Program 4

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>

#define STK_SIZE 10

void fnPush(char [], int*, char); char
fnPop(char [], int*);
int fnPrecd(char);

int main()
{
	int i, j=0;
	char acExpr[50], acStack[50], acPost[50], cSymb;
	int top = -1;

	printf("\nEnter a valid infix expression\n");
	scanf("%s", acExpr);

	fnPush(acStack, &top, '#');
	for(i=0;acExpr[i]!='\0'; ++i)
	{
		cSymb = acExpr[i];
if(isalnum(cSymb))
		{
			acPost[j++] = cSymb;
		}
		else if(cSymb == '(')
		{
			fnPush(acStack, &top, cSymb);
		}
		else if(cSymb == ')')
		{
			while(acStack[top] != '('
```

```
                {
                        acPost[j++] = fnPop(acStack, &top);
                }
                fnPop(acStack, &top);
        }
        else
        {
                while(fnPrecd(acStack[top]) >= fnPrecd(cSymb))
                {
                        if((cSymb == '^') && (acStack[top] == '^'))
                                break;
            acPost[j++] = fnPop(acStack, &top);
                }
                fnPush(acStack, &top, cSymb);
        }

    }
    while(acStack[top] != '#')
    {
        acPost[j++] = fnPop(acStack, &top);
    }
    acPost[j] = '\0';

    printf("\nInfix Expression is %s\n", acExpr);
 printf("\nPostfix Expression is %s\n", acPost);  return 0;
}

void fnPush(char Stack[], int *t , char elem)
{
    *t = *t + 1;
    Stack[*t] = elem;

}
```

Program 5

```
void  push(int  [],  int*,  int);
int pop(int [], int*);
```

```
int main()
{
        int iastack[50], i, op1, op2, res;
 char expr[50], symb;  int top =
-1;

        printf("\nEnter a valid postfix expression\n");
        scanf("%s", expr);

        for(i=0; i<strlen(expr); i++)
        {
                symb = expr[i];

                if(isdigit(symb))

                {
                        push(iastack, &top, symb-'0');
                }
                else
                {
                        op2 = pop(iastack, &top);
                        op1 = pop(iastack, &top);
                        switch(symb)
                        {
                                case '+' :      res = op1 + op2;
                                                break;
                                case '-' :      res = op1 - op2;
                                                break;
                                case '*' :      res = op1 * op2;
                                                break;
                                case '/' :      res = op1 / op2;
                                                break;
                                case '%' :      res = op1 % op2;
                                                break;
                                case '^' :      res = (int)pow(op1 , op2);
                                                break;
                        }
                        push(iastack, &top, res);
                }
        }
res = pop(iastack, &top);
 printf("\nValue of %s expression is %d\n", expr, res);  return 0;
```

```
}

void push(int Stack[], int *t , int elem)
{
        *t = *t + 1;
        Stack[*t] = elem;

}

int pop(int Stack[], int *t)
{
 int  elem;     elem  =
Stack[*t];   *t  =  *t  -1;
return elem;
}
```

**5. Develop a Program in C for the following Stack Applications**
**b. Solving Tower of Hanoi problem with n disks**

```
#include <stdio.h>
void  towers(int,  char,  char,  char);
int main()
{
   int num;
   printf("Enter the number of disks : ");
   scanf("%d", &num);
   printf("The sequence of moves involved in the Tower of Hanoi are :\n");
towers(num, 'A', 'C', 'B');     printf("\n");     return 0;
}
void towers(int num, char frompeg, char topeg, char auxpeg)
{
   if (num == 1)
   {
      printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);
return;
   }
   towers(num - 1, frompeg, auxpeg, topeg);
   printf("\n Move disk %d from peg %c to peg %c", num, frompeg, topeg);
towers(num - 1, auxpeg, topeg, frompeg);
}
```

Program 6

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <stdbool.h>

#define SIZE 5

void insert(char [], int*, int*,
char); char del(char[], int*,
int*); void display(char [], int,
int); bool qfull(int, int);
bool qempty(int, int);

int main()
{
   char q[SIZE];
int f = -1, r = -1;
int ch;     char
elem;

   for(;;)
   {
        printf("\nQueue Operations\n");
       printf("====================");

printf("\n1.Qinsert\n2.Qdelete\n3.Qdisplay\n4.Exit\n");
   printf("Enter your choice\n");
scanf("%d",&ch);      getchar();              switch(ch)
           {
                case 1: if(!qfull(f,r))
                        {
                            printf("\nEnter an
element : ");                           scanf("%c",
&elem);                         insert(q, &f, &r,
elem);
                         }
                else
                        {
                        printf("\nQueue is Full\n");
                        }
```

```
                        break;

                case 2: if(!qempty(f, r))
                        {                               elem = del(q,
&f, &r);
                       printf("\nDeleted element is %c\n", elem);
                        }
                 else
                        {
                        printf("\nQueue is Empty\n");
                        }
                         break;
                case 3: if(!qempty(f, r))
                        {
              printf("\nContents of the Queue is \n");
display(q, f, r);
                         }
                 else
                        {
                        printf("\nQueue is Empty\n");
                        }
                         break;
                 case 4: exit(0);

               default: printf("\nInvalid choice\n");
                         break;
          }
   }
return 0;
}

bool qfull(int fr, int rr)
{
   if((rr+1) % SIZE == fr)
       return true;
else
       return false;
}
```

```
bool qempty(int fr, int rr)
{
   if(fr == -1)
return     true;
else
       return false;
}

void insert(char queue[], int *f, int *r, char val)
{
   if(*r == -1)
   {
      *f = *f + 1;
      *r = *r +
1;    }
   else        *r = (*r +
1)%SIZE;

   queue[*r] = val;
}

char del(char queue[], int *f, int *r)
{    char
el;
   el = queue[*f];

   if(*f == *r)
   {
      *f = -1;
      *r = -
1;    }
else
   {
      *f = (*f + 1)%SIZE;
   }
return el;
}
```

```
void display(char queue[], int fr, int rr)
{
   int i;
if(fr<=rr)
{
       for(i=fr; i<=rr; i++)
       {
          printf("%c\t", queue[i]);
       }
printf("\n");
}     else
   {
       for(i=fr; i<=SIZE-1; i++)
       {
          printf("%c\t", queue[i]);
       }         for(i=0;
i<=rr; i++)
       {
          printf("%c\t", queue[i]);
       }
printf("\n");
   }
}
```