Date ___ / ___ / ____

1> WAP to convert a given valid paranthesized infix arithmetic expression to prefix expression. The expression consists of single character operands and the binary operators +(plus), -(minus), *(multiply) and /(divide).

```
# include <stdio.h>
# include <string.h>
# define MAX 20
void infixtoprefix (char infix[20], char prefix[20]);
void reverse (char array [30]);
char pop ();
void push (char symbol);
int top = -1;
char stack [MAX];
main () {
char infix [20], prefix[20], temp;
printf ("Enter infix operation : ");
gets(infix);
infixtoprefix(infix, prefix);
reverse (prefix);
puts ((prefix));
}
void infixtoprefix (char infix [20], char prefix [20]
{
int i, j = 0;
char symbol;
stack [++ top] = '#';
reverse (infix);
for ( i = 0; i < strlen (infix); i++) {
```

Date __/__/____

```
symbol = infix [i];
if (isoperator (symbol) = = 0 )
{
~~symbol = infix [i]~~ prefix [j] = symbol;
~~if (isop~~ j++;
 } else
  {
  if ( symbol = = ')')
  {
  push (symbol);
 }
else if ( symbol = = '(' )
{
while  (stack [top] != ')') .
{
 prefix [j] = pop ();
   j++;
 }
push (symbol);
}
}
}
}
while (stack [top] != '#') {
prefix [j] = pop ();
 j++;
}
prefix [j] = '\0' ;
}

void reverse (char array [30]) {
  int i, j ;
```

Page No.

Date ___/___/______

```
~~void rever~~ char temp [100];
for ( i = strlen(array) -1, j = 0; i + 1 != 0; --i, +
 {
   temp [j] = '\0';
   strcpy(array, temp);
 }
char pop() {
char a;
a = stack [top];
top --;
return a;
}
void push (char symbol) {
top ++;
stack [top] = symbol;
}
int pred (char symbol) {
switch(symbol) {
case '+':
case '-':
     return 2;
     break;
Case '*':
Case '/':
     return 4;
     break;
Case '$':
Case '^':
   return 6;
   break;
Case '#':
```

Date ___ /___ /______

```
Case '(' :
Case ')' :
  return 1 ;
 break ;
 }
}
int isOperator (char symbol) {
switch (symbol) {
Case '+' :
 Case '-' :
 Case '*' :
Case '/' :
Case '^' :
Case '$' :
Case '&' :
Case '(' :
Case ')' :
   return 1 ;
   break ;
default :
    return 0;
 }
}
```

Date ___/___/______

2> ~~# incl~~ WAP to demonstrate the Evaluation of postfix expression

```
# include <stdio.h>
# include <math.h>
# include <ctype.h>
# include <string.h>
double compute(char symbol, double op1, double op
 {
  switch(symbol)
   {
  case '+' : return op1 + op2;
  case '-' ;  return op1 - op2;
  Case '*' : return op1 * op2;
  Case '/' : return op1 / op2;
  Case '$' :
  Case '^' ;  return pow (op1, op2);
  }
 }
void main ()
 {
 double S[20];
 double res;
 double op1, op2;
 int top, i;
char postfix [20], symbol;
printf(" Enter postfix expression );
Scanf("%s", postfix);
top = -1;
for (i=0; i< strlen(postfix); i++)
 {
```

```
Date __/__/__

symbol = postfix [i];
if (isdigit(symbol))
S [++top] = symbol-'0';
else
 {
 op2 = s[top --];
 op1 = S[top--];
 res = compute (symbol, op1, op2);
 S[++ top] = res;
 }
}
 res = s [top --]
printf ("result = %f \n", res);
}
```

Date ___ / ___ / ______

3) WAP to perform fractorial of a number using Recursion

```
# include <stdio.h>
~~int~~ int fact (int n)
{
    if (n==0)
     return 1;
     else
    return n* fact(n-1);
}
void main()
{
 int n;
printf("Enter the value of n \n");
Scanf(" %d", &n);
printf(" The factorial of %d = %d \n", n,
                        fact (n));
}
```

Date __/__/____

4) WAP to perform GCD of two numbers using Recursion.

```
# include <stdio.h>
int GCD (int , int);
int main()
{
int num 1, num 2, res;
printf ("\n Enter the two numbers :");
scanf ("%d %d", & num1 , & num2) ;
res = GCD (num1, num 2);
printf ("\n GCD of %d and %d = %d", num1,
                    num 2, res);

return 0;
}
int GCD (int x , int y)
{
int rem;
rem = x% y ;
if ( rem == 0)
return y;
else
return ( GCD ( y, rem)) ;
}
```

Scanned with CamScanner