Government College of Engineering, Karad Programming for Problem Solving Lab

Nanekar Saurabh Rajesh

20141212

I1

**Experiment No. 5**

**Title**: Implement stack as an ADT to perform expression conversion and evaluation for prefix to infix.

**Outcome:** Students can perform expression conversion and evaluation for prefix to infix.

and its related applications.

**Theory:**

**Prefix to Infix Conversion**

Objective: Given a Prefix expression, write an algorithm to convert it into Infix expression.

**Infix** : An expression is called the Infix expression if the operator appears in between the operands in the expression. Simplify of the form (operand1 operator operand2).

Example : (A+B) \* (C-D)

**Prefix** : An expression is called the prefix expression if the operator appears in the expression before the operands. Simplify of the form (operator operand1 operand2).

Example : \*+AB-CD (Infix : (A+B) \* (C-D) )

Given a Prefix expression, convert it into a Infix expression.

Computers usually does the computation in either prefix or postfix (usually postfix). But for humans, its easier to understand an Infix expression rather than a prefix. Hence conversion is need for human understanding.

Examples:

Input: Prefix expression: + A B

Output: Infix expression- (A + B)

Input : Prefix : \*+AB-CD

Output : Infix : ((A+B)\*(C-D))

Input : Prefix : \*-A/BC-/AKL

Output : Infix : ((A-(B/C))\*((A/K)-L))

Approach: Use Stacks

Algorithm for Prefix to Infix:

* Read the Prefix expression in reverse order (from right to left)
* If the symbol is an operand, then push it onto the Stack
* If the symbol is an operator, then pop two operands from the Stack

Create a string by concatenating the two operands and the operator between them.

string = (operand1 + operator + operand2) And push the resultant string back to Stack

Repeat the above steps until end of Prefix expression.

**Algorithm**:

Iterate the given expression from right to left (in reverse order), one character at a time

* If character is operand, push it to stack.
* If character is operator,
* pop operand from stack, say it’s s1.
* pop operand from stack, say it’s s2.
* perform (s1 operator s2) and push it to stack. i.e. string = (operand1 + operator + operand2)
* Once the expression iteration is completed, initialize result string and pop out from stack and add it to result.
* Return the result.

**Evaluation of Infix Expression**

Infix notation is commonly used in arithmetic formula or statements, the operators are written in-between their operands.

Let’s assume the below:

* Operands are real numbers.
* Permitted operators: +,-, \*, /, ^(exponentiation)
* Blanks are permitted in expression.
* Parenthesis are permitted

**Example:**

A \* ( B + C ) / D

2 \* (5 + 3) / 4

Output: 4

Approach: Use Stacks

We will use two stacks

* Operand stack: This stack will be used to keep track of numbers.
* Operator stack: This stack will be used to keep operations (+, -, \*, /, ^)

Order of precedence of operations–

* ^ (Exponential)
* / \*
* + –

Note: brackets ( ) are used to override these rules.

Let’s define the Process: (will be used for the main algorithm)

* Pop-out two values from the operand stack, let’s say it is A and B.
* Pop-out operation from operator stack. let’s say it is ‘+’.
* Do A + B and push the result to the operand stack.

**Algorithm:**

Iterate through given expression, one character at a time

* If the character is an operand, push it to the operand stack.
* If the character is an operator,
* If the operator stack is empty then push it to the operator stack.
* Else If the operator stack is not empty,
* If the character’s precedence is greater than or equal to the precedence of the stack top of the operator stack, then push the character to the operator stack.
* If the character’s precedence is less than the precedence of the stack top of the operator stack then do Process (as explained above) until character’s precedence is less or stack is not empty.
* If the character is “(“, then push it onto the operator stack.
* If the character is “)”, then do Process (as explained above) until the corresponding “(” is encountered in operator stack. Now just pop out the “(“.Once the expression iteration is completed and the operator stack is not empty, do Process until the operator stack is empty. The values left in the operand stack are our final result.

**Analysis:**



**List of similar programs: Solve any one.**

1. Write a program to find [Longest Common Prefix using Character by Character Matching](https://www.geeksforgeeks.org/longest-common-prefix-using-character-by-character-matching/?ref=rp).
2. Write a program to perform Decimal to octal conversion with minimum use of arithmetic operators.
3. Write a program to [check if Arithmetic Expression contains duplicate parenthesis](https://algorithms.tutorialhorizon.com/check-if-arithmetic-expression-contains-duplicate-parenthesis/).

**Title Program:**  Implement stack as an ADT to perform expression conversion and evaluation for prefix to infix.

**Source code of Implemented Programs:**

//Nanekar Saurabh Rajesh

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#include <conio.h>

char opnds[50][80],oprs[50];

int  topr=-1,topd=-1;

pushd(char \*opnd)

{

    strcpy(opnds[++topd],opnd);

}

char \*popd()

{

    return(opnds[topd--]);

}

pushr(char opr)

{

    oprs[++topr]=opr;

}

char popr()

{

    return(oprs[topr--]);

}

int empty(int t)

{

    if( t == 0) return(1);

    return(0);

}

int main()

{

    char prfx[50],ch,str[50],opnd1[50],opnd2[50],opr[2];

    int i=0,k=0,opndcnt=0;

    printf("Give an Expression = ");

    gets(prfx);

    printf(" Given Prefix Expression : %s\n",prfx);

    while( (ch=prfx[i++]) != '\0')

    {

        if(isalnum(ch))

        {

            str[0]=ch; str[1]='\0';

            pushd(str); opndcnt++;

            if(opndcnt >= 2)

            {

                strcpy(opnd2,popd());

                strcpy(opnd1,popd());

                strcpy(str,"(");

                strcat(str,opnd1);

                ch=popr();

                opr[0]=ch;opr[1]='\0';

                strcat(str,opr);

                strcat(str,opnd2);

                strcat(str,")");

                pushd(str);

                opndcnt-=1;

            }

        }

        else

        {

            pushr(ch);

            if(opndcnt==1)opndcnt=0;

        }

    }

    if(!empty(topd))

    {

        strcpy(opnd2,popd());

        strcpy(opnd1,popd());

        strcpy(str,"(");

        strcat(str,opnd1);

        ch=popr();

        opr[0]=ch;opr[1]='\0';

        strcat(str,opr);

        strcat(str,opnd2);

        strcat(str,")");

        pushd(str);

    }

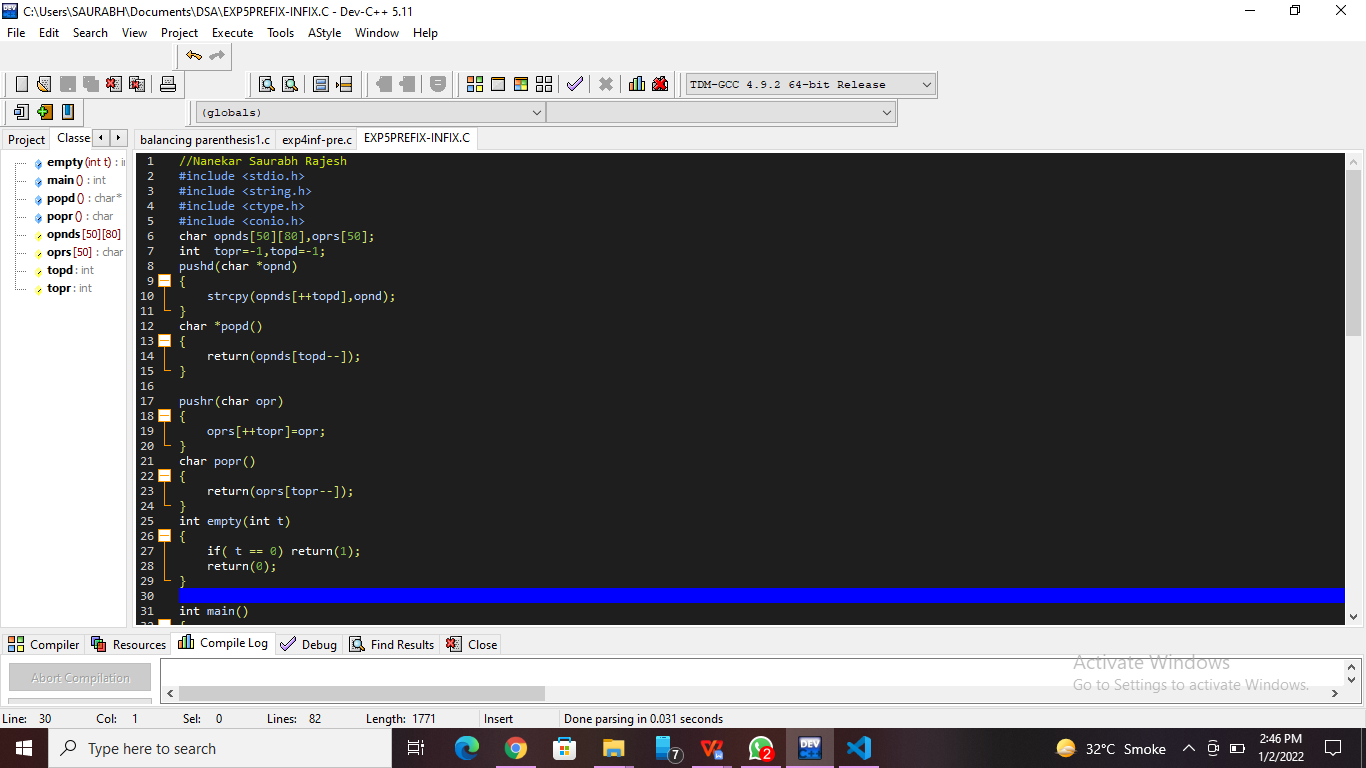
    printf(" Infix Expression: ");

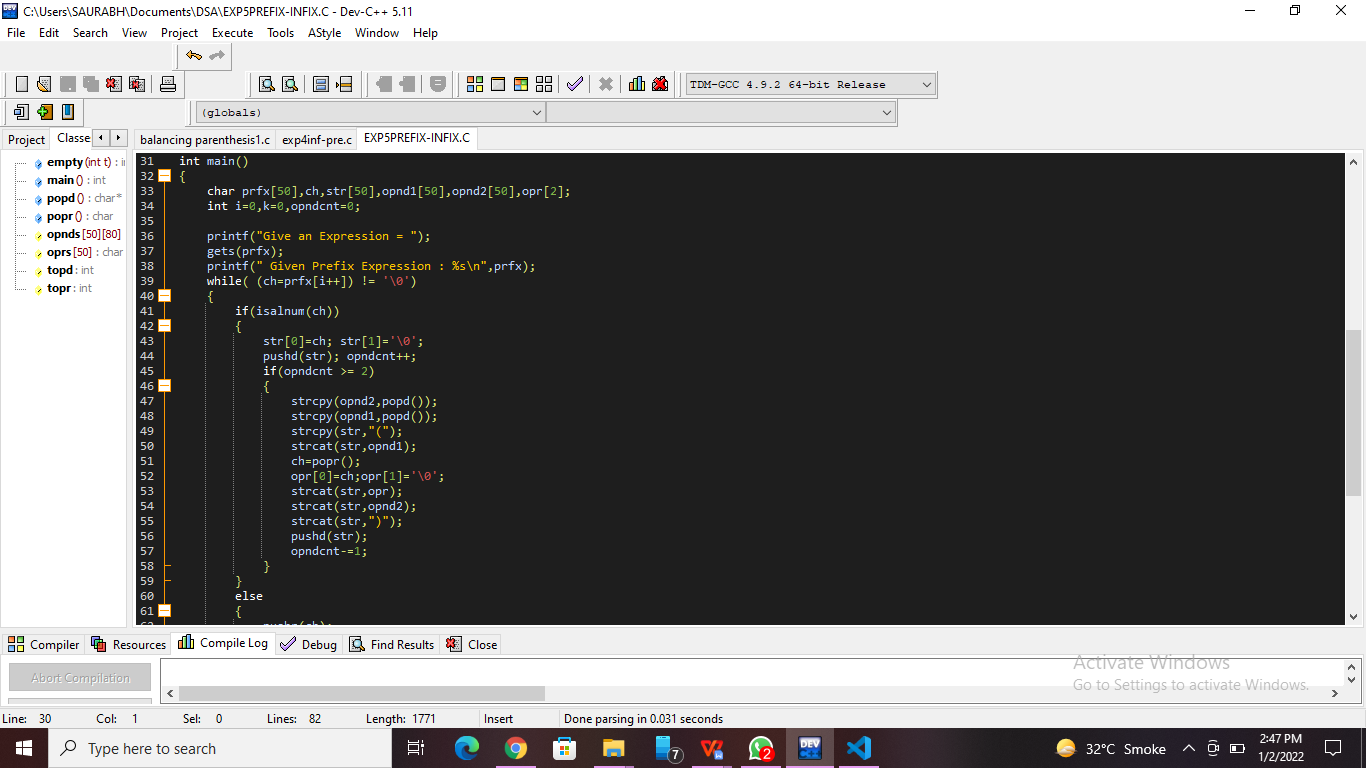
    puts(opnds[topd]);

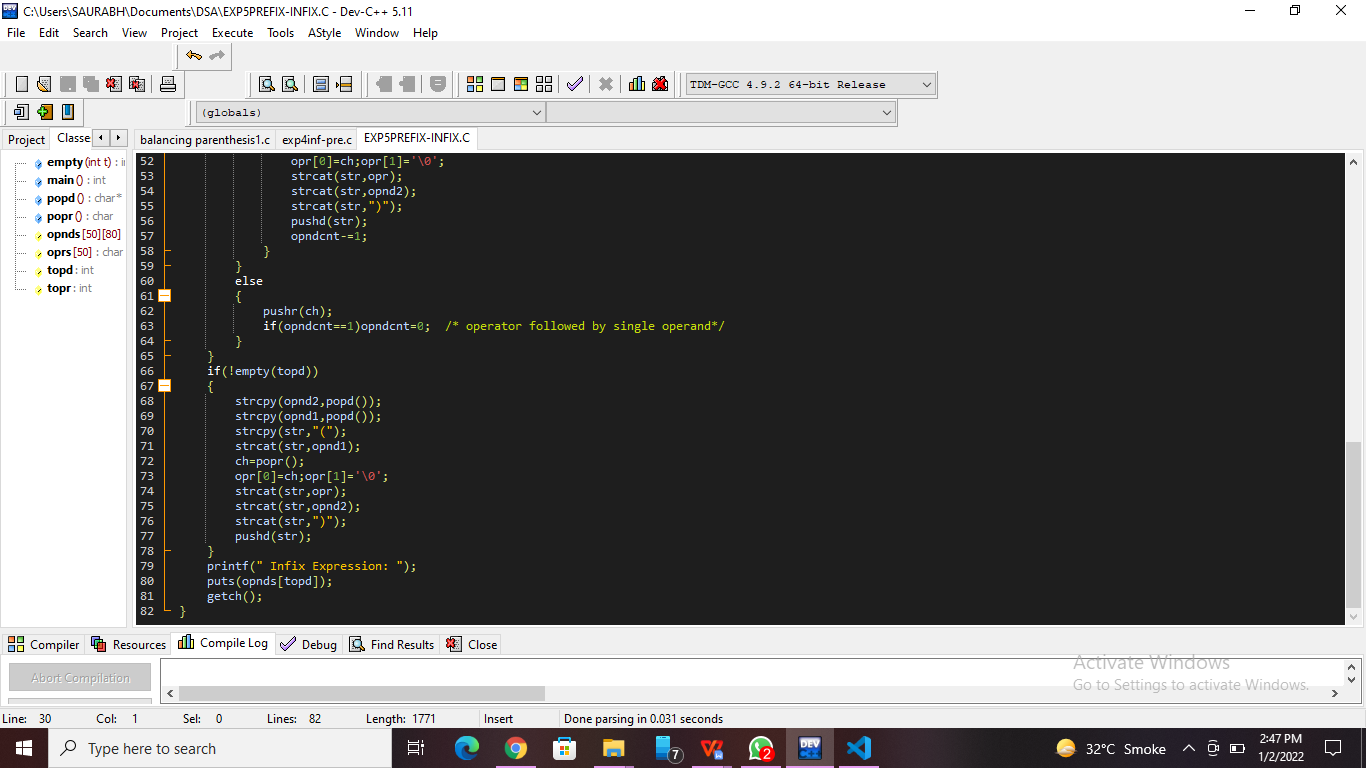
    getch();

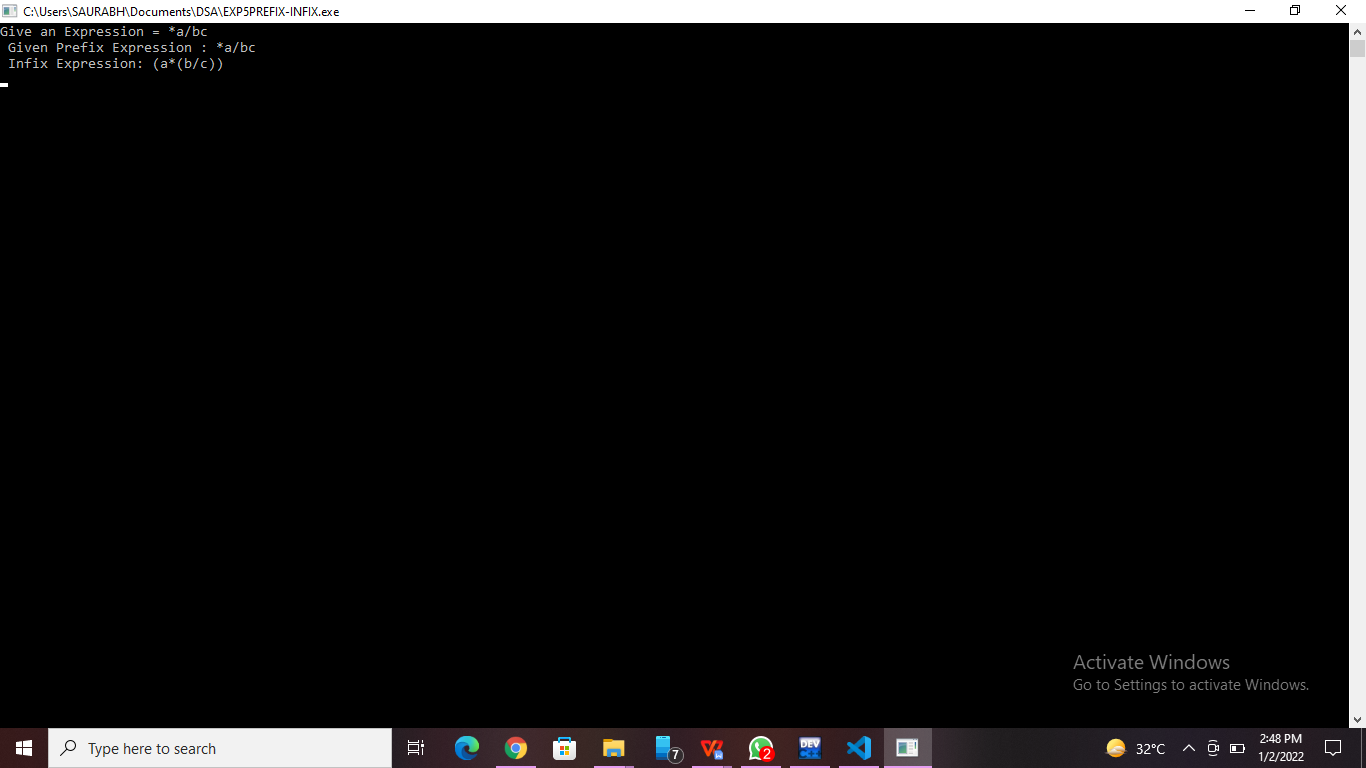
}

**Screenshots of Output:**

****

****

****

****

**Practice Program:**

//Nanekar Saurabh Rajesh

#include <stdio.h>

int main()

{

    long decimalno, remainder, quotient;

    int octalno[100], i = 1, j;

    printf("Enter the decimal number: ");

    scanf("%ld", &decimalno);

    quotient = decimalno;

    while (quotient != 0)

    {

        octalno[i++] = quotient % 8;

        quotient = quotient / 8;

    }

    printf("Equivalent octal value of decimal no %d: ", decimalno);

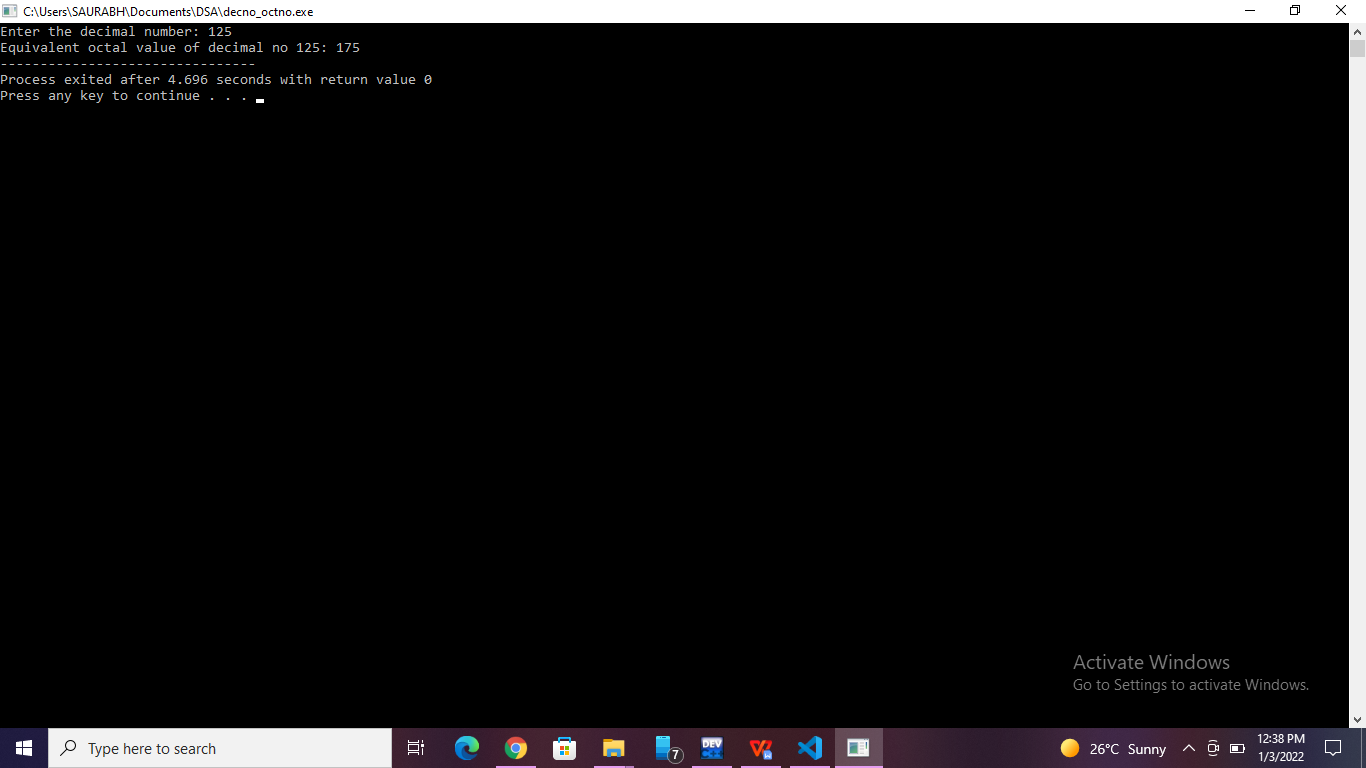
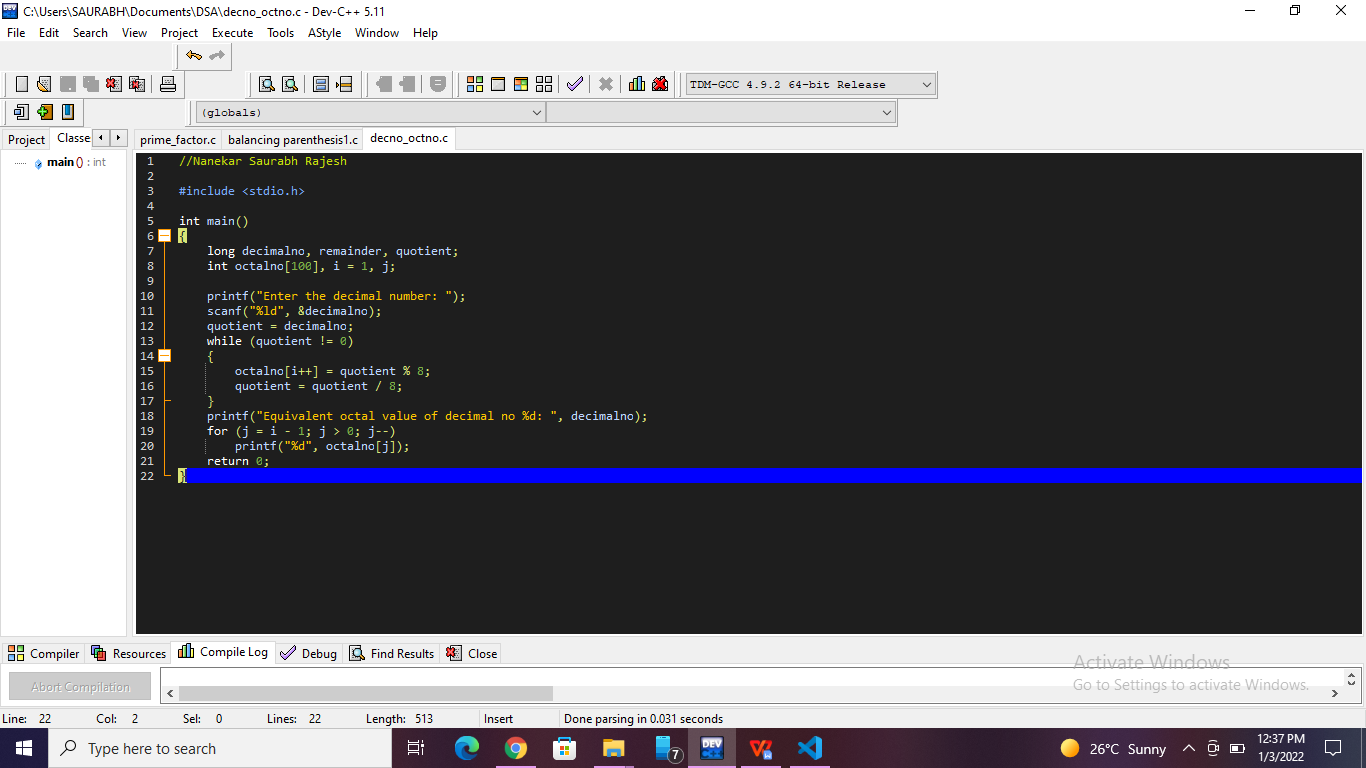
    for (j = i - 1; j > 0; j--)

        printf("%d", octalno[j]);

    return 0;

}

**Practice Program Screenshots:-**

****

**List of sample questions for oral examination:**

1. How do I convert infix to prefix manually?
2. How do you convert prefixes?
3. How can you convert an infix expression to postfix expression using stack give one example?
4. What is infix expression of the given prefix expression :+ A \* BC?
5. What is the postfix expression for the infix expression a B \* C +( D \* E?

**Conclusion:**

I can perform expression conversion and evaluation for prefix to infix. and its related applications.