Name – Shrinivas Hatyalikar
Div – CS-B
Roll no – 24

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

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


void push(char x){
    struct node *t;
    t=(struct node *)malloc(sizeof(struct node));

    if(t==NULL){
        printf("Stack is full");
    }
    else{
        t->data=x;
        t->next=top;
        top=t;
    }
}

void pop(){
    struct node *t;
    char x=-1;
    if(top==NULL){
        //printf("Stack is empty");
        return;
    }
    else{
        t=top;
        top=top->next;
```

```
        x=t->data;
        free(t);
    }
    //return x;
}

int isOperand(char x){
    if(x>='0' && x<='9') return 1;
    if(x>='a' && x<='z') return 1;
    if(x>='A' && x<='Z') return 1;
    return 0;
}
int IsOpeningParentheses(char C)
{
    if(C == '(' || C == '{' || C=='[')
        return 1 ;
    return 0;
}

int IsClosingParentheses(char C)
{
    if(C == ')' || C == '}' || C==']')
        return 1 ;
    return 0;
}

int isoperand(char C)
{
          if(C >= '0' && C <= '9') return 1;
          if(C >= 'a' && C <= 'z') return 1;
          if(C >= 'A' && C <= 'Z') return 1;
          return 0;
}
int isoperator(char c)
{
    if(c=='+' || c=='*' || c=='-' || c=='/' || c=='^') return 1;
    return 0;
}

int priority(char x){
```

```
    if(x=='+'|| x=='-'){
        return 1;
    }
    else if(x=='*'|| x=='/'){
        return 2;
    }
    else if(x=='^'){
        return 3;
    }
    return 0;
}

int peek(){
    if(top==NULL) return 0;
    return top->data;
}

char* infixtopostfix(char *A, int n)
{
    char *postfix = (char*)malloc((n+1)*sizeof(char));
    int j=0;
    for(int i=0;i<n;i++)
    {
        if(A[i] == ' ' || A[i] == ',')
            continue;
        if(isoperand(A[i])>0){
            postfix[j++]=A[i];
        }
        else if(isoperator(A[i])>0)
        {
            while(top!=NULL && !IsOpeningParentheses(peek()) &&
priority(peek())>=priority(A[i]))
            {
                postfix[j++]=peek();
                pop();
            }
            push(A[i]);
        }
        else if(IsOpeningParentheses(A[i]))
        {
```

```
            push(A[i]);
        }
        else if(IsClosingParentheses(A[i]))
        {
            while(top!=NULL && !IsOpeningParentheses(peek())) {
                postfix[j++]=peek();
                pop();
            }
            if (top != NULL && IsOpeningParentheses(peek()))
                pop();
        }

    }
    while (top != NULL && !IsOpeningParentheses(peek())) {
        postfix[j++] = peek();
        pop();
    }
    postfix[j]='\0';
    printf("Postfix:%s\n",postfix);
    return postfix;
}


void reverse (char *exp)
{

  int size = strlen (exp);
  int j = size, i = 0;
  char temp[size];

  temp[j--] = '\0';
  while (exp[i] != '\0')
    {
      temp[j] = exp[i];
      j--;
      i++;
    }
  strcpy (exp, temp);
}
```

```
void brackets (char *exp)
{
  int i = 0;
  while (exp[i] != '\0')
    {
    if (exp[i] == '(')
              exp[i] = ')';
    else if (exp[i] == ')')
        exp[i] = '(';
      i++;
    }
}

void infixtoprefix(char *infix, int n){
    reverse(infix);
    brackets(infix);
    char *prefix = (char*)malloc((n+1)*sizeof(char));
    int j=0;
    for(int i=0;i<n;i++)
    {
        if(infix[i] == ' ' || infix[i] == ',')
            continue;
        if(isoperand(infix[i])>0){
            prefix[j++]=infix[i];
        }
        else if(isoperator(infix[i])>0)
        {
            if(priority(peek())>priority(infix[i]))
            {
                while(top!=NULL && !IsOpeningParentheses(peek()))
                {
                    prefix[j++]=peek();
                    pop();
                }
            }
            push(infix[i]);
        }
        else if(IsOpeningParentheses(infix[i]))
                    {
                                push(infix[i]);
```

```
            }
            else if(IsClosingParentheses(infix[i]))
            {
                while(top!=NULL && !IsOpeningParentheses(peek())) {
                    prefix[j++]=peek();
        pop();
                }
                pop();
            }

    }
    while (top != NULL && !IsOpeningParentheses(peek())) {
        prefix[j++] = peek();
        pop();
    }
    if (top != NULL && IsOpeningParentheses(peek()))
        pop();
    prefix[j]='\0';


    reverse(prefix);

    printf("Prefix:%s\n",prefix);
}

int main(){

    char arr[100];
    printf("Infix: ");
    int n;
    gets(arr);
    n=strlen(arr);
    infixtopostfix(arr,n);
    infixtoprefix(arr,n);
}
```

```
PS C:\Users\sheeh\OneDrive\Desktop\C\output> & .\'infixtopostfix.exe'
Infix: x^y/(5*z)+2
Postfix:xy^5z*/2+
Prefix:+/^xy*5z2
PS C:\Users\sheeh\OneDrive\Desktop\C\output>
```