Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Department of Information Technology

Chapter-2
Stack

Introduction to Stack, Stack as ADT, Operations on stack, Application of stack: – reversing string, Polish notations

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

UNIT - 2

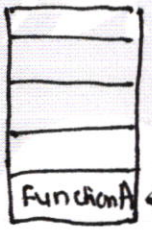
Stacks

A stack is an important data structure which is extensively used in computer applications.
Stack is an important data structure which stores its elements in an ordered manner.
You must have seen a pile of plates where one plate is placed on top of another as shown in following fig. Now when you want to remove a plate, you remove the topmost plate first Hence you can add and remove an element only at / from one position which is the topmost position.

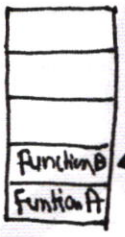
A stack is linear data structure which uses the same principle ie the elements in a stack are added and removed only from one end, which is called the TOP Hence, a stack is called a LIFO (Last-In-First-Out) data structure, as the element that was inserted last is the first one to be taken out.

Now question is where do we need stacks in computer science? The answer is in function calls. Consider an example

where we are executing function A. In the course of its execution, function A calls anather function B. Function B in turn calls another function C, which calls function D.



When A calls B, A is pushed on top of the system stack. When the execution of B is complete, the system control will remove A from the stack and continue with its execution



When B calls C, B is pushed on top of the system stack. When the execution of C is complete, the system control will remove B from the stack and continue with its execution

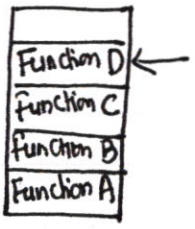
When C calls D, C is pushed on top of the system stack. When the execution of D is complete, the system control will remove C from the stack and continue with its execution.

When D calls E, D is pushed on top of the system stack. When the execution of E is complete, the system control will remove D from stack and continue with its execution.

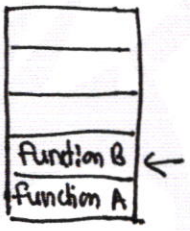
System Stack in the case of function calls

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

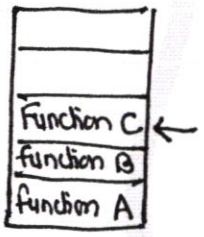
Semester: III Subject: DSA Academic Year: 2017-18



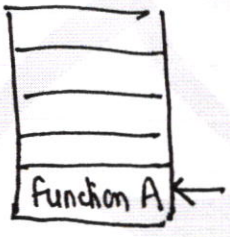
When E has executed, D will be removed for execution



When C has executed, B will be removed for execution



When D has executed C will be removed for execution



When B has executed, A will be removed for execution.

System stack when a called function returns to the calling function.

In order to keep track of returning point of each active function a special stack called system stack or call stack is used. Whenever a function calls another function, the calling function is pushed onto the top of the stack. This is because after the called function gets executed the control is passed back to the calling function.

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

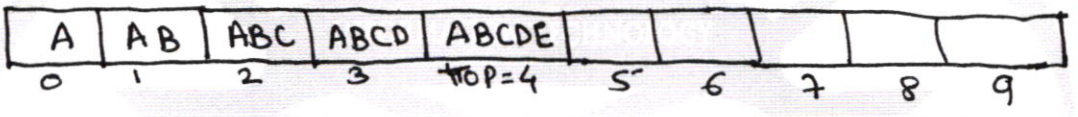
Semester: III Subject: DSA Academic Year: 2017-18

ARRAY REPRESENTATION OF STACKS

In the computer's memory, stacks can be represented as a linear array. Every stack has a variable called TOP associated with it, which is used to store the address of the topmost element of the stack. It is this position where the element will be added to or deleted from. There is another variable called MAX, which is used to store the maximum number of element that the stack can hold.

If TOP = NULL then it indicates that the stack is empty and if TOP = MAX - 1, then the stack is full.

You must be wondering why we have written MAX - 1. It is because array indices start from 0.



The stack in above figure shows that TOP = 4, so insertions and deletion will be done at this position. In the above stack five more elements can still be stored.

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

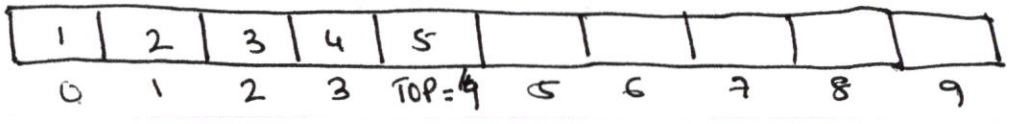
Semester: III Subject: DSA Academic Year: 2017-18

OPERATIONS ON A STACK

A stack supports three basic operation: push, pop and peek. The push operation adds an element to the top of the stack and the pop operation removes the element from the top of the stack. The peek operation returns the value of the topmost element of the stack.

Push Operation

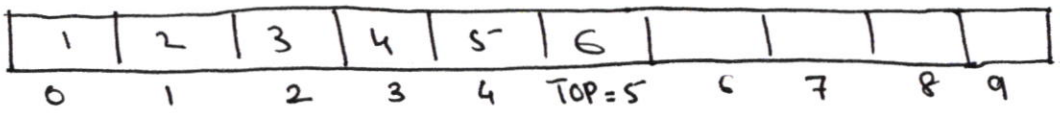
The push operation is used to insert an element into the stack. The new element is added at the topmost position of the stack. However, before inserting the value, we must first check if TOP = MAX - 1, because if that is the case, then the stack is full and no more insertions can be done. If an attempt is made to insert a value in a stack that is already full, an OVERFLOW message is printed. Consider the stack given below



To insert an element with value 6, we first check if TOP = MAX - 1. If the condition is false, then we increment the value of TOP and store the new element at the position given by stack [TOP], Thus, the updated stack becomes as shown below

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: III Subject: DSA Academic Year: 2017-18



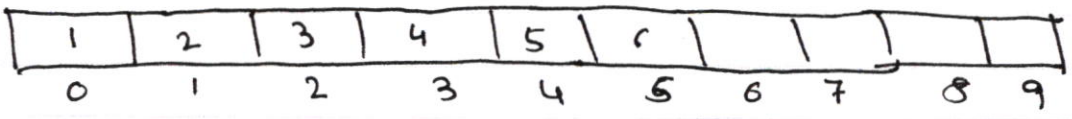
Stack after insertion

```
Step 1 : IF TOP = MAX -1
           PRINT "OVERFLOW"
           Goto step 4
         [END OF IF]
Step 2 : SET TOP = TOP + 1
Step 3 : SET STACK[TOP] = VALUE
Step 4 : END
```

ALgorithm to insert on element in a stack

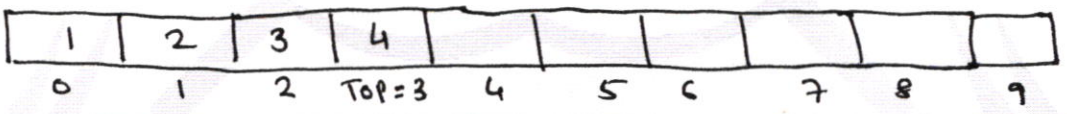
Pop Operation

The pop operation is used to delete the top most element from the stack. However, before deleting the value, we must first check if TOP = NULL because if that is the case, then it means the stack is empty and no more deletions can be done. if an attempt is made to delete a value from a stack that is already empty, an UNDERFLOW message is printed. Consider the stack given below



To delete the top most element, we first check if TOP = NULL. If the condition is false

Then we decrement the value pointed by TOP thus, the updated stack becomes as shown below



stack after deletion

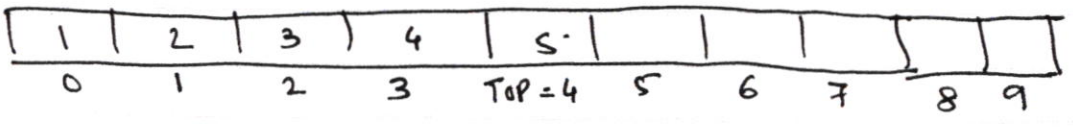
```
Step 1  : IF TOP = NULL
            PRINT " UNDERFLOW "
            Goto Step 4
          [ END OF IF ]
Step 2  : SET VAL = STACK [TOP]
Step 3  : SET TOP = TOP - 1
Step 4  : END
```

Algorithm to delete an element from a stack.

Peek Operation

Peek is an operation that returns the value of the topmost element of the stack without deleting it from the stack. The algorithm for peak operation is given below

However the peak operation first check, if checks if the stack is empty i.e if TOP = NULL, then an appropriate message is printed, else the value is retured. Consider the stack given



Implementation of stack menu driven program.

Source Code

```
#include<stdio.h>
#define STACKSIZE 5

struct bufferstack
{
int stk[STACKSIZE];
int top; // We will use it as POINTER to top of the stack
}s; // Here s is struct variable, you can see here how to implement structure in C

void push(); // To push elements in stack
int  pop();  // To Pop elements in stack
void display();

int main()
{
int c;
s.top=-1; //Set s to -1
int x=1;
while(x)  //While loop to keep program in loop
  {
printf("\n TOP is at : %d\t",s.top+1); //this line is to test, the position of s
printf("\n ****MENU****\n");
printf("1: Push \n");
printf("2: Pop \n");
```

```
printf("3: Display \n");
printf("4: Exit");
printf("\n Please enter your choice : ");
scanf("%d",&c);
   switch(c)
   {

   case 1:
      push();
      break;
   case 2:
      pop();
      break;
   case 3:
      display();
      break;
   case 4:
      return 0;
   }
printf(" \n Press 1 to continue..... ");
scanf("%d",&x);
  }
}

void push() //function to insert element in stack
{
   int num;
   printf("\n ***PUSH OPERATION***");
   if(s.top==(STACKSIZE-1))//check for overflow condition
   {
      printf("\n Sorry You can't push any element into stack .... ,Stack is full");
   }
   else //if stack is not full then you can insert element with else part
   {
   printf("\n Enter the number to push into stack:-\t");
   scanf("%d",&num);
   s.top+=1; //increment value of top pointer
   s.stk[s.top]= num; //now store element at top most position of stack

   }
}

int pop()  //function to delete element from stack
{
   printf("\n ***POP OPERATION***");
   int num;
   if(s.top==-1)//check for underflow condition
   {
   printf("\nstack is empty ");
   }
   else //if stack is not empty then you can delete top most element from stack
   {
   num=s.top;//store top most element from stack in num
   printf("\n Poped number is : %d\t",s.stk[num]);// display value of num
   s.stk[num]=0; //To delete top element from stack
   s.top=s.top-1;//decrement value of top pointer
```

```
    }
    return num;

}

void display()
{
    int i;
    printf("\n ***DISPLAY OPERATION***");
    if(s.top==-1)//check if stack is empty
    {
    printf("\n Stack is empty \n");
    }
    else//if stack is not empty display the contents of the stack
    {
    for(i=s.top;i>=0;i--)// start for loop from top of stack till 0th element
        {
        printf("\n%d : %d",i,s.stk[i]);

        }
     printf("\tTOP=%d",s.top);
     }
 }
```

OUTPUT: 1. PUSH the data on Stack

```
apsit@apsit-CQ3551IX:~/Desktop$ gcc stack.c
apsit@apsit-CQ3551IX:~/Desktop$ ./a.out

 TOP is at : 0
 ****MENU****
1: Push
2: Pop
3: Display
4: Exit
 Please enter your choice : 1

 ***PUSH OPERATION***
 Enter the number to push into stack:-  10

 Press 1 to continue.... 1
```

2. Display contents of stack

```
 Please enter your choice : 3

 ***DISPLAY OPERATION***
0 : 10  TOP=0
 Press 1 to continue.... 1
```

3. Pop element from the stack

```
 Please enter your choice : 2

 ***POP OPERATION***
 Poped number is : 10
 Press 1 to continue....
```

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: III Subject: DSA Academic Year: 2017-18

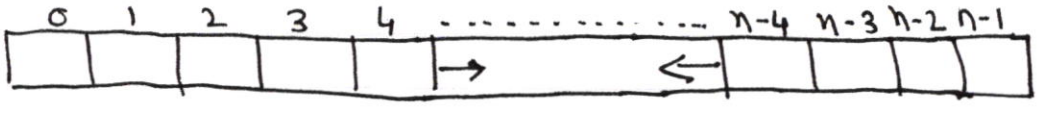
```
STEP 1 :   IF TOP = NULL
           PRINT "STACK IS EMPTY"
           GOTO step 3
STEP 2 :   RETURN STACK[TOP]
STEP 3 :   END
```

Algorithm for Peek operation.

MULTIPLE STACKS

If the stack is allocated less space then frequent OVERFLOW conditions will be encountered. To deal with this problem the code will have to be modified to reallocate more space for the array. In case we allocate a large amount of space for the stack, it may result in sheer wastage of memory. Thus, there lies a trade-off between the frequency of overflows and the space allocated.

So, a better solution to deal with this problem is to have multiple stacks or to have more than one stack in the same array of sufficient size



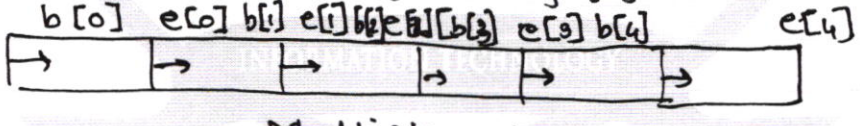
Multiple Stacks

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: III Subject: DSA Academic Year: 2017-18

In above fig an array STACK[n] is used to represent two stacks, Stack A and Stack B The value of n is such that the combined size of both the stacks will never exceed n. While operating on these stacks, it is important to note one thing - Stack A will grow from left to right, whereas Stack B will grow from right to left at the same time.

Extending this concept to multiple stacks, a stack can also be used to represent n number of stacks in the same array. That is, if we have a STACK[n], then each stack I will be allocated an equal amount of space bounded by indices b[i] and e[i] This is shown in following fig



Multiple stacks.

APPLICATION OF STACKS

//This is a C Program to Implement two Stacks using a Single Array & Check for Overflow & Underflow

```
#include <stdio.h>

#define SIZE 10
int ar[SIZE];
int top1 = -          1;
int top2 = SIZE;

    //Function to push data in stack1
void push_stack1 (int data)
{
 if (top1 < top2 -                1)
 {
         top1++;
  ar[top1] = data;
 }
 else
 {
  printf(          "Stack Full! Cannot Push\n");
 }
}

     //Function to push data in stack2
void push_stack2 (int data)
{
 if (top1 < top2 -                1)
 {
         top2--;
  ar[top2] = data;
 }
 else
 {
  printf(          "Stack Full! Cannot Push\n");
 }
}
    //Function to pop data from stack1
void pop_stack1()
{
 if (top1 >=             0)
 {
  int popped_value = ar[top1];
         top1--;
  printf (          "%d is being popped from Stack 1\n", popped_value);
 }
 else
 {
  printf (          "Stack Empty! Cannot Pop\n");
 }
}

    //Function to pop data from stack2
void pop_stack2()
{
 if (top2 < SIZE)
 {
  int popped_value = ar[top2];
```

```
        top2++;

    printf ("%d is being popped from Stack 2\n", popped_value);
  }
  else
  {
    printf (          "Stack Empty! Cannot Pop\n");
  }

}

    //Function to Print contents of Stack 1
void print_stack1()
{
  int i;
  printf(          "\n Stack1 contents: ");
  for (i = top1; i >=                0; i--)
  {
    printf (          "%d ", ar[i]);
  }

  printf (          "\n");
}

    //Function to Print contents of Stack 2
void print_stack2()
{
int i;
printf(        "\n Stack2 contents: ");
for (i = top2; i < SIZE; i++)
{
    printf (          "%d ", ar[i]);
 }
        printf (      "\n");
}

int main()
{
  int ar[SIZE];
  int i;
  int num_of_ele;

  printf (          "We can push a total of 10 values\n");

  //Number of elements pushed in stack 1 is 6

  //Number of elements pushed in stack 2 is 4
 for(i = 1; i <= 6; ++i)
 {
   push_stack1 (i);
   printf (          "Value Pushed in Stack 1 is %d\n", i);
 }

 for (i =          1; i <= 4; ++i)
 {
   push_stack2 (i);
   printf (          "Value Pushed in Stack 2 is %d\n", i);
 }
      //Print Both Stacks
 print_stack1 ();
 print_stack2 ();

      //Pushing on Stack Full
 printf ("Pushing Value in Stack 1 is %d\n", 11);
 push_stack1 (          11);

 //Popping All Elements From Stack 1
 num_of_ele = top1 + 1;
 while (num_of_ele)
 {
   pop_stack1 ();
   --num_of_ele;
```

```
    }
        //Trying to Pop From Empty Stack
    pop_stack1 ();
    return          0;
}
```

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

APPICATIONS OF STACKS

In this section we will discuss typical problems where stacks can be easily applied for a simple and efficient solution. The topics that will be discussed in this section include the following

- Reversing a list
- Parentheses check
- Conversion of an infix expression into a postfix expression
- Evaluation of a postfix expression
- Conversion of an infix expression into a prefix expression
- Evaluation of a prefix expression
- Recursion
- Tower of Hanoi

Reversing a List

A List of numbers can be reversed by reading each number from an array starting from the first index and pushing it on a stack. Once all the numbers have been read, the numbers can be popped one at a time and then stored in the array starting from the first index.

Implementing Parentheses Checker

Stacks can be used to check the validity of parantheses in any algebraic expression. for example, an algebraic expression is valid if for every open bracket there is a corresponding closing bracket. for example, the expression (A+B} is invalid but an expression { A + (B-C)} is valid. Look at the program below which traverses an algebraic expression to check for its validity.

Polish Notations

Infix, postfix and prefix notations are three different but equivalent notations of writting algebraic expression. But before learning about prefix and postfix notations, let us first see what an infix notation is.

while writing an arithmatic expression using infix notation, the operator is placed in between the operands. For example, A + B, here, plus operator is placed between the two operands A and B. Although it is easy for us to write expression using infix notation, computers find it difficult to parse as the computer needs a lot of information to evaluate the expression. Information is needed about operator precedence and

associativity rules, and brackets which overide these rules.

So computers work more efficiently with expressions written using prefix and postfix notation.

In postfix notation, as the name suggests, the operator is placed after the operands. for example, if an expression is written as A+B in infix notation, the same expression can be written as AB+ in postfix notation. The order of evaluation of a postfix expression is always from left to right. Even brackets cannot alter the order of evaluation

The expression (A+B) * C can be written [AB+] *C

AB+C* in the postfix notation

Example :-

Convert the following infix expression into postfix expression

Solution -

ⓐ (A-B) * (C+D)

[AB-] * [CD+]

AB-CD+*

ⓑ (A+B) / (C+D) - (D*E)

[AB+] / [CD+] - [DE*]

[AB+CD+/] - [DE*]

AB+CD+/DE*-

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Although a prefix notation is also evaluated from left to right, the only difference between a postfix notation, the operator is placed before the operands. For example, if A+B is an expression in infix notation, then the corresponding expression in prefix notation is given by +AB

While evaluating a prefix expression the ~~operand~~ operators are applied to the operands that are present immediately on the right of operator. Like postfix, prefix expression also do not follow the rules of operator precedence and associativity and even brackets cannot alter the order of evaluation.

Convesion of an Infix Expression into a Postfix Expression

Let I be ~~can~~ an algebraic expression written in infix notation. I may contain parentheses, Operands, and operators. For simplicity of the algorithm we will use only +, -, *, /, % operators. The precedence of these operator can be given as

Higher priority *, /, %
Lower priority +, -

Convert the following infix expression into prefix expression

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: III Subject: DSA Academic Year: 2017-18

Solution :-

(a) (A+B) * C
(+AB) * C
* + ABC

(b) (A-B) * (C+D)
[-AB] * [+CD]
* -AB+CD

(c) (A+B) / (C + D) - CD * E)
[+AB] / [+CD] - [*DE]
[/ +AB+CD] - [* DE]
- / + AB + CD * DE

The algorithm given below transforms an infix expression into postfix expression. The algorithm uses a stack to temporarily hold operators. The postfix expression is obtained from left to right using the operands from the infix expression and the operators which are removed from the stack. The first step in algorithm is to push a left parenthesis on the stack and to add a corresponding right parenthesis at the end of the infix expression. The algorithm is repeated until the stack is empty.

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Step 1 : Add ")" to the end of the infix expression
Step 2 : Push "(" on to the stack
step 3 : Repeat until each character in the infix notation is scanned.
If a "C" is encountered, push it on the stack
If an operand (whether a digit or a character) is encountered, add it to the postfix expression.
If a ")" is encountered, then
a. Repeatedly pop from stack and add it to the postfix expression until a "C" is encountered.
b. Discard the "C". That is, remove the "C" from stack and do not add it to the postfix expression
If an operator O is encountered, then
a. Repeatedly pop from stack and add each operator (popped from the stack) to the postfix expression which has the same precedence or higher precedence than O
b. Push the operator O to the stack
[END OF IF]
step 4 : Repeatedly pop from the stack and add it to the postfix expression until the stack is empty
Step 5 : EXIT

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: III Subject: DSA Academic Year: 2017-18

Convert the following infix expression into postfix expression using the algorithm given above

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

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

Infix character Scanned	stack	Postfix Expression
	C	
A	C	A
-	C -	A
(	C - C	A
B	C - C	AB
/	C - C /	AB
C	C - C /	ABC
+	C - C +	ABC/
(	C - C + C	ABC/
D	C - C + C	ABC/D
%	C - C + C %	ABC/D
E	C - C + C %	ABC/DE
*	C - C + C % *	ABC/DE
F	C - C + C % *	ABC/DEF
)	C - C +	ABC/DEF*%
/	C - C + /	ABC/DEF*%
G	C - C + /	ABC/DEF*% G
)	C -	ABC/DEF*%G/+
*	C - *	ABC/DEF*%G/+
H	C - *	ABC/DEF*%G/+H
)		ABC/DEF*%G/+H*

Semester: III Subject: DSA Academic Year: 2017-18

```
/*This is a C Program for conversion of infix to postfix expression and its evaluation*/
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
#define max 20

struct stack //strucutre of stack to store operators
{
  int top;
  char stack[max];//stack stores string values
}s;

void push(char x)//push function to push operators on stack
{
  if(s.top>=max-      1)//check if stack is full
  printf("\n\nOVERFLOW ! ! !");
  else    //if stack is not empty push the operator
 {
  s.top++;
  s.stack[s.top]=x;
 }
}

char pop()    //pop function to pop operators from stack
{
  char x;
  if(s.top==(-       1))
  printf(       "\n\nUNDERFLOW ! ! !");
  else
 {
  x=s.stack[s.top];
  s.top--;
 }
  return x;     //return the top most element from stack
}

char stacktop()           //return tpye changed(int->char)
{
  if(s.top==-      1)
   return -       1;
  else
   return s.stack[s.top];
}

int isoperand(char c)           //check if current element is operand i.e. in between A to Z
{
  if((c>=    'A' && c<='Z')||(c>='a' && c<='z'))
  return       1;
  else
  return       0;
}

int op(char x)       // assign prorities to operators
{
  int r;
  switch(x)
```

```
{
 case    '(' :r=1;
              break;

 case    '+' :
 case    '-' :r=2;
              break;

 case    '*' :
 case    '/' :r=3;
              break;

 case    '^' :r=4;
              break;
}
 return r;
}

int prcd(char a,char b)          //compare priorities of operators
{
 if(op(a)>=op(b))
  return       1;
 else
  return       0;
}
void main()
{
 char ip[    40],out[40],temp;
 int i,j,a,x,op1,op2,result;
s.top=-    1; //set top of stack to -1
printf("\nEnter infix expression : ");
scanf(   "%s",ip);
 for(i=   0,j=0;ip[i]!='\0';i++)
{
 if(ip[i]==          '(') //element is "(" then push the element.
 {
  push(ip[i]);
 }
 else
 if(ip[i]==          ')')//element is ")" then pop the elements till ")" or stack isempty
 {
  while(s.top!=-          1 && s.stack[s.top]!='(')
  {
   out[j++]=pop();
  }
  if(s.top==-          1)// if we provide only ")" then it is an incorrect expression
  {
   printf(         "\nINCORRECT EXPRESSION");
   break;
  }
  temp=pop();
 }
 else       //element is an operand then no need to push just store in out[]
 if(isoperand(ip[i]))
 {
  out[j++]=ip[i];
      //i++;
 }
 else       //element is an operator
 {
  a=prcd(stacktop(),ip[i]);              //get precedence of the operator and compare with
                                         //precedence of operator present at stack top
  if(a==0)
   push(ip[i]);          //if the precedence is lower push the operator


  else //if the precedence is higer then pop the operators till stack is empty                    //or we get an
opertor having precedence higher
  {
   while(stacktop()!=-          1 && prcd(stacktop(),ip[i])!=0)
    out[j++]=pop();
    push(ip[i]);
```

```
  }//end of else
 }//end of else
}//end of for loop

 while(s.top!=-1)/*store all the remaining elements from stack to out[]*/                    out[j++]=pop();/*and
make stack empty*/

out[j]='\0';//put "/0" at the end of the string

printf("\nPost-fix Expression : %s\n",out);
 int n=strlen(out);

printf(      "\n Evalution of the expression\n");
 for(i=    0;i<n;i++)
{
 if(isalpha(out[i]))              //if element is alphabet
 {
  printf(        "\nEnter a value for the variable %c :",out[i]);
  scanf(      "%d",&x);
  push(x);       //push the value of var
 }
 else      //if element is operator
 {
  op2=pop();       //pop value of 1st var
  op1=pop();//pop value of 2nd var

  switch(out[i])
  {
   case       '+': result=op1+op2;
                   break;
   case       '-': result=op1-op2;
                   break;
   case       '*': result=op1*op2;
                   break;
   case       '/': result=op1/op2;
                   break;
   case       '^': result=op1^op2;
                   break;
  }
  push(result);          //store result in stack
 }
}
printf(      "\nThe result is : %d\n",result);
}
```

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: III Subject: DSA Academic Year: 2017-18

Evaluation of a Postfix Expression

Using stacks, any postfix expression can be evaluated very easily. Every character of the postfix expression is scanned from left to right. If the character encountered is an operand, it is pushed on to the stack. However, if an operator is encountered, then the top two value are popped from the stack and the operator is applied on these value. The result is then pushed on to the stack.

Step 1 : Add a ")" at the end of postfix expression
step 2 : Scan every character of the postfix expression and repeat step 3 & 4 untill ")" is encountered
step 3 : If an operand is encountered push it on the stack
If an operator O is encountered, then
a. Pop the top two elements from the stack as A and B as A and B
b. Evaluate B O A, where A is the topmost element and B is the element below A
c. Push the result of evaluation on the stack
[END OF IF]
step 4 : SET RESULT equal to the topmost element of the stack
step 5 : EXIT

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Let us now take an example that makes use of this algorithm. Consider the infix expression given as 9 - ((3 * 4) + 8) / 4 . Evaluate the expression.

Character Scanned	Stack
9	9
3	9, 3
4	9, 3, 4
*	9, 12
8	9, 12, 8
+	9, 20
4	9, 20, 4
/	9, 5
−	4

Recursion :

A recursive function is defined as a function that calls itself to solve a smaller version of its task until a final call is made which does not require a call to itself. Since a recursive function repeatedly calls itself, it makes use of the system stack to temporarily store the return address and local variables of calling function.