Q1Write a program to perform Parenthesis matching in an expression.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include<iostream>

using namespace std;

struct stack

{

    int size;

    int top;

    char \*arr;

};

int isEmpty(struct stack \*ptr)

{

    if (ptr->top == -1)

    {

        return 1;

    }

    else

    {

        return 0;

    }

}

int isFull(struct stack \*ptr)

{

    if (ptr->top == ptr->size - 1)

    {

        return 1;

    }

    else

    {

        return 0;

    }

}

void push(struct stack\* ptr, char val){

    if(isFull(ptr)){

        cout<<"Stack Overflow! Cannot push "<<val<<" to the stack"<<endl;

    }

    else{

        ptr->top++;

        ptr->arr[ptr->top] = val;

    }

}

char pop(struct stack\* ptr){

    if(isEmpty(ptr)){

        cout<<"Stack Underflow! Cannot pop from the stack"<<endl;

        return -1;

    }

    else{

        char val = ptr->arr[ptr->top];

        ptr->top--;

        return val;

    }

}

int parenthesisMatch(char \* exp){

    // Create and initialize the stack

    struct stack\* sp;

    sp->size = 100;

    sp->top = -1;

    sp->arr = (char \*)malloc(sp->size \* sizeof(char));

    for (int i = 0; exp[i]!='\0'; i++)

    {

        if(exp[i]==('('||'['||'{')){

            push(sp, exp[i]);

        }

        else if(exp[i]==(')'||']'||'}')){

            if(isEmpty(sp)){

                return 0;

            }

            pop(sp);

        }

    }

    if(isEmpty(sp)){

        return 1;

    }

    else{

        return 0;

    }

}

int main()

{

    char \* exp = "{[(14+6)\*(2+7)]/16}";

    if(parenthesisMatch(exp)){

        cout<<"The parenthesis is matching";;

    }

    else{

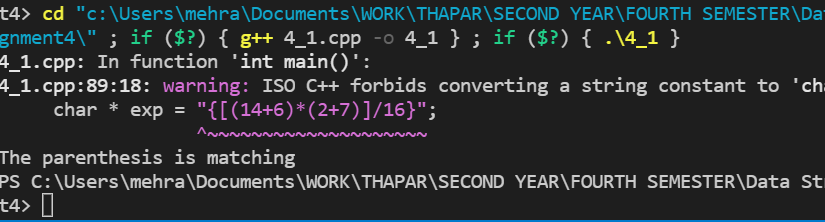
        cout<<"The parenthesis is not matching";

    }

    return 0;

}

**OUTPUT:**

****

Q2. What is the value of the following postfix expression?

54 6 + 7 4 - \* 9 / 35 15 + +

**CODE:**

#include<iostream>

#include<string.h>

using namespace std;

#define MAX 100

//these are globally declared as every function can use them

char stack[MAX]; //to store the whole postfix expression

char postfix[MAX];

int top=-1;//indicates the index of the topmost element of the stack;

void push(int data)

{

    if(top==MAX-1)

    {

        cout<<"Stack overflow\n";

        return;

    }

    top++;

    stack[top]=data;

}

int pop()

{

    if(top<0)

    {

        cout<<"Stack underflow\n";

        return 0;

    }

    else

    return stack[top--];

}

float post\_eval()

{

    int i,a,b;

    for( i=0;i<strlen(postfix);i++)

    {

        //because we are comparing the characters and not the numbers, we are

        //actually comparing the ASCII codes of 0-9

        if(postfix[i] >= '0' && postfix[i] <= '9')

        {

            push(postfix[i]-'0');

        }

        else

        {

            a=pop();

            b=pop();

            //we have to do b-a

            switch (postfix[i])

            {

            case '+':

                    push(b+a); break;

             case '\*':

                    push(b\*a); break;

             case '/':

                    push(b/a);break;

             case '-':

                    push(b-a);break;

             case '^':

                    push(b^a);break;

            }

        }

    }

    return pop();

}

int main()

{

    cout<<"Enter the postfix expression you want to evaluate\n";

    gets(postfix);

    float result;

    result = post\_eval();

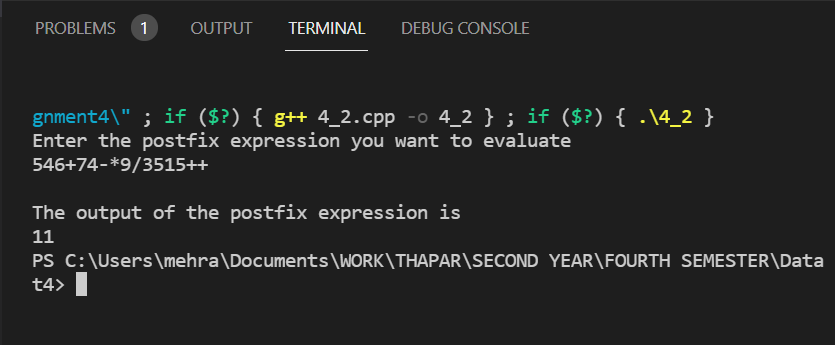
    cout<<endl;

    cout<<"The output of the postfix expression is\n"<<result;

    return 0;

}

**OUTPUT:**



Q3. Use the conversion algorithm to change the following infix expression into prefix using stack.

Show each step using a tabular approach.

(A \* B - (C - D)) / (E + F)

**CODE:**

#include<iostream>

#include<string.h>

using namespace std;

#define MAX 100

//these are globally declared as everyunction can use them

char stack[MAX]; //to store the whole postfix expression

char infix[MAX]; //infix to store the expression

char postfix[MAX];

int top=-1;//indicates the index of the topmost element of the stack;

void push(int data)

{

    if(top==MAX-1)

    {

        cout<<"Stack overflow\n";

        return;

    }

    top++;

    stack[top]=data;

}

int pop()

{

    if(top<0)

    {

        cout<<"Stack underflow\n";

        return 0;

    }

    else

    return stack[top--];

}

int isEmpty()

{

    if (top<0)

    {

        return 1;

    }

    else

    return 0;

}

int precedence(char symbol)

{

    switch(symbol)

    {

        case '^':

            return 3;

        case '/':

        case '\*':

            return 2; //this shows that for both the cases, this value will be returned

        case '+':

        case '-':

        return 1;

        default:

        return 0;

    }

}

void inToPost()

{

    char symbol,next;

    int i,j=0;

    for(i=0;i<strlen(infix);i++)

    {

        symbol = infix[i];

        //in the algo, there are many if else cases, hence it is better to just use a switch case syntax

        switch (symbol)

        {

        case '(':

            push(symbol);

            break;

        case ')':

            while ((next=pop())!='(')

            {

                postfix[j]=next;

                j++;

                //but instead of printing, we store the elements in the postfix array

            }

            //our job was also to see that both the parenthesis get ppopped out of the stack and

            //this loop does ensure that

            break;

            case '-':

            case '+':

            case '\*':

            case '/':

            case '^':

                while(!isEmpty()&&(precedence(stack[top])>=precedence(symbol)))

                {

                    postfix[j++]=pop();

                }

                push(symbol);

                break;

        default:

        //if the symbol is an operand, we simply put the operand in the postfix array

        {

            postfix[j]=symbol;

            j++;

        }

        }

    }

    //we must check if the while stack is empty or not. It is an important thing to do

    while(!isEmpty())

    {

        postfix[j]=pop();

        j++;

        postfix[j]='\0';

    }

}

void print()

{

    for(int i=0;i<sizeof(postfix)/sizeof(postfix[0]);i++)

    {

        cout<<postfix[i];

    }

    cout<<endl;

}

void brackets (char \*exp)

{

  int i = 0;

  while (exp[i] != '\0')

    {

      if (exp[i] == '(')

    exp[i] = ')';

      else if (exp[i] == ')')

    exp[i] = '(';

      i++;

    }

}

int main()

{

    //get the expression from the user

    cout<<"Enter the infix expression: \n";

    gets(infix); //this is from the string module. And because it's a string there will be a "end of string" function

    //cout<<infix[0]<<endl;

    int end=strlen(infix);

    //cout<<end;

   // cout<<endl<<strrev(infix)<<endl;

    string reversed\_infix = strrev(infix);

    for(int i =0;i<strlen(infix);i++)

    {

        infix[i]=reversed\_infix[i];

    }

    // for(int i =0;i<strlen(infix);i++)

    // {

    //       if(infix[i]==')')

    //     {

    //         infix[i]='(';

    //         break;

    //     }

    // }

    cout<<endl;

    brackets(infix);

    for(int i =0;i<end;i++)

    {

        cout<<infix[i];

    }

    cout<<endl;

    //cout<<infix[0];

    //call the function to convert infix to postfix

    inToPost();

    reversed\_infix = strrev(postfix);

    // for(int i =0;i<strlen(infix);i++)

    // {

    //     reversed\_infix[i]=postfix[i];

    // }

 //   int n =sizeof(postfix)/sizeof(postfix[0]);

    //print();

    for(int i =0;i<end-1;i++)

    {

        cout<<reversed\_infix[i];

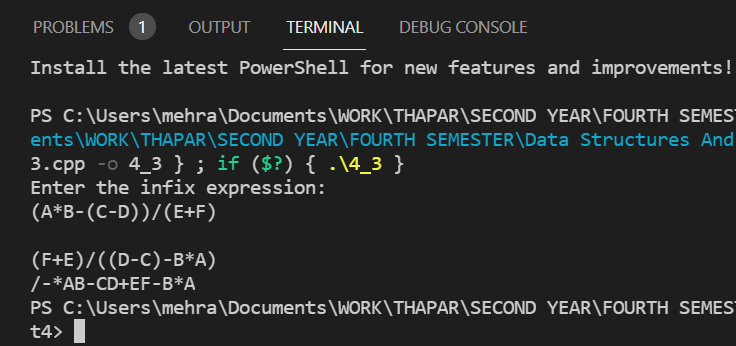
    }

    cout<<endl;

    return 0;

}

**OUTPUT:**



Q4. Implement Circular queue and implement the following functions:

 Insert or enqueue

 Remove or dequeue

 Isfull

 Isempty

Write main function to exemplify the results. Also write a main function to make the implementation a “Menu-

Driven”.

**CODE:**

#include <iostream>

#define SIZE 100

using namespace std;

class Queue {

   private:

  int items[SIZE], front, rear;

   public:

  Queue() {

    front = -1;

    rear = -1;

  }

  bool isFull() {

    if (front == 0 && rear == SIZE - 1) {

      return true;

    }

    if (front == rear + 1) {

      return true;

    }

    return false;

  }

  bool isEmpty() {

    if (front == -1)

      return true;

    else

      return false;

  }

  void enQueue(int element) {

    if (isFull()) {

      cout << "Queue is full";

    } else {

      if (front == -1) front = 0;

      rear = (rear + 1) % SIZE;

      items[rear] = element;

      cout << endl

         << "Inserted " << element << endl;

    }

  }

  int deQueue() {

    int element;

    if (isEmpty()) {

      cout << "Queue is empty" << endl;

      return (-1);

    } else {

      element = items[front];

      if (front == rear) {

        front = -1;

        rear = -1;

      }

      else {

        front = (front + 1) % SIZE;

      }

      return (element);

    }

  }

  void display() {

    int i;

    if (isEmpty()) {

      cout << endl

         << "Empty Queue" << endl;

    } else {

      cout << "Front -> " << front;

      cout << endl

         << "Items -> ";

      for (i = front; i != rear; i = (i + 1) % SIZE)

        cout << items[i];

      cout << items[i];

      cout << endl

         << "Rear -> " << rear;

    }

  }

};

int main() {

  Queue q;

  q.deQueue();

  q.enQueue(1);

  q.enQueue(2);

  q.enQueue(3);

  q.enQueue(4);

  q.enQueue(5);

  q.enQueue(6);

  q.display();

  int elem = q.deQueue();

  if (elem != -1)

    cout << endl

       << "Deleted Element is " << elem;

  q.display();

  q.enQueue(7);

  q.display();

  q.enQueue(8);

  int num;

  do{

      cout<<"Which operation do you want to run?\n";

      cout<<"1. enQueue\n";

      cout<<"2. deQueue\n";

      cout<<"3. isEmpty\n";

      cout<<"4. isFull\n";

      cout<<"5. EXIT\n";

      cin>> num;

      if(num==5)

      {break;}

  switch(num)

  {

      case 1:

        {

            cout<<"eNter the number to enqueue\n";

            int x;

            cin>>x;

            q.enQueue(x);

            cout<<endl;

            q.display();

            cout<<endl;

            break;

        }

        case 2:

        {

            q.deQueue();

            cout<<endl;

            q.display();

            cout<<endl;

            break;

        }

        case 3:

        {

            q.isEmpty();

            cout<<endl;

            q.display();

            cout<<endl;

            break;

        }

        case 4:

        {

            q.isFull();

            cout<<endl;

            q.display();

            cout<<endl;

            break;

        }

  }

}

    while(num!=5);

  return 0;

}

**OUTPUT:**

