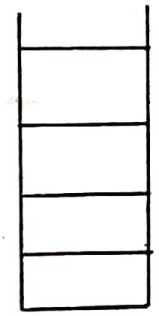
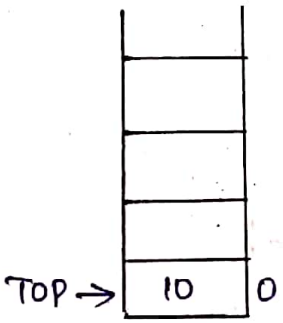
Stack

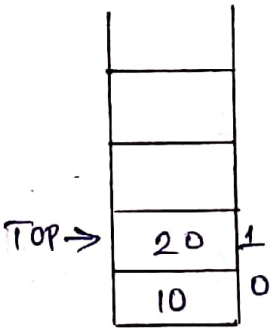
- Stack was first proposed in 1955 and patented in 1957 by German Friedrich L. Bauer.
- Stack is a linear data structure.
- A stack is an ordered collection of homogeneous data elements where the insertion and deletion operations occur at only one end. This end is known as Top of stack (TOP).
- Stack is also called Last-In-First-Out (LIFO).
- Assume 5 elements 10, 20, 30, 40 and 50 to be added into a stack.



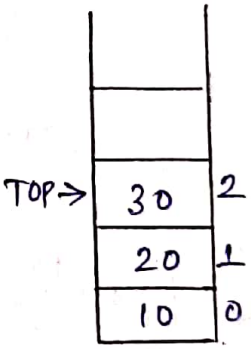
TOP = -1



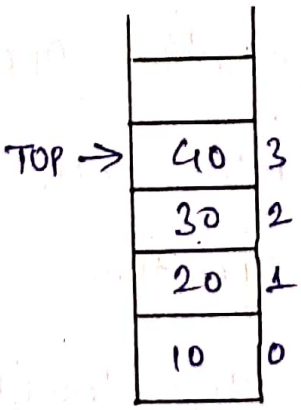
Insert 10



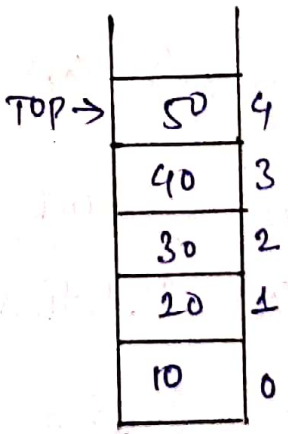
Insert 20



Insert 30



Insert 40

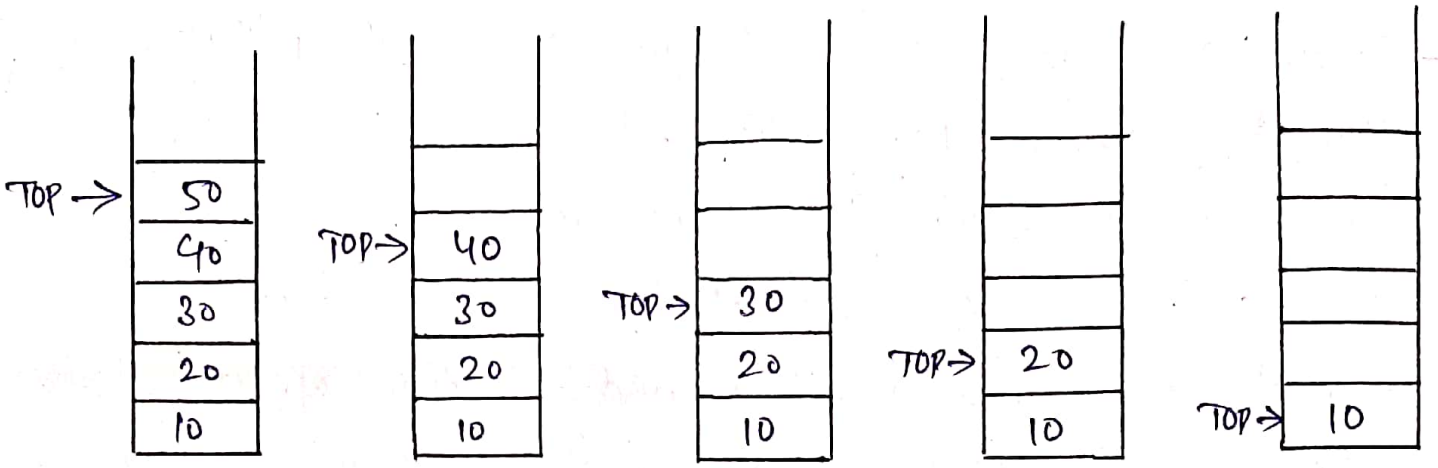


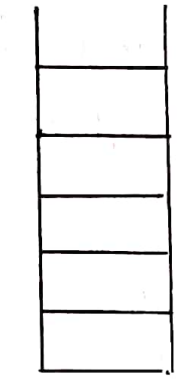
Insert 50

∴ TOP = 4

Initially TOP of stack is -1. For every insertion, TOP of stack is incremented by 1.

(2) Now assume 5 elements are deleted from stack.





For every deletion operation on stack, TOP of stack is decremented by 1.

Operations on Stack

The following operations are performed on stack.

1. Push (stack, element) (or) Insertion Operation
   - to insert "element" into stack
2. Pop (stack) (or) Deletion Operation
   - to access and remove top element of stack.
3. Peek (stack) (or) Peep (stack)
   - to access top element of stack without removing it from the stack.

Algorithm For Push and POP operations

(1) Push Operation

The process of adding a new element to the top of stack is called Push operation.

Steps :->

1. If TOP = MAX - 1, then write "STACK OVERFLOW" and stop.
2. READ element
3. TOP = TOP + 1
4. STACK [TOP] = element
5. Stop.

(4)

Stack Overflow Condition

Suppose we are inserting an element onto stack and max size is reached i.e stack become FULL and if we want to insert one more new element onto stack then it is not possible to insert new element. This condition is known as stack overflow condition.

if TOP = MAX − 1 , then stack is OVERFLOW.

PUSH function

```
void push()
{
  int num;
  if (top == MAX-1)
   {
    printf ("\n Stack Overflow");
    return;
   }
  else {
     top ++;
     printf ("\n Enter number to insert : ");
     scanf ("%d", &num);
     stack [top] = num;
   }
}
```

(2) Pop Operation

The process of deleting an element from the top of stack is called pop operation.

Steps :->

1. If top < 0 then write "Stack Underflow" and stop.
2. Stack [top] = NULL
3. top = top - 1
4. stop

Stack Underflow condition

When all the elements are deleted, then value of top becomes -1. Suppose we again perform Pop operation on stack, when stack is empty, it is not possible to delete any element and this situation is called stack underflow.

If top = -1, then Stack is Underflow.

Pop function

```
void pop()
{
 int num;
 if (top == -1)
    { printf("\n Stack Underflow");
     return;
    }
 else {
```

(5)

```
    num = stack[top];
    top--;
    return num;
  }
}
```

Implementation of Stack Using Array

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 10
int stack[MAX];
int top=-1;
void push();
void pop();
void peek();
void display();

void main()
{
  int ch;
  while(1)
    {
      printf("\n Stack Implementation Using Array\n");
      printf("1. Push \n");
      printf("2. Pop \n");
      printf("3. Peek \n");
      printf("4. Display \n");
      printf("5. Quit \n");
      printf("Enter Your Choice: ");
      scanf("%d",&ch);

      switch(ch)
       {
```

```
        case 1: push();
                break;
        case 2: pop();
                break;
        case 3: peek();
                break;
        case 4: display();
                break;
        case 5: exit(1);
        default: printf("Invalid choice \n");
    } /* End of switch case */
  } /* End of while loop */
} /* End of main() */

void push()
  {
    if (top == MAX-1)
      {
        printf("\n Stack Overflow \n");
      }
    else{
        int num;
        printf("Enter number to push: ");
        scanf("%d", &num);
```

```
        top++;
        stack [top] = num;
        printf (" %d pushed \n", num);
    }
}

void pop()
{
    if (top == -1)
    {
        printf (" \n Stack Underflow \n");
    }
    else{
        int num;
        num = stack [top];
        top--;
    printf (" %d popped \n", num);
    }
}

    void peek()
    {
        if (top == -1)
        {
            printf (" \n Stack is empty \n");
        }
    else{
            int num;
            num = stack [top];
```

```
printf(" %d is the TOP of stack \n", num);
  }
}
void display()
 { int i;
  if (top == -1)
    {
    printf(" No elements in stack \n");
    }
  else{
     for(i= top ; i>=0 ; i--)
       {
       printf(" Stack Elements are: %d \n", stack[i]);
       }
    }
 }
```

(10)

Array implementation has the following disadvantages:→

(1) Memory space is wasted if we store less number of items in the array than the maximum size of the array.

(2) Limitation in storing the items into the stack.

(3) Insertion and deletion operations in a stack using an array involves more data movements.

Advantages of array implementation is that- the implementation of stack using array is easy.

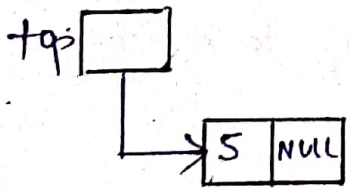
Linked List Implementation of Stack

- When the size of stack is not known in advance, it is better to implement it as a linked list.

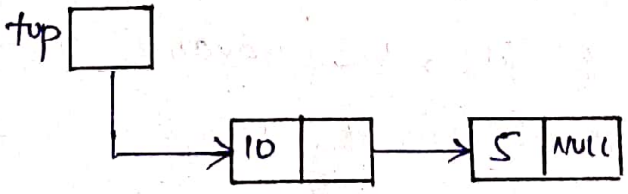
We will take a single linked list so the structure of node would be –

```
struct node {
    int data;
    struct node *next;
};
```

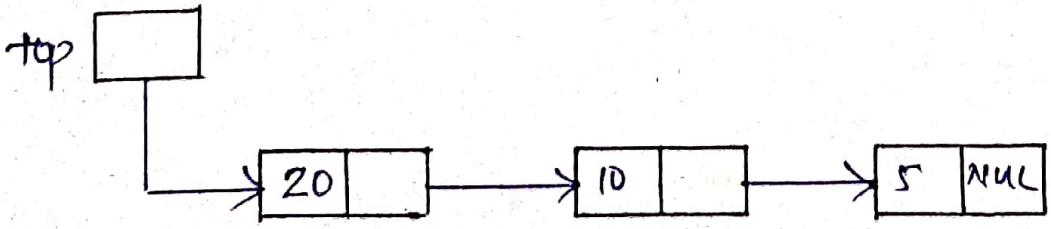
- We will take the beginning of linked list as the top of stack.
- For Push operation, a node will be inserted in the beginning of the list.
- For Pop operation, first node of the list will be deleted.
- We will take a pointer "top" that points to the first node of the linked list



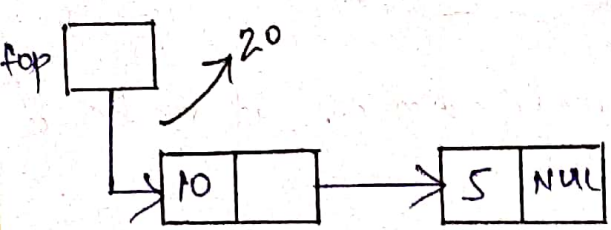
(a) Push 5



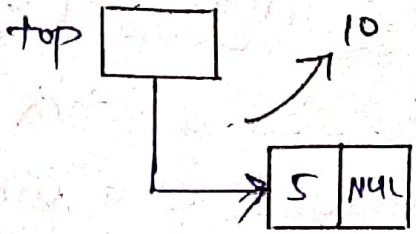
(b) Push 10



(c) Push 20



(d) Pop



(e) Pop

fig. Push and Pop operation in stack using linked list

Advantages

(1) Insertion and deletion operation do not involve more data movements.

(2) Another advantage is that memory space is not wasted because memory space is allocated only when the user wants to push an element into the stack.

(3) To implement a push, we create a new node in the list and attach it as the new first element.

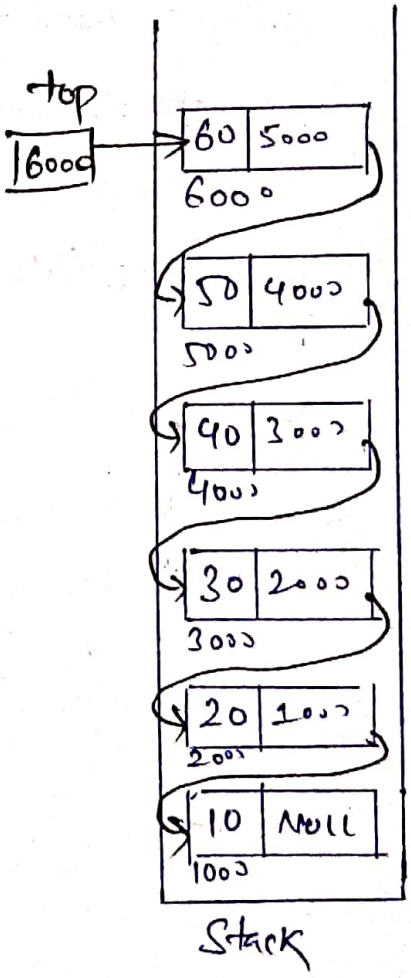
To implement a pop, we advance top of stack to the second element in the list.

Stack Using Linked list

Structure of Node

```
struct node
{
   int data;
   struct node * next;
};
   struct node *top =NULL;

void push()
{
  struct node *temp;
 temp= (struct node *)malloc(sizeof
                    (struct node));
  if (temp == NULL)
   {
  printf(" Stack overflow \n");
   }
  else {
    printf("Enter Node data: ");
    Scanf ("%d", &temp->data);
     temp->next=NULL;
      temp->next = top;
       top= temp;
   }
}
```



```
void pop()
{
  if (top == NULL)
  {
   printf(" Stack Underflow \n");
  }
  else{
     struct node *temp;
     temp= top;
  printf(" Data to be deleted is %d ", temp->data);
       top= temp->next;
       temp ->next = NULL;
       free (temp);
   }
}
```

Implementation of Stack Using linked list

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node * nxt;
};
struct node * top = NULL;
void push();
void pop();
void peek();
void display();
void main()
{
  int ch;
  while(1)
    {
    printf("\n Stack Implementation Using Linked list\n");
    printf("1. Push \n");
    printf("2. Pop \n");
    printf("3. Peek \n");
    printf("4. Display \n");
    printf("5. Quit \n");
    printf("Enter Your Choice : ");
    scanf("%d", &ch);
```

```
switch (ch)
{
case 1: push();
        break;
case 2: pop();
        break;
case 3: peek();
        break;
case 4: display();
        break;
case 5: exit(1);
default: printf("Invalid choice \n");
} /* End of switch case */
} /* End of while loop */
} /* End of main() */

void push()
{
  struct node *temp;
 temp = (struct node *) malloc (sizeof(struct node));
  if (temp == NULL)
     {
      printf (" Stack Overflow \n");
     }
```

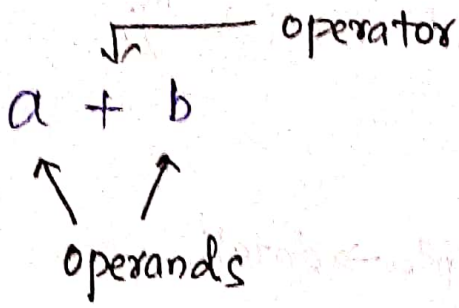
```
    else{
     printf(" Enter the number to push : ");
     scanf("%d", &temp->data);
     temp->next = NULL; /* optional */
     temp->next = top;  /* Mandatory */
     top = temp;
    }
}

void pop()
{
    if (top== NULL)
      {
       printf(" Stack Underflow \n");
      }
    else{
        struct node *temp;
        temp = top;
        printf(" %d popped \n", temp->data);
        top = top->next;
        temp->next = NULL;
        free (temp);
      }
}
```

```
void peek()
{
  if ( top == NULL)
    {
     printf(" Stack is empty \n");
    }
  else{
        struct node * temp;
        temp = top;
    printf (" %d is the TOP of stack \n", temp->data);
    }
}
void display()
  {
    if (top == NULL)
      {
      printf (" No elements in stack \n");
      }
   else {
     struct node * temp;
      temp = top ;
    printf (" Stack elements are: \n");
     while ( temp != NULL)
      {
       printf ("%d \n", temp -> data);
        temp = temp -> next;
      }
  } }
```

Applications of stack

1. Reversal of a string
2. Checking the validity of an Arithmetic Expression
3. Conversion of infix expression to postfix form.
4. Conversion of prefix expression to postfix form.
5. Conversion of postfix expression to prefix form.
6. Conversion of postfix expression to infix form.
7. Conversion of infix expression to prefix form.
8. Evaluation of Postfix expression.
9. Towers of Hanoi
10. Recursion
    (i) Tail Recursion
    (ii) Non-Tail Recursion
    (iii) Indirect Recursion
    (iv) Nested Recursion
11. Function calls



infix : a+b
postfix : ab+
prefix : +ab
} order of operands are always same.

The prefixes "pre-", "in-" and "post-" refers to the relative position of the operator with respect to its operands.

If the operator is placed before its two operands, then the expression is in prefix form.

If the operator is placed in the middle, it is known as infix form.

If the operator is placed ~~after~~ after two operands, then it is known as postfix form.

Prefix : +AB (operator before its operand)
Infix : A+B (operator in the middle of its operands)
Postfix : AB+ (operator after its operands)

NOTE: Prefix notation is also known as Polish Notation.
Postfix notation is also known as Reverse Polish Notation.

Symbol	Priority	Associativity
+, −	1 (Lowest)	Left-Right
*, /, %	2	Left-Right
↑ or ∧	3	Right-Left
( )	Highest	Left-Right

Example (1) Infix : a + (b * c) (1) (2)

∴ Postfix is abc*+

∴ Prefix is +a*bc

Example (2) Infix : (a+b)(1) − c (2)

∴ Postfix is ab+c−

∴ Prefix is −+abc

Q. Translate the following infix expression into its equivalent postfix expression.

(i) (A+B)/D ∧ ((E−F)+G)

(ii) A * (B+D)/E − F * (G + H/K)

(iii) A + (B*C − (D/E ∧F) * G) * H

(iv) (A−B)/C * ((C − D/C + D))

(v) (A+B) * D + E/(F+G+D)

(20) $(A+B)/D \wedge ((E-F)+G)$

Solution:→ $(A+B)/D \wedge ((E-F)+G)$

$= AB+/D \wedge ((E-F)+G)$

$= AB+/D \wedge (EF-+G)$

$= AB+/D \wedge EF-G+$

$= AB+/DEF-G+\wedge$

$= AB+DEF-G+\wedge/$

$A * (B+D)/E - F * (G + H/K)$

Solution:→ $A * (B+D)/E - F * (G+H/K)$

$= A * BD+/E - F * (G+H/K)$

$= A * BD+/E - F * (G+HK/)$

$= A * BD+/E-F * GHK/+$

$= ABD+*/E-F * GHK/+$

$= ABD+*E/-F * GHK/+$

$= ABD+*E/-FGHK/+*$

$= ABD+*E/FGHK/+* -$

$A + (B * C - (D / E \wedge F) * G) * H$

Solution:→ $A + (B * C - (D / E \wedge F) * G) * H$

$= A + (B * C - (D / EF\wedge) * G) * H$

$= A + (B * C - DEF\wedge / * G) * H$

$= A + (BC* - DEF\wedge / * G) * H$

$= A + (BC* - DEF\wedge / G*) * H$

$= A + BC*DEF\wedge / G* - * H$

$= A + BC*DEF\wedge / G* - H*$

$= ABC*DEF\wedge / G* - H* +$

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

Solution:→ $(A-B)/C * ((C-D/C+D))$

$= AB- / C * ((C-D/C+D))$

$= AB- / C * ((C-DC/+D))$

$= AB- / C * ((CDC/-+D))$

$= AB- / C * CDC/-D+$

$= AB-C/ * CDC/-D+$

$= AB-C/CDC/-D+*$

(22) $(A+B)*D+E/(F+G+D)$

Solution:→ $(A+B)*D+E/(F+G+D)$

$= AB+*D+E/(F+G+D)$

$= AB+*D+E/(FG++D)$

$= AB+*D+E/FG+D+$

$= AB+D*+E/FG+D+$

$= AB+D*EFG+D+/+$

Translate the following infix expression into its equivalent prefix expression.

(i) $(A+B)/D \wedge ((E-F)+G)$

(ii) $A * (B+D)/E - F * (G + H/K)$

(iii) $A + (B * C - (D/E \wedge F) * G) * H$

(iv) $(A-B)/C * ((C-D/C+D))$

(v) $(A+B) * D + E/(F+G+D)$

Soln (i) $(A+B)/D \wedge ((E-F)+G)$

$= +AB/D \wedge ((E-F)+G)$

$= +AB/D \wedge (-EF+G)$

$= +AB/D \wedge +-EFG$

$= +AB/ \wedge D+-EFG$

$= /+AB \wedge D+-EFG$

(iii) $A * (B+D)/E - F * (G + H/K)$

$= A * +BD/E - F * (G + /HK)$

$= A * +BD/E - F * +G/HK$

$= *A+BD/E - F * +G/HK$

$= /*A+BDE - F * +G/HK$

$= /*A+BDE - *F+G/HK$

$= -/*A+BDE*F+G/HK$

(iii) $A + (B * C - (D / E \wedge F) * G) * H$

$= A + (B * C - (D / \wedge EF) * G) * H$

$= A + (B * C - / D \wedge EF * G) * H$

$= A + (* BC - / D \wedge EF * G) * H$

$= A + (* BC - * / D \wedge EFG) * H$

$= A + - * BC * / D \wedge EFG * H$

$= A + * - * BC * / D \wedge EFGH$

$= + A * - * BC * / D \wedge EFGH$

(iv) $(A - B) / C * ((C - D / C + D))$

$= - AB / C * ((C - D / C + D))$

$= - AB / C * ((C - / DC + D))$

$= - AB / C * ((- C / DC + D))$

$= - AB / C * + - C / DC D$

$= / - ABC * + - C / DC D$

$= * / - ABC + - C / DCD$

(v) $(A+B)*D+E/(F+G+D)$

$= +AB*D+E/(F+G+D)$

$= +AB*D+E/(+FG+D)$

$= +AB*D+E/++FGD$

$= *+ABD+E/++FGD$

$= *+ABD+/E++FGD$

$= +*+ABD/E++FGD.$

Infix to postfix Conversion using Stack

Steps involved :->

1. Scan the expression from left to right.
   We shall get a symbol which may be an operand or a parenthesis (opening or closing) or an operator.
2. (i) If the symbol is an operand, then add it to the postfix expression.
   (ii) If the symbol is an opening parenthesis, then Push it onto the stack.
   (iii) If the symbol is an operator, then check the top of the stack.
3. If the precedence of the operator at the top of the stack is higher or the same as the current operator then repeatedly it is popped and added to the postfix expression, otherwise it is pushed onto the stack.
4. If the symbol is a closing parenthesis, then repeatedly pop from the stack and add each operator to the postfix expression until the corresponding opening parenthesis is encountered.
5. Remove the opening parenthesis from the stack.

(2.b)

Q. Consider the expression

$$5 * (6+2) - (12/4)$$

Convert the above infix expression to postfix expression using stack.

Solution:->

Symbol Scanned	Stack	Postfix Expression
5	[ ]	5
*	*	5
(	*, (	5
6	*, (	5, 6
+	*, (, +	5, 6
2	*, (, +	5, 6, 2
)	*	5, 6, 2, +
−	−	5, 6, 2, +, *
(	−, (	5, 6, 2, +, *
12	−, (	5, 6, 2, +, *, 12
/	−, (, /	5, 6, 2, +, *, 12
4	−, (, /	5, 6, 2, +, *, 12, 4
)	−	5, 6, 2, +, *, 12, 4, /
		5, 6, 2, +, *, 12, 4, /, −

∴ Postfix Expression is 5, 6, 2, +, *, 12, 4, /, −.

Q Convert A + (B * C - (D / E - F) * G) * H into postfix form showing stack status after every step in tabular form.

Soln

Symbol	Stack	Postfix Expression
A		A
+	+	A
(	+, (	A
B	+, (	A, B
*	+, (, *	A, B
C	+, (, *	A, B, C
-	+, (, -	A, B, C, *
(	+, (, -, (	A, B, C, *
D	+, (, -, (	A, B, C, *, D
/	+, (, -, (, /	A, B, C, *, D
E	+, (, -, (, /	A B, C, *, D, E
-	+, (, -, (, -	A B, C, *, D, E, /
F	+, (, -, (, -	A, B, C, *, D, E, /, F
)	+, (, -	A, B, C, *, D, E, /, F, -
*	+, (, -, *	A, B, C, *, D, E, /, F, -
G	+, (, -, *	A, B, C, *, D, E, /, F, -, G
)	+	A B, C, *, D, E, /, F, -, G, *, -
*	+, *	A, B, C, *, D, E, /, F, -, G, *, -
H	+, *	A, B, C, *D, E, /, F, -G, *, -H
		A, B, C, *D, E, /, F, -G, *, -H*, +

∴ Postfix Expression is ABC*DE/F-G*-H*+

20 Q. Consider the following infix expression:

$((A+B)*D)\wedge(E-F)$

Find its equivalent postfix expression using stack.

Symbol	Stack	Expression
(	(	
(	((	
A	((	A
+	(, (, +	A
B	(, (, +	A, B
)	(	A, B, +
*	(, *	A, B, +
D	(, *	A, B, +, D
)	Empty	A, B, +, D, *
^	^	A, B, +, D, *
(	^, (	A, B, +, D, *
E	^, (	A, B, +, D, *, E
-	^, (, -	A, B, +, D, *, E
F	^, (, -	A, B, +, D, *, E, F
)	^	A, B, +, D, *, E, F, -
	Empty	A, B, +, D, *, E, F, -, ^

∴ Postfix Expression is A, B, +, D, *, E, F, -, ^.

Q Convert following infix to Postfix Expression :→

A + B / C * (D + E) − F

Solution :→

Symbol	Stack	Postfix Expression
A	Empty	A
+	+	A
B	+	A, B
/	+, /	A, B
C	+, /	A, B, C
*	+, *	A, B, C, /
(	+, *, (	A, B, C, /
D	+, *, (	A, B, C, /, D
+	+, *, (, +	A, B, C, /, D
E	+, *, (, +	A, B, C, /, D, E
)	+, *	A, B, C, /, D, E, +
−	−	A, B, C, /, D, E, +, *, +
F	−	A, B, C, /, D, E, +, *, +, F
	Empty	A, B, C, /, D, E, +, *, +, F, −

∴ Postfix Expression is ABC/DE+*+F−

Q3 Convert the Following infix to Postfix Expression:
A ^ B * C / (D * E - F)

Solution

Symbol	Stack	Postfix Expression
A	Empty	A
^	^	A
B	^	A, B
*	*	A, B, ^
C	*	A, B, ^, C
/	/	A, B, ^, C, *
(	/, (	A, B, ^, C, *
D	/, (	A, B, ^, C, *, D
*	/, C, *	A, B, ^, C, *, D
E	/, (, *	A, B, ^, C, *, D, E
-	/, (, -	A, B, ^, C, *, D, E, *
F	/, (, -	A, B, ^, C, *, D, E, *, F
)	/	A, B, ^, C, *, D, E, *, F, -
	Empty	A, B, ^, C, *, D, E, *, F, -, /

Q. Convert following infix expression to postfix expression.

A-B/C^D*E+F/G

Solution:->

Symbol	Stack	Postfix Expression
A	Empty	A
-	-	A
B	-	A,B
/	-,/	A,B
C	-,/	A,B,C
^	-,/,^	A,B,C
D	-,/,^	A,B,C,D
*	-,*	A,B,C,D,^,/
E	-,*	A,B,C,D,^,/,E
+	+	A,B,C,D,^,/,E,*,-
F	+	A,B,C,D,^,/,E,*,-,F
/	+,/	A,B,C,D,^,/,E,*,-,F
G	+,/	A,B,C,D,^,/,E,*,-,F,G
	Empty	A,B,C,D,^,/,E,*,-F,G,/,+

∴ Postfix Expression is : ABCD^/E*-FG/+

Q Convert given infix expression to postfix expression.

$8 * (5 \wedge 4 + 2) - 6 \wedge 2 / (9 * 3)$

Soln

Symbol	Stack	Postfix Expression
8	Empty	8
*	*	8
(	*, (	8
5	*, (	8, 5
^	*, (, ^	8, 5
4	*, (, ^	8, 5, 4
+	*, (, +	8, 5, 4, ^
2	*, (, +	8, 5, 4, ^, 2
)	*	8, 5, 4, ^, 2, +
−	−	8, 5, 4, ^, 2, +, *
6	−	8, 5, 4, ^, 2, +, *, 6
^	−, ^	8, 5, 4, ^, 2, +, *, 6
2	−, ^	8, 5, 4, ^, 2, +, *, 6, 2
/	−, /	8, 5, 4, ^, 2, +, *, 6, 2, ^
(	−, /, (	8, 5, 4, ^, 2, +, *, 6, 2, ^
9	−, /, (	8, 5, 4, ^, 2, +, *, 6, 2, ^, 9
*	−, /, (, *	8, 5, 4, ^, 2, +, *, 6, 2, ^, 9
3	−, /, (, *	8, 5, 4, ^, 2, +, *, 6, 2, ^, 9, 3
)	−, /	8, 5, 4, ^, 2, +, *, 6, 2, ^, 9, 3, *
	Empty	8, 5, 4, ^, 2, +, *, 6, 2, ^, 9, 3, *, /, −

∴ Postfix Expression : 8, 5, 4, ^, 2, +, *, 6, 2, ^, 9, 3, *, /, −

Q Convert given infix expression to postfix expression. (3)

Soln $7+5*3\wedge2/(9-2\wedge2)+6*4$

Symbol	Stack	Postfix Expression
7		7
+	+	7
5	+	7,5
*	+,*	7,5
3	+,*	7,5,3
^	+,*,^	7,5,3
2	+,*,^	7,5,3,2
/	+,/	7,5,3,2,^,*
(	+,/,(	7,5,3,2,^,*
9	+,/,(	7,5,3,2,^,*,9
-	+,/,(,-	7,5,3,2,^,*,9
2	+,/,(,-	7,5,3,2,^,*,9,2
^	+,/,(,-,^	7,5,3,2,^,*,9,2
2	+,/,(,-,^	7,5,3,2,^,*,9,2,2
)	+,/	7,5,3,2,^,*,9,2,2,^,-
+	+	7,5,3,2,^,*,9,2,2,^,-,/,+
6	+	7,5,3,2^,*,9,2,2,^,-,/,+,6
*	+,*	7,5,3,2,^,*,9,2,2,^,-,/,+,6
4	+,*	7,5,3,2,^,*,9,2,2,^,-,/,+,6,4
		7,5,3,2,^,*,9,2,2,^,-,/,+,6,4,*,+

∴ Postfix Expression is 7532^*922^-/+64*+

Evaluation of Postfix Expression Using Stack

Steps :→

1. If an operand is encountered, push it on STACK.
2. If an operator 'op' is encountered, then
   (i) Pop two elements of STACK, where A is the top element and B is the next top element.
   (ii) Evaluate B op A.
   (iii) Push the result on STACK.
3. The evaluated value is equal to the value at top of STACK.

Q. Evaluate the postfix Expression 5 6 2 + * 12 4 / − using stack.

Soln

Symbol	Stack	Postfix Expression (B op A)
5	5	
6	5,6	
2	5,6,2	
+	5,8	[6+2] (A=2, B=6)
*	40	[5*8] (A=8, B=5)
12	40,12	
4	40,12,4	
/	40,3	[12/4] (A=4, B=12)
−	37	[40−3] (A=3, B=40)

∴ Answer = 37.

8 Evaluate 12 7 3 - / 2 1 5 + * + using stack.

Soln

Symbol	Stack	Operation (B op A)
12	12	
7	12, 7	
3	12, 7, 3	
-	12, 4	[7-3] (A=3, B=7)
/	3	[12-4] (A=4, B=12)
2	3, 2	
1	3, 2, 1	
5	3, 2, 1, 5	
+	3, 2, 6	[1+5] (A=5, B=1)
*	3, 12	[2*6] (A=6, B=2)
+	15	[3+12] (A=12, B=3)

∴ Answer = 15

Q Evaluate 7 5 3 2 ^ * 9 2 2 ^ - / + 6 4 * +
using stack.

Symbol	Stack	Operation (B op A)
7	7	
5	7, 5	
3	7, 5, 3	
2	7, 5, 3, 2	
^	7, 5, 9	[3 ^ 2] (A=2, B=3)
*	7, 45	[5 * 9] (A=9, B=5)
9	7, 45, 9	
2	7, 45, 9, 2	
2	7, 45, 9, 2, 2	
^	7, 45, 9, 4	[2 ^ 2] (A=2, B=2)
-	7, 45, 5	[9 - 4] (A=4, B=9)
/	7, 9	[45 / 5] (A=5, B=45)
+	16	[7 + 9] (A=9, B=7)
6	16, 6	
4	16, 6, 4	
*	16, 24	[6 * 4] (A=4, B=6)
+	40	[16 + 24] (A=24, B=16)

∴ Answer = 40

Q Evaluate 5 9 3 / 2 1 + * + 6 2 / - 3 + using STACK.

Symbol	Stack	Operation (B op A)
5	5	
9	5,9	
3	5,9,3	
/	5,3	[9/3] (A=3, B=9)
2	5,3,2	
1	5,3,2,1	
+	5,3,3	[2+1] (A=1, B=2)
*	5,9	[3*3] (A=3, B=3)
+	14	[5+9] (A=9, B=5)
6	14,6	
2	14,6,2	
/	14,3	[6/2] (A=2, B=6)
-	11	[14-3] (A=3, B=14)
3	11,3	
+	14	[11+3] (A=3, B=11)

∴ Answer = 14

Q Evaluate 4 3 2 ^ + 1 8 * 2 2 + / - 2 - using stack.

Symbol	Stack	Operation (B op A)
4	4	
3	4,3	
2	4,3,2	
^	4,9	[3 ^ 2] (A=2, B=3)
+	13	[4 + 9] (A=9, B=4)
1	13,1	
8	13,1,8	
*	13,8	[1 * 8] (A=8, B=1)
2	13,8,2	
2	13,8,2,2	
+	13,8,4	[2+2] (A=2, B=2)
/	13,2	[8/4] (A=4, B=8)
-	11	[13-2] (A=2, B=13)
2	11,2	
-	9	[11-2] (A=2, B=11)

∴ Answer = 9

26

Q Evaluate 2 3 ^ 1 - 4 2 / 6 * + 3 1 + 2 / - using stack.

Sol

Symbol	Stack	Operation (B op A)
2	2	
3	2,3	
^	8	[2^3] (A=3, B=2)
1	8,1	
-	7	[8-1] (A=1, B=8)
4	7,4	
2	7,4,2	
/	7,2	[4/2] (A=2, B=4)
6	7,2,6	
*	7, 12	[2*6] (A=6, B=2)
+	19	[7+12] (A=12, B=7)
3	19,3	
1	19,3,1	
+	19,4	[3+1] (A=1, B=3)
2	19,4,2	
/	19,2	[4/2] (A=2, B=4)
-	17	[19-2] (A=2, B=19)

∴ Ans = 17

Infix Expression to Prefix Expression Using Stack

Steps involved :=>

1. Scan the expression from Right to Left.
   We shall get a symbol which may be an operand or a parenthesis (opening or closing) or an operator.
2. (i) If the symbol is an operand, add it to the postfix expression.
   (ii) If the symbol is a closing parenthesis, then Push it onto the stack.
   (iii) If the symbol is an operator, then check the top of the stack.
3. If the precedence of the operator at the top of the stack is higher as the current operator then repeatedly it is popped and added to the prefix expression otherwise it is pushed onto the stack.
4. If the symbol is an opening parenthesis, then repeatedly Pop from the stack and add each operator to the prefix expression until the corresponding closing parenthesis is encountered.
5. Reverse the output expression to get the Final prefix Expression for the given Infix Expression.

(35) Q Convert infix Expression 2 * 3 / (2-1) + 5 * (4-1) into prefix form using stack.

Soln Infix : 2 * 3 / (2-1) + 5 * (4-1)

Reversed Expression :) 1 - 4 (* 5 +) 1 - 2 (/ 3 * 2

Symbol	Stack	Prefix Expression (Right to Left)
)	)	
1	)	1
-	), -	1
4	), -	1, 4
(	Empty	1, 4, -
*	*	1, 4, -
5	*	1, 4, -, 5
+	+	1, 4, -, 5, *
)	+,)	1, 4, -, 5, *
1	+,)	1, 4, -, 5, *, 1
-	+,), -	1, 4, -, 5, *, 1
2	+,), -	1, 4, -, 5, *, 1, 2
(	+	1, 4, -, 5, *, 1, 2, -
/	+, /	1, 4, -, 5, *, 1, 2, -
3	+, /	1, 4, -, 5, *, 1, 2, -, 3
*	+, /, *	1, 4, -, 5, *, 1, 2, -, 3
2	+, /, *	1, 4, -, 5, *, 1, 2, -, 3, 2
	Empty	1, 4, -, 5, *, 1, 2, -, 3, 2, *, /, +

Reverse output string : + / * 2 3 - 2 1 * 5 - 4 1

∴ Prefix Expression is + / * 2 3 - 2 1 * 5 - 4 1

Q Convert given infix expression to prefix expression using stack. $(8+2)/2*3-2/1+3$

Soln Infix Expression : $(8+2)/2*3-2/1+3$

Reverse infix expression : $3+1/2-3*2/)2+8($

Symbol	Stack	Prefix Expression (Right to Left)
3	Empty	3
+	+	3
1	+	3,1
/	+,/	3,1
2	+,/	3,1,2
−	+,−	3,1,2,/
3	+,−	3,1,2,/,3
*	+,−,*	3,1,2,/,3
2	+,−,*	3,1,2,/,3,2
/	+,−,*,/	3,1,2,/,3,2
)	+,−,*,/,)	3,1,2,/,3,2
2	+,−,*,/,)	3,1,2,/,3,2,2
+	+,−,*,/,),+	3,1,2,/,3,2,2
8	+,−,*,/,),+	3,1,2,/,3,2,2,8
(	+,−,*,/	3,1,2,/,3,2,2,8,+
	Empty	3,1,2,/,3,2,2,8,+,/,*,−,+

Reverse output string: +,−,*,/,+,8,2,2,3,/,2,1,3

∴ Prefix Expression: +,−,*,/,+,8,2,2,3,/,2,1,3

Q) Convert given infix expression to prefix expression using stack.

6/3+3*2^2+1.

Soln Infix Expression: 6/3+3*2^2+1

Reversed Infix Expression: 1+2^2*3+3/6

Symbol	Stack	Prefix Expression (Right-to-left)
1	Empty	1
+	+	1
2	+	1,2
^	+,^	1,2
2	+,^	1,2,2
*	+,*	1,2,2,^
3	+,*	1,2,2,^,3
+	+,+	1,2,2,^,3,*
3	+,+	1,2,2,^,3,*,3
/	+,+,/	1,2,2,^,3,*,3
6	+,+,/	1,2,2,^,3,*,3,6
	Empty	1,2,2,^,3,*,3,6,/,+,+

Reverse output expression: ++/63*3^221

∴ Prefix Expression: ++/63*3^221.

Q Convert following infix expression to equivalent prefix expression using stack.

$$A-(B+C)*D/E$$

Sol Infix Expression: $A-(B+C)*D/E$

Reversed Infix Expression: $E/D*)C+B(-A$

Symbol	Stack	Postfix Expression (Right-to-Left)
E	Empty	E
/	/	E
D	/	E, D
*	/, *	E, D
)	/, *,)	E, D
C	(, *,)	E, D, C
+	(, *,), +	E, D, C
B	/, *,), +	E, D, C, B
(	/, *	E, D, C, B, +
—	—	E, D, C, B, +, *, /
A	—	E, D, C, B, +, *, /, A
	Empty	E, D, C, B, +, *, /, A, —

Reverse Output Expression: —, A, /, *, +, B, C, D, E

Prefix Expression: —, A, /, *, +, B, C, D, E

Q Convert following infix expression to prefix expression using stack.

H ^ (J + K) * I % S

Soln Prefix Expression : H ^ (J + K) * I % S

Reverse Prefix Expression : S % I *) K + J (^ H

Symbol	Stack	Prefix Expression (Right-to-left)
S		S
%	%	S
I	%	S, I
*	%, *	S, I
)	%, *,)	S, I
K	%, *,)	S, I, K
+	%, *,), +	S, I, K
J	%, *,), +	S, I, K, J
(	%, *	S, I, K, J, +
^	%, *, ^	S, I, K, J, +
H	%, *, ^	S, I, K, J, +, H
		S, I, K, J, +, H, ^, *, %

Reverse Output Expression : %, *, ^, H, +, J, K, I, S

Prefix Expression : % * ^ H + J K I S