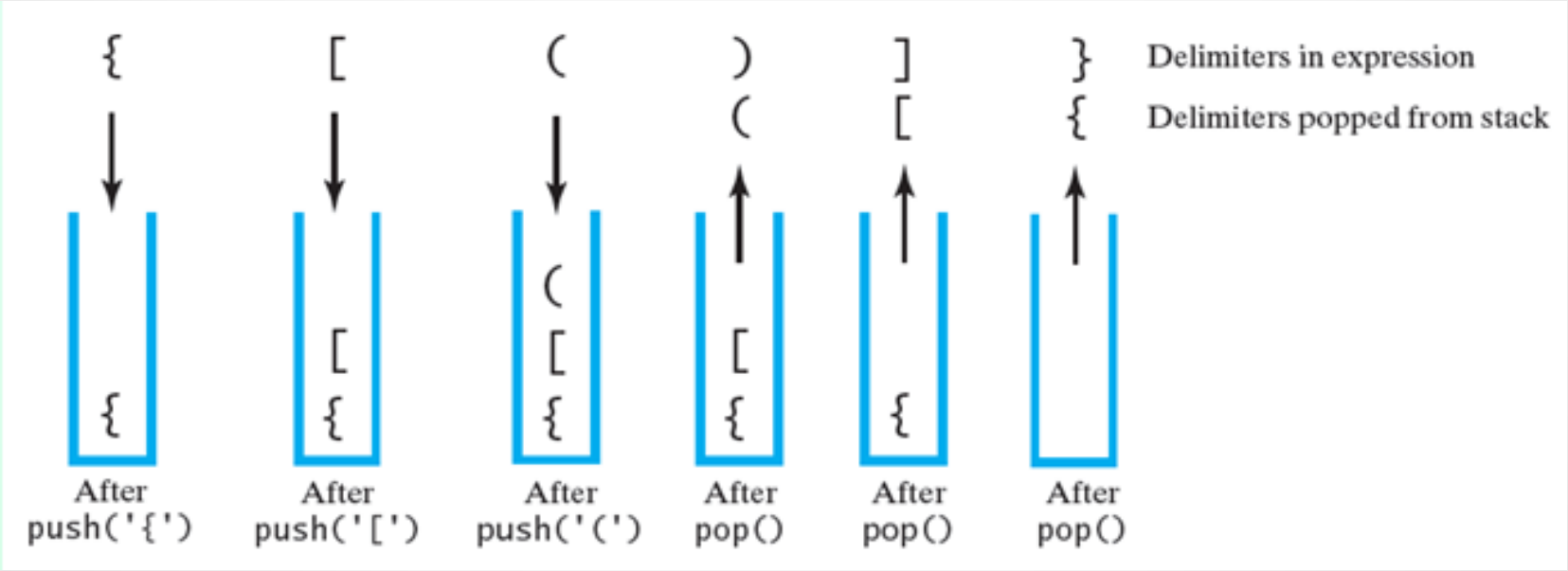
Applications of Stack

- Infix to Postfix /Prefix conversion
- Reversing a string
- Balancing of symbols
- Redo-undo features at many places like editors, photoshop.
- Forward and backward feature in web browsers
- Used in many algorithms like Tower of Hanoi, tree traversals,
- stock span problem, histogram problem.
- Other applications can be Backtracking, Knight tour problem, rat
- in a maze, N queen problem and sudoku solver
- In Graph Algorithms like Topological Sorting and

DSL Group - D

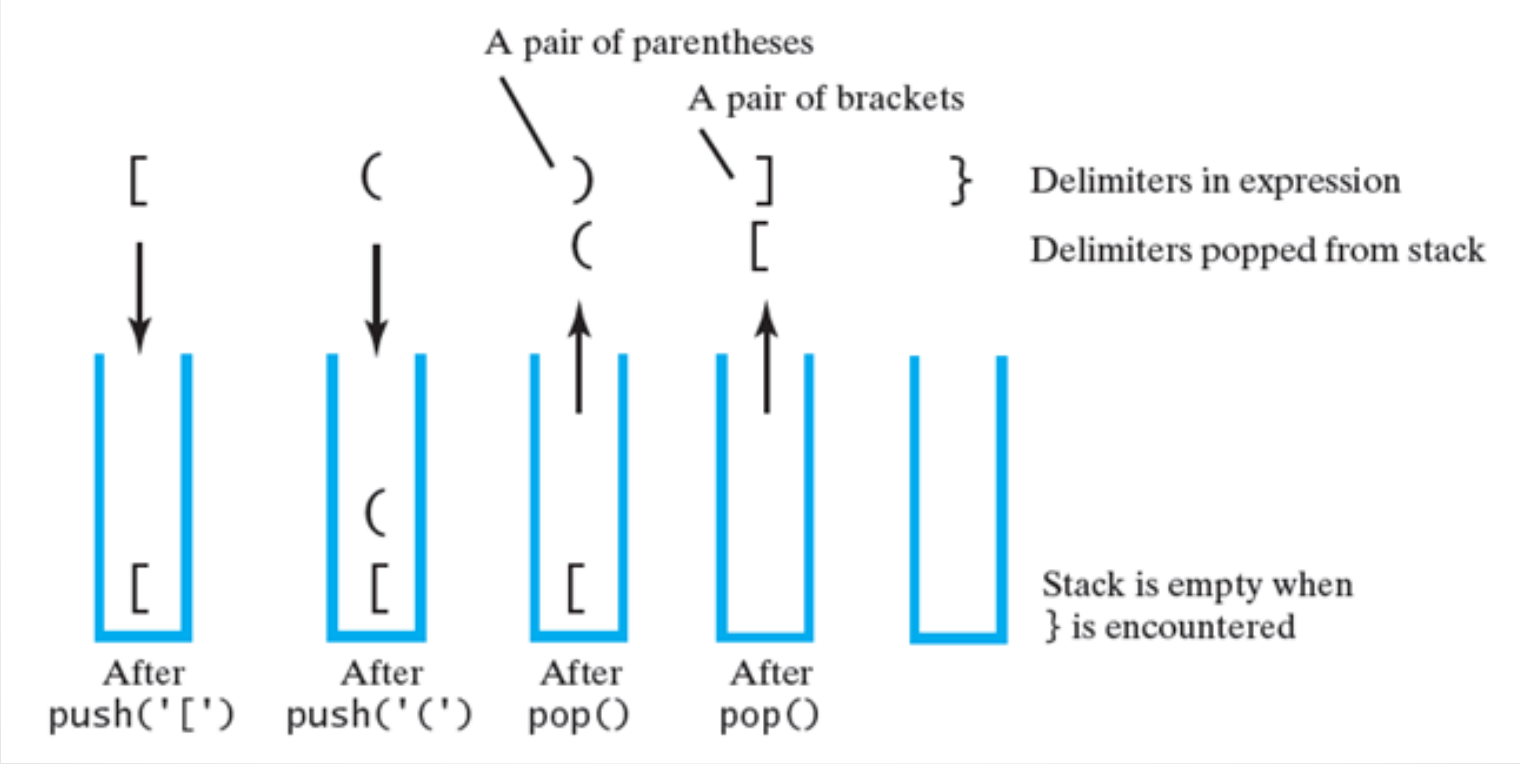
26	In any language program mostly syntax error occurs due to unbalancing delimiter such as (),{},[]. Write C++ program using stack to check whether given expression is well parenthesized or not.
27	Implement C++ program for expression conversion as infix to postfix and its evaluation using stack based on given conditions: 1. Operands and operator, both must be single character. 2. Input Postfix expression must be in a desired format. 3. Only '+', '-', '*' and '/ ' operators are expected.

Balancing of symbols



The contents of a stack during the scan of an expression that contains the balanced delimiters { [( )] }

Balancing of symbols



The contents of a stack during the scan of an expression that contains the unbalanced delimiters [( )] }

DSL Group - D

26	In any language program mostly syntax error occurs due to unbalancing delimiter such as (),{},[]. Write C++ program using stack to check whether given expression is well parenthesized or not.
27	Implement C++ program for expression conversion as infix to postfix and its evaluation using stack based on given conditions: 1. Operands and operator, both must be single character. 2. Input Postfix expression must be in a desired format. 3. Only '+', '-', '*' and '/ ' operators are expected.

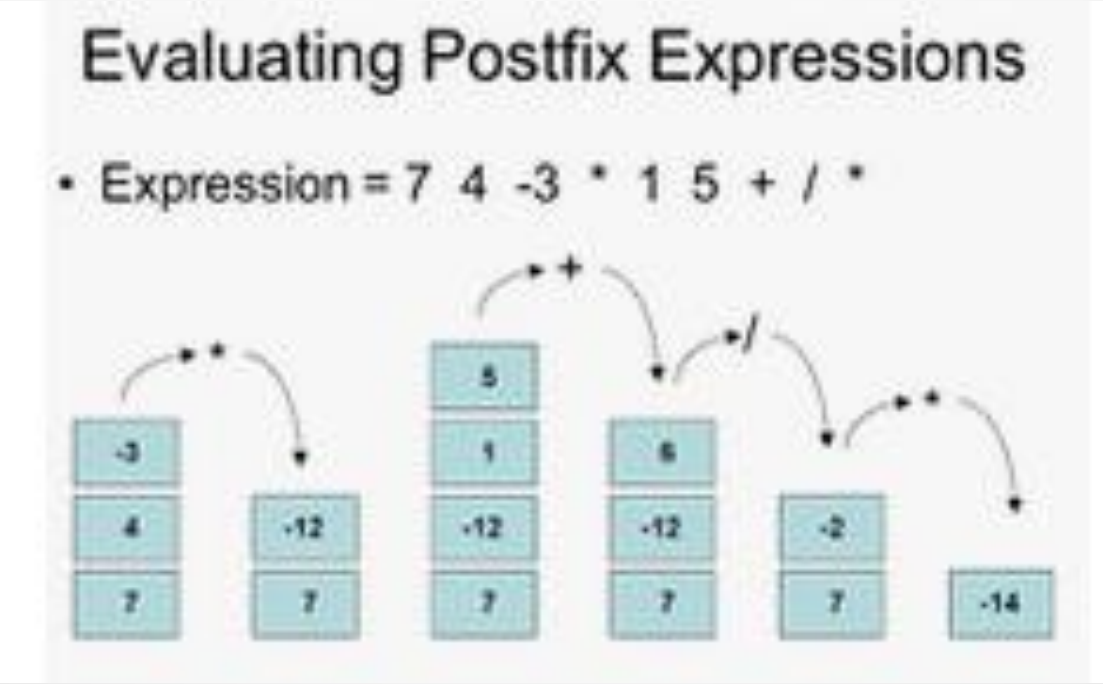
Need for Prefix and Postfix Expressions

- Operators are no longer ambiguous with respect to the operand

- Order of operations within prefix and postfix expressions is completely determined by only the position of the operator

- During compilation of expression, infix expression is converted to postfix form first and then it is evaluated

Postfix Expression Evaluations



Postfix Expression Evaluations

- AB+C-BA+C$-
  - For A=1, B=2, C=3

Postfix Expression Evaluations

- ABC+*CBA-+*
  - For A=1, B=2, C=3

Postfix Expression Evaluations

Example: postfix expression Evaluation

- Evaluate: ABC+*CBA-+* where A=1, B=2, C =3.

S.N	Scan char	Value	Op1	Op2	Result	Vstack
1	A	1				1
2	B	2				1, 2
3	C	3				1,2,3
4	+		2	3	5	1,5
5	*		1	5	5	5
6	C	3				5,3
7	B	2				5,3,2
8	A	1				5,3,2,1
9	-		2	1	1	5,3,1
10	+		3	1	4	5,4
11	*		5	4	20	20
Final Result of Expression : 20						

Postfix Expression Evaluations

Evaluate RPN in Stack

- ab*c+
- (a*b)+c *(Infix)*

- Push operands
- When reach operator evaluate previous 2 operands
- Pop 2 operands
- Push evaluation

UNIT V Stack

Unit V	Stack	(07 Hours)
Basic concept, stack Abstract Data Type, Representation of Stacks Using Sequential Organization, stack operations, Multiple Stacks, Applications of Stack- Expression Evaluation and Conversion, Polish notation and expression conversion, Need for prefix and postfix expressions, Postfix expression evaluation, Linked Stack and Operations. **Recursion-** concept, variants of recursion- direct, indirect, tail and tree, backtracking algorithmic strategy, use of stack in backtracking.		
#Exemplar/Case Studies	Android- multiple tasks/multiple activities and back-stack, Tower of Hanoi, 4 Queens problem.	
***Mapping of Course Outcomes for Unit V**	CO1, CO2, CO3, CO5, CO6	

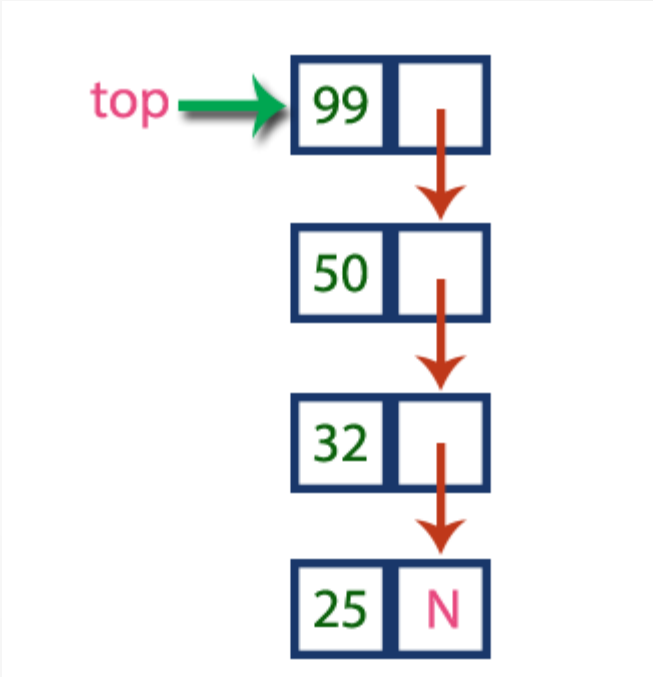
Linked Stack and Operations

- Stack by using Linked List
- Size of stack
- Node structure for linked stack

```
Struct node
{
Int data
Struct node *next
}node;
```

Linked Stack and Operations

```
Struct node
{
Int data
Struct node *next
}node;
```



Linked Stack and Operations

Write C++ Program for implementation of Stack using Linked List

Recursion

- A procedure that contains a procedure call to itself or a procedure call to second procedure which eventually causes the first procedure to be called is known as recursive procedure.
- There are two important conditions that must be satisfied by any recursive procedure
  a. Each time a procedure calls itself it must be nearer in some sense to a solution
  b. There must be a decision criterion for stopping the process or computation
- There are two types of recursion
  - Primitive Recursion: this is recursive defined function. E.g. Factorial function
  - Non-Primitive Recursion: this is recursive use of procedure. E.g. Find GCD of given two nunbers