**EXPERIMENT NUMBER: –** **6**

|  |
| --- |
| **STUDENT’S NAME: -** Vaibhav Ahuja **STUDENT’S UID: -** 19BCS1065  **BRANCH: -** CSE **SECTION AND GROUP**: - 1/B  **SUBJECT NAME: -** Data Structure Lab **DATE OF PERFOMANCE: -** 13/9/2020  **SEMESTER: -** 3  **SUBJECT CODE: -** CSP-231 |

**AIM/OVERVIEW OF PRACTICAL**–

Write a program to find infix to postfix notation of **a+b\*c+(d\*e).**

**TASK TO BE DONE**—

Making a program using stack to find postfix notation of infix notation: - **a+b\*c+(d\*e).**

**ALGORITHM/FLOWCHART—**

**Step 1: -** Scan expression left to right**.**

**Step 2: -** for all the character in expression left to right repeat step 3 to step 11

**Step 3: -** if char is equal to operand(any alphabet) print it.

**Step 4: -** If operator (‘+’ ; ‘-‘ ; ‘\*’ ; ‘/’ ; ‘^’ ) arrives and stack is empty push operator onto . stack.

**Step 5: -** If incoming operator has higher precedence than the operator present at the . top of the stack, push it onto the stack. ( precedence of ‘(‘ is set to -1)

**Step 6: -** If the incoming operator has lower precedence than the top of the stack, then . POP and print the top element of the stack and repeat from step-3 again.

**Step 7: -** If incoming operator has equal precedence with top of stack do step 8 and step . 9 else go to step 10.

**Step 8: -** If operator is power operator ( ‘ ^ ‘ ) push incoming operator onto stack.

**Step 9: -** else POP and print the top of the stack and then push the incoming operator.

**Step 10: -** If incoming symbol is ‘(‘ PUSH it onto the stack

**Step 11: -** if incoming symbol is ‘)’ POP and print the elements in the stack until ‘(‘ is found, and the POP ‘(‘ from the stack without printing.

**Step 12: -** At the end of the expression POP and print all the elements from the stack.

**CODE: -**

#include<iostream>

#include<stack>

using namespace std;

int precedence(char pre)

{

if(pre == '^')

return 3;

else if(pre == '\*' || pre == '/')

return 2;

else if(pre == '+' || pre == '-')

return 1;

else

return -1;

}

string infix\_to\_postfix(string infix)

{

  string postfix;

  stack <char> s;

  for(int i=0;i<infix.length();i++)

  {

    if((infix[i] >= 'a' && infix[i] <= 'z')||(infix[i] >= 'A' && infix[i] <= 'Z'))

    {

      postfix+=infix[i];

    }

    else if(infix[i]=='(')

    {

      s.push('(');

    }

    else if(infix[i]==')')

    {

      while((s.top()!='(') && (!s.empty()))

      {

        char temp=s.top();

        postfix+=temp;

        s.pop();

      }

      if(s.top()=='(')

      {

        s.pop();

      }

    }

    else if(infix[i]=='+' || infix[i]=='-' || infix[i]=='\*' || infix[i]=='/' || infix[i]=='^')

    {

      if(s.empty())

      {

        s.push(infix[i]);

      }

      else

      {

        if(precedence(infix[i])>precedence(s.top()))

        {

          s.push(infix[i]);

        }

        else if((precedence(infix[i])==precedence(s.top()))&&(infix[i]=='^'))

        {

          s.push(infix[i]);

        }

        else

        {

          while((!s.empty())&&( precedence(infix[i])<=precedence(s.top())))

          {

            postfix+=s.top();

            s.pop();

          }

          s.push(infix[i]);

        }

      }

    }

  }

  while(!s.empty())

  {

    postfix+=s.top();

    s.pop();

  }

  return postfix;

}

int main()

{

  string infix\_1,postfix\_1;

  cout<<"Enter a infix expression"<<endl;

  cin>>infix\_1;//a+b\*c+(d\*e)

  cout<<"Infix expression :- "<<infix\_1<<endl;

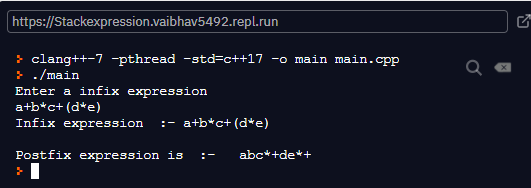
  postfix\_1=infix\_to\_postfix(infix\_1);

  cout<<endl<<"Postfix expression is :- "<<postfix\_1<<endl;

  return 0;

}

**Result/Output: -**



**Learning outcomes (What I have learnt): -**

**1. Algorithm to convert infix to postfix expression.**

**2. How to use stack library.**

**3. How to implement the algorithm to convert infix to postfix.**