Government College of Engineering, Karad Programming for Problem Solving Lab

Nanekar Saurabh Rajesh

20141212

I1

**Experiment No. 6**

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

**Outcome:** Students can perform expression conversion and evaluation for prefix to postfix and its related applications.

**Theory:**

**Prefix to Postfix Conversion:**

**What is Prefix conversion?**

An infix expression is an expression in which the operators are written between the two operands. If we move the operator before the operands then it is known as a prefix expression. In other words, prefix expression can be defined as an expression in which all the operators precede the two operands.

**For Example:**

If the infix expression is given as: A + B \* C

As we know that the multiplication operator \* has a higher precedence than the addition operator. First, multiplication operator will move before operand B shown as below:

A + \* B C

Once the multiplication operator is moved before 'B' operand, addition operator will move before the operand 'A' shown as below:

+ A \* B C

**Evaluation of Prefix Expression using Stack:**

**Step 1:** Initialize a pointer 'S' pointing to the end of the expression.

**Step 2:** If the symbol pointed by 'S' is an operand then push it into the stack.

**Step 3:** If the symbol pointed by 'S' is an operator then pop two operands from the stack. Perform the operation on these two operands and stores the result into the stack.

**Step 4:** Decrement the pointer 'S' by 1 and move to step 2 as long as the symbols left in the expression.

**Step 5:** The final result is stored at the top of the stack and return it.

**Step 6:** End

**Let's understand the evaluation of prefix expression through an example.**

**Expression: +, -, \*, 2, 2, /, 16, 8, 5**

First, we will reverse the expression given above.

Expression: 5, 8, 16, /, 2, 2, \*, -, +

We will use the stack data structure to evaluate the prefix expression.

|  |  |
| --- | --- |
| **Symbol Scanned** | **Stack** |
| 5 | 5 |
| 8 | 5, 8 |
| 16 | 5, 8, 16 |
| / | 5, 2 |
| 2 | 5, 2, 2 |
| 2 | 5, 2, 2, 2 |
| \* | 5, 2, 4 |
| - | 5, 2 |
| + | 7 |

**The final result of the above expression is 7.**

### What is Postfix expression?

If we move the operators after the operands then it is known as a postfix expression. In other words, postfix expression can be defined as an expression in which all the operators are present after the operands.

**For example:**

If the infix expression is A + B \* C

As we know that the multiplication operator has a higher precedence than the addition operator, so multiplication operator will move after the operands B and C shown as below:

A + B C \*

Once the multiplication operator is moved after the operand C, then the addition operator will come after the multiplication operator shown as below:

A B C \* +

**Evaluation of Postfix expression using Stack**

Algorithm for the evaluation of postfix expression using stack:

**Step 1**: Create an empty stack used for storing the operands.

**Step 2:** Scan each element of an expression one be one and do the following:

* If the element is an operand then push it into the stack.
* If the element is an operator then pop two operands from the stack. Perform operation on these operands. Push the final result into the stack.

**Step 3:** When the expression is scanned completely, the value available in the stack would be the final output of the given expression.

### Conversion of Prefix to Postfix Expression

Here, we will see the conversion of prefix to postfix expression using a stack data structure.

**Rules for prefix to postfix expression using stack data structure:**

* Scan the prefix expression from right to left, i.e., reverse.
* If the incoming symbol is an operand then push it into the stack.
* If the incoming symbol is an operator then pop two operands from the stack. Once the operands are popped out from the stack, we add the incoming symbol after the operands. When the operator is added after the operands, then the expression is pushed back into the stack.
* Once the whole expression is scanned, pop and prints the postfix expression from the stack.

**Pseudocode for prefix to postfix conversion:**

1. Function PrefixToPostfix(string prefix)
2. 1.   Stack s
3. 2.   Loop: i = prefix.length-1 to 0
4. •   **if** prefix[i] is operand ->
5. s.push(prefix[i])
6. •   **else** **if** prefix[i] is operator->
7. op1 = s.top()
8. s.pop()
9. op2 = s.top()
10. s.pop()
11. exp = op1 + op2 + prefix[i]
12. s.push(exp)
13. End Loop
14. 3.   Return s.top

**Let's understand the conversion of Prefix to Postfix expression using Stack through an example.**

**If the expression is: \* - A / B C - / A K L**

|  |  |  |  |
| --- | --- | --- | --- |
| **Symbols to be scanned** | **Action** | **Stack** | **Description** |
| L | Push L into the stack | L |  |
| K | Push K into the stack | L, K |  |
| A | Push A into the stack | L, K, A |  |
| / | Pop A from the stack Pop K from the stack Push A K / into the stack | L, A K / | Pop two operands from the stack, i.e., A and K. Add '/' operator after K operand, i.e., AK/. Push AK/ into the stack. |
| - | Pop A K / and L from the stack. Push (A K / L -) into the stack | A K / L - | Pop two operands from the stack, i.e., AK/ and L. Add '-' operator after 'L' operand. |
| C | Push C into the stack | AK/L-, C |  |
| B | Push B into the stack | AK/L-, C, B |  |
| / | Pop B and C from the stack. Push BC/ into the stack. | AK/L-, BC/ | Pop two operands from the stack, i.e., B and C. Add '/' operator after C operator, i.e., BC/. Push BC/ into the stack. |
| A | Push A into the stack | AK/L-, BC/, A |  |
| - | Pop BC/ and A from the stack. Push ABC/- into the stack. | AK/L-, ABC/- | Pop two operands from the stack, i.e., A and BC/. Add '-' operator after '/'. |
| \* | Pop ABC/- and AK/L- from the stack. Push ABC/AK/L-\* into the stack. | ABC/-AK/L-\* | Pop two operands from the stack, i.e., ABC/-, and AK/L- . Add '\*' operator after L and '-' operator, i.e., ABC/-AK/L-\*. |

**Analysis:**



**List of similar programs:**

1. Write a program to find prime factors of a number using Stack.

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

**Source code of Implemented Programs:**

//Nanekar Saurabh Rajesh

#include<stdio.h>

#include<string.h>

#include<math.h>

#include<stdlib.h>

#define BLANK ' '

#define TAB '\t'

#define MAX 50

char \*pop();

char prefix[MAX];

char stack[MAX][MAX];

void push(char \*str);

int isempty();

int white\_space(char symbol);

void prefix\_to\_postfix();

int top;

int main()

{

        top = -1;

        printf("Enter Prefix Expression : ");

        gets(prefix);

        prefix\_to\_postfix();

}

void prefix\_to\_postfix()

{

        int i;

        char operand1[MAX], operand2[MAX];

        char symbol;

        char temp[2];

        char strin[MAX];

        for(i=strlen(prefix)-1;i>=0;i--)

        {

                symbol=prefix[i];

                temp[0]=symbol;

                temp[1]='\0';

                if(!white\_space(symbol))

                {

                        switch(symbol)

                        {

                        case '+':

                        case '-':

                        case '\*':

                        case '/':

                        case '%':

                        case '^':

                                strcpy(operand1,pop());

                                strcpy(operand2,pop());

                                strcpy(strin,operand1);

                                strcat(strin,operand2);

                                strcat(strin,temp);

                                push(strin);

                                break;

                        default:

                             push(temp);

                        }

                }

        }

        printf("\nPostfix Expression :: ");

        puts(stack[0]);

}

void push(char \*str)

{

        if(top > MAX)

        {

                printf("\nStack overflow\n");

                exit(1);

        }

        else

        {

                top=top+1;

                strcpy( stack[top], str);

        }

}

char \*pop()

{

        if(top == -1 )

        {

                printf("\nStack underflow \n");

                exit(2);

        }

        else

                return (stack[top--]);

}

int isempty()

{

        if(top==-1)

                return 1;

        else

                return 0;

}

int white\_space(char symbol)

{

        if(symbol==BLANK || symbol==TAB || symbol=='\0')

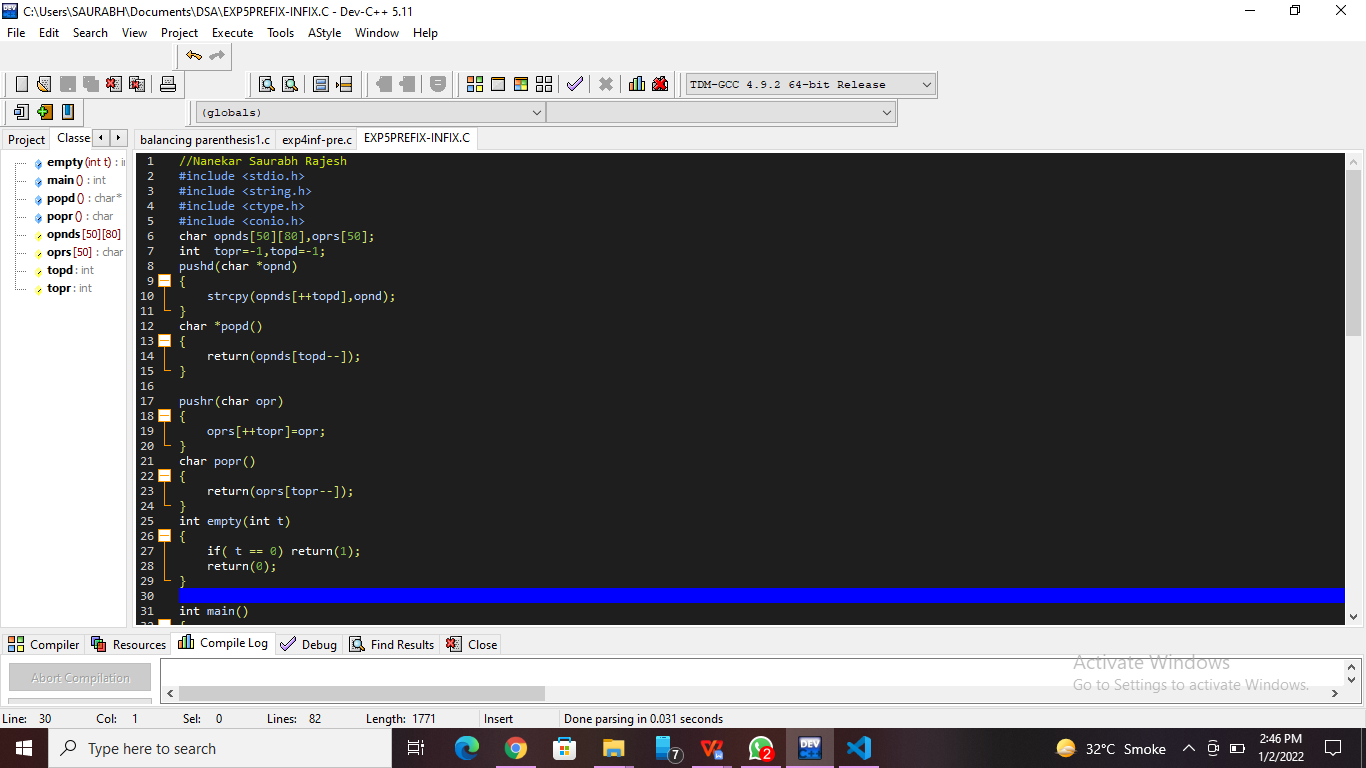
                return 1;

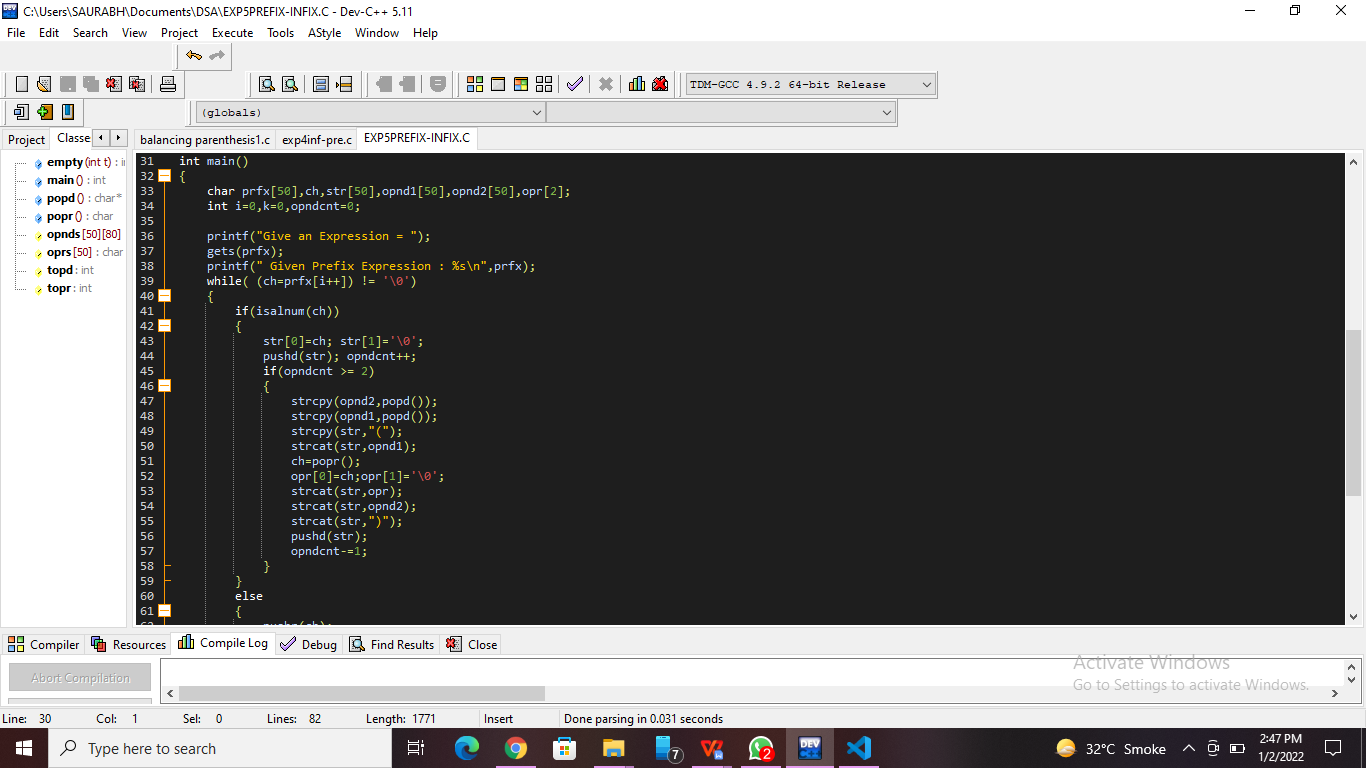
        else

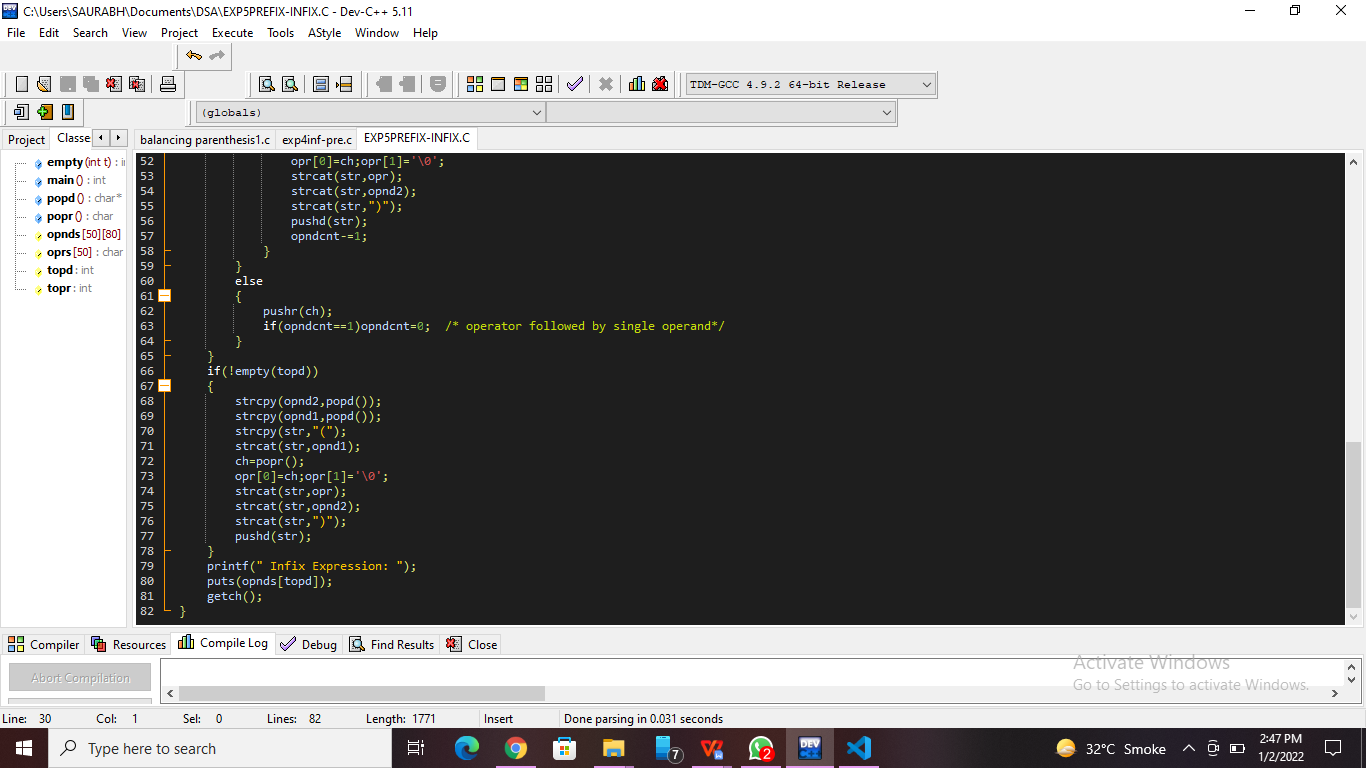
                return 0;

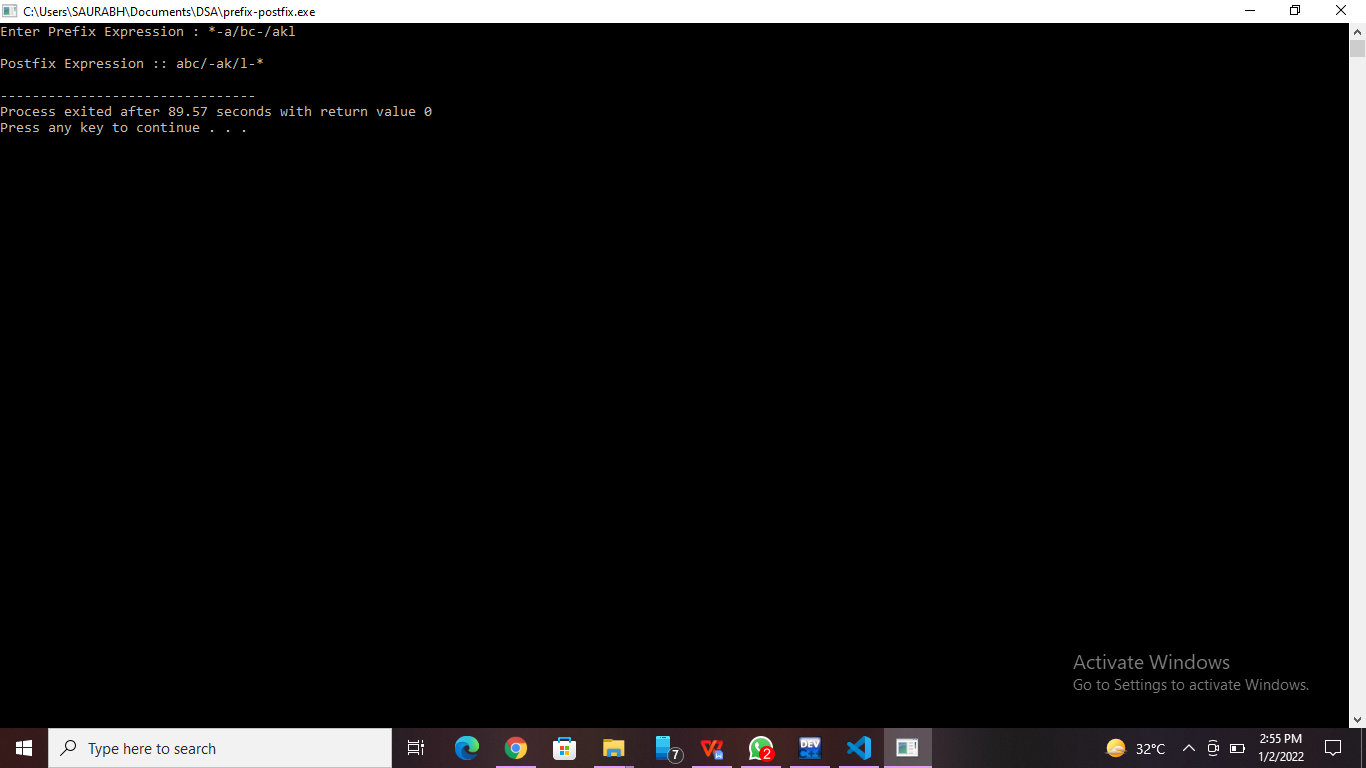
}

**Screenshots of Output:**

****

****

****

****

**Practice Program:**

Write a program to find prime factors of a number using Stack.

//Nanekar Saurabh Rajesh

#include<stdio.h>

#include<stdlib.h>

#define MAX 50

int isEmpty(int top, int stack\_arr[]);

void PFactors( int num);

void push(int x, int \*top, int stack\_arr[]);

int pop(int \*top, int stack\_arr[]);

int main()

{

        int num;

        printf("Enter an integer : ");

        scanf("%d",&num);

        printf("Prime factors are : ");

        PFactors(num);

        return 0;

}

void PFactors( int num)

{

        int stack[MAX], top=-1, i;

        for(i=2; num!=1; i++)

                while(num%i == 0)

                {

                        push(i, &top, stack);

                        num = num/i;

                }

        while(top!=-1)

        {

                printf("%d ",stack[top]);

                pop(&top, stack);

        }

        printf("\n");

    }

void push(int x, int \*top, int stack\_arr[])

{

        if(\*top == (MAX-1))

                printf("Stack Overflow\n");

        else

        {

                \*top=\*top+1;

                stack\_arr[\*top] = x;

        }

}

int pop(int \*top, int stack\_arr[])

{

        int x;

        if(\*top == -1)

        {

                printf("Stack Underflow\n");

                exit(1);

        }

        else

        {

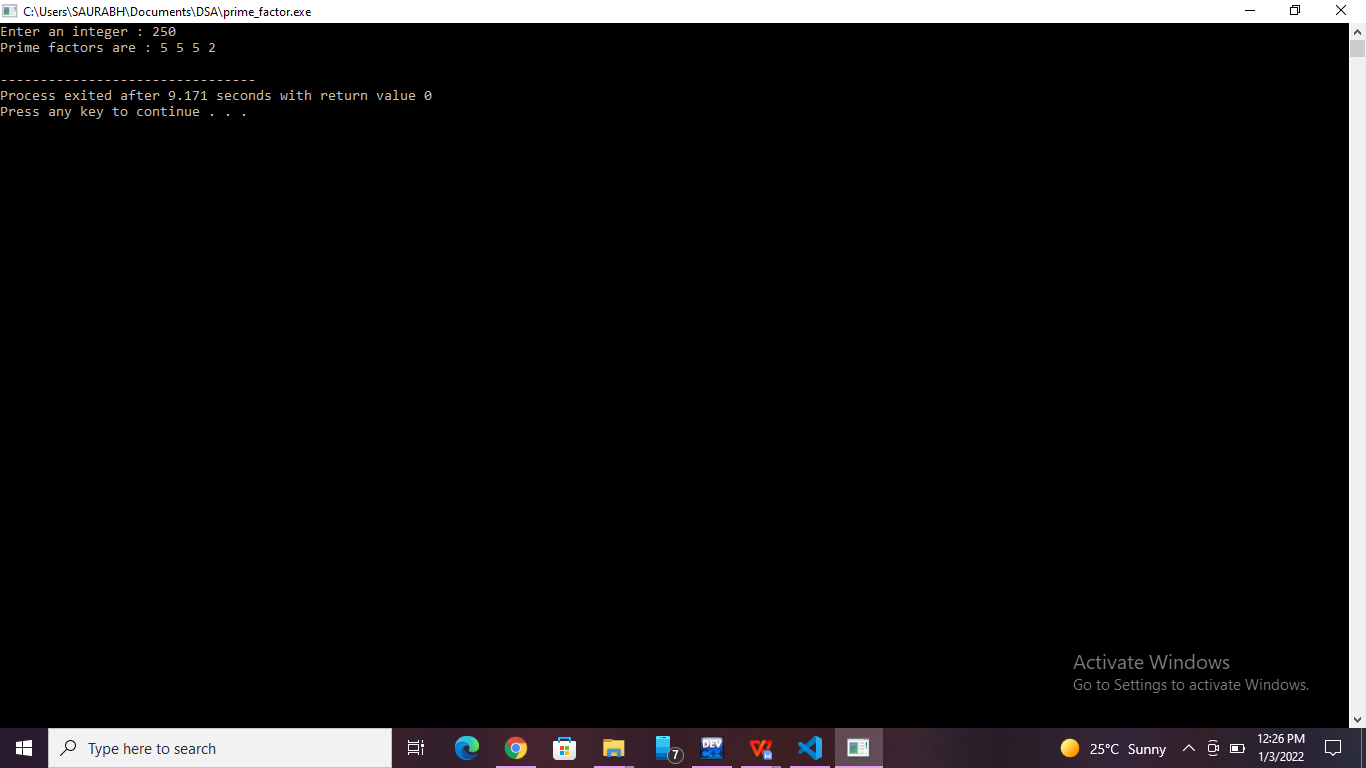
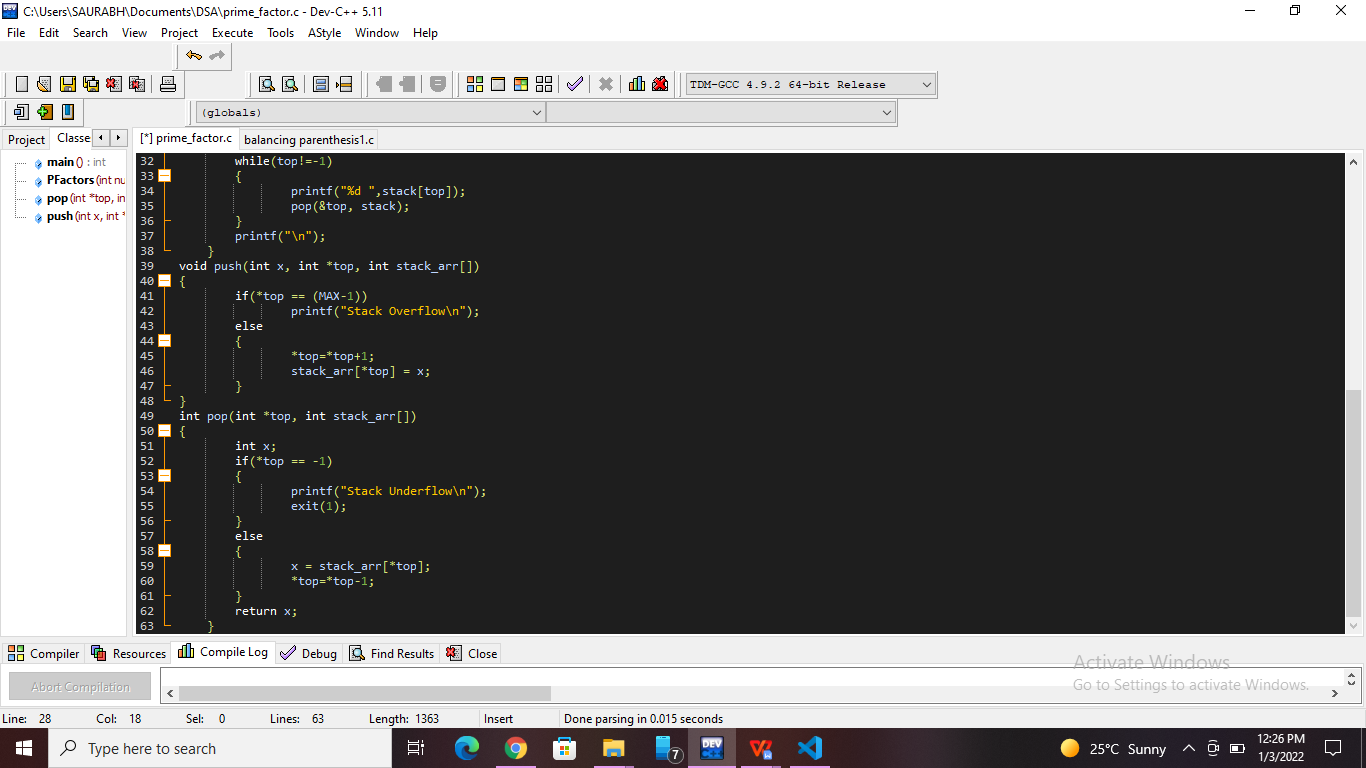
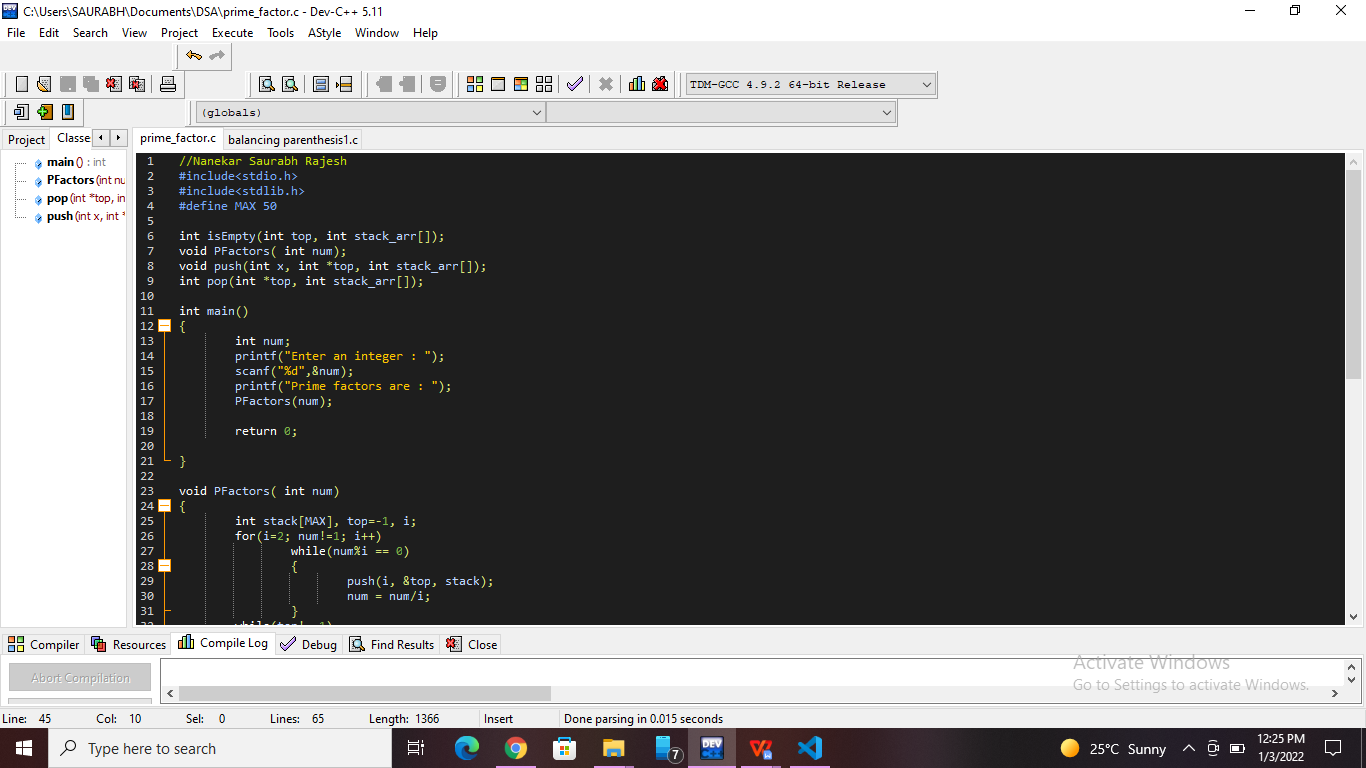
                x = stack\_arr[\*top];

                \*top=\*top-1;

        }

        return x;

}

**Practice program Screenshots:-**

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

1. Why do we convert prefix to postfix?
2. Which is better prefix or postfix?
3. Is prefix reverse of postfix?
4. What is the postfix form of the following prefix expression \*- ABC?
5. What determines the order of evaluation of a prefix expression \*?

**Conclusion:**

We can fastly perform postfix expression as compared to infix expression because we don’t need any parenthesis in postfix expression.