Source Code:-

```
/*
 Name - Neevan Redkar
 Class - SE
 Batch S3
 Roll no 3069
*/
#include <iostream>
#include <cctype>
#include <algorithm>
using namespace std;



struct node{ //defined struct node
  char data;
  struct node *next;
};

struct evalNode{
  double dat;
  struct evalNode *next;
};

node* createNode(char input){
  node* new_node = (node*)malloc(sizeof(node));
  new_node->data = input;
  new_node->next = NULL;
  return new_node;
}
evalNode* createNode(double input){
  evalNode* new_node = (evalNode*)malloc(sizeof(evalNode));
  new_node->dat = input;
  new_node->next = NULL;
  return new_node;
}
struct Stack{ //define stack
      node* head;
};

struct evalStack{
  evalNode* head;
};



string in2post(Stack*);
  string in2pre(Stack*);
  double evalPost(evalStack*);
  double evalPre(evalStack*);

  int main(){
    Stack stack;
    evalStack stack1;
    cout<<"Enter choice of operation\n1.Infix to postfix\n2.Infix to Prefix";
    cout<<"\n3.Evaluate postfix\n4.Evaluate prefix"<<endl;
    int choice;
    cin>>choice;
    switch(choice){
      case 1:
      cout<<in2post(&stack)<<endl;
      break;
      case 2:
      cout<<in2pre(&stack)<<endl;
      break;
      case 3:
      cout<<evalPost(&stack1)<<endl;
      break;
      case 4:
      cout<<evalPre(&stack1)<<endl;
      break;
      default:
      cout<<"Option not yet available/invalid input"<<endl;;

    }

      return 0;
  }

  void push(Stack* stack,char input){
    node *new_node = createNode(input);
    new_node->next=stack->head;
    stack->head= new_node;
  }

  void push(evalStack* stack,double input){
    evalNode *new_node = createNode(input);
    new_node->next=stack->head;
    stack->head= new_node;
  }

  char pop(Stack* stack){
    char t = stack->head->data;
    node* temp = stack->head;
    stack->head = (stack->head)->next;
  free(temp);
  return t;
}

double pop(evalStack* stack){
  double t = stack->head->dat;
  evalNode* temp = stack->head;
  stack->head = (stack->head)->next;
  free(temp);
  return t;
}


string in2post(Stack* stack){
  string input;
  cout<<"Enter infix expression"<<endl;
  cin>>input;
  string exp="";
  int counter=0;


  for(int i=0;i<input.length();i++){
    if(isdigit(input[i])||isalpha(input[i])){
      exp.append(1,input[i]);

    }else if(input[i]=='+'||input[i]=='-
'||input[i]=='*'||input[i]=='/'||input[i]=='('||input[i]==')'){
      exp+=' ';
      if((stack->head->data=='*'||stack->head-
>data=='/')&&(input[i]=='+'||input[i]=='-')){
       while(counter>0){
        if(stack!=NULL){
          exp.push_back(pop(stack));
          counter--;
        }
       }
       push(stack,input[i]);
       counter++;
      }else if(input[i]==')'){
        while(stack->head->data!='('){
          exp.push_back(pop(stack));
          counter--;
       }
        char t=pop(stack);
        counter--;

      }else{
        push(stack,input[i]);
        counter++;
        }
      }
    }
    while(counter>0){
          if(stack!=NULL){
            exp.push_back(pop(stack));
            counter--;
          }
    }

    return exp;

  }

  string in2pre(Stack* stack){
    string input;
    cout<<"Enter infix expression"<<endl;
    cin>>input;
    reverse(input.begin(),input.end());
    for(int i=0;i<input.length();i++){
      if(input[i]==')'){
        input[i]='$';
      }else if(input[i]=='('){
        input[i]='@';
      }
    }
    for(int i=0;i<input.length();i++){

    }

    for(int i=0;i<input.length();i++){
      if(input[i]=='$'){
        input[i]='(';
      }else if(input[i]=='@'){
        input[i]=')';
      }
      }
    cout<<"Reversed input is"<<input<<endl;
    string exp="";
    int counter=0;


    for(int i=0;i<input.length();i++){
      if(isdigit(input[i])||isalpha(input[i])){
        exp.append(1,input[i]);

      }else if(input[i]=='+'||input[i]=='-
'||input[i]=='*'||input[i]=='/'||input[i]=='('||input[i]==')'){
      exp+=' ';
      if((stack->head->data=='*'||stack->head-
>data=='/')&&(input[i]=='+'||input[i]=='-')){
       while(counter>0){
        if(stack!=NULL){
          exp.push_back(pop(stack));
          counter--;
        }
       }
       push(stack,input[i]);
       counter++;
      }else if(input[i]==')'){
        while(stack->head->data!='('){
          exp.push_back(pop(stack));
          counter--;
       }
        char t=pop(stack);
        counter--;

      }else{
        push(stack,input[i]);
        counter++;
      }
    }
  }
  while(counter>0){
        if(stack!=NULL){
          exp.push_back(pop(stack));
          counter--;
        }
  }
  reverse(exp.begin(),exp.end());
  return exp;



}

double evalPost(evalStack* stack){
  string input;
  cout<<"Enter postfix expression to be evaluated"<<endl;
  cin>>input;
  double answer=0;
  for(int i=0;i<input.length();i++){
    if(isdigit(input[i])){
      double x = input[i] -48;
      push(stack,x);
      }else if(input[i]=='+'||input[i]=='-'||input[i]=='*'||input[i]=='/'){
        double a = pop(stack);
        double b = pop(stack);
        switch(input[i]){
          case '+':
          answer=a+b;
          break;
          case '-':
          answer=a-b;

          break;
          case '*':
          answer=(a*b);

          break;
          case '/':
          answer=(a/b);

          break;
        }
        push(stack,answer);
      }
    }
    if(answer< 0){
      return pop(stack)*-1;
    }else{
    return pop(stack);
  }
  }

  double evalPre(evalStack* stack){
    string input;
    cout<<"Enter prefix expression to be evaluated"<<endl;
    cin>>input;
    reverse(input.begin(),input.end());
    double answer=0;
    for(int i=0;i<input.length();i++){
      if(isdigit(input[i])){
        double x = input[i] -48;
        push(stack,x);

      }else if(input[i]=='+'||input[i]=='-'||input[i]=='*'||input[i]=='/'){
        double a = pop(stack);
        double b = pop(stack);
        switch(input[i]){
          case '+':
          answer=a+b;
          break;
```

```
            case '-':
            answer=a-b;

            break;
            case '*':
            answer=(a*b);

            break;
            case '/':
            answer=(a/b);

            break;
          }
          push(stack,answer);
        }
      }
      if(answer< 0){
        return pop(stack)*-1;
      }else{
      return pop(stack);
    }
    }
```

Output :-

```
neevsr@DESKTOP-VQKL5KK:/mnt/c/Users/AR/Documents/Assignments/DSA/Assignments/Assignment 2$ ./Stack

Enter choice of operation

1.Infix to postfix

2.Infix to Prefix

3.Evaluate postfix

4.Evaluate prefix

1

Enter infix expression

5*(3+4)

5  3 4 +*

neevsr@DESKTOP-VQKL5KK:/mnt/c/Users/AR/Documents/Assignments/DSA/Assignments/Assignment 2$ ./Stack

Enter choice of operation

1.Infix to postfix

2.Infix to Prefix
```

```
3.Evaluate postfix

4.Evaluate prefix

2

Enter infix expression

5*(3+4)

Reversed input is(4+3)*5

*5 + 3 4

neevsr@DESKTOP-
VQKL5KK:/mnt/c/Users/AR/Documents/Assignments/DSA/Assignments/Assignment 2$ ./Stack

Enter choice of operation

1.Infix to postfix

2.Infix to Prefix

3.Evaluate postfix

4.Evaluate prefix

3

Enter postfix expression to be evaluated

534+*

35

neevsr@DESKTOP-
VQKL5KK:/mnt/c/Users/AR/Documents/Assignments/DSA/Assignments/Assignment 2$ ./Stack

Enter choice of operation

1.Infix to postfix

2.Infix to Prefix

3.Evaluate postfix

4.Evaluate prefix

4

Enter prefix expression to be evaluated

*5+34

35
```