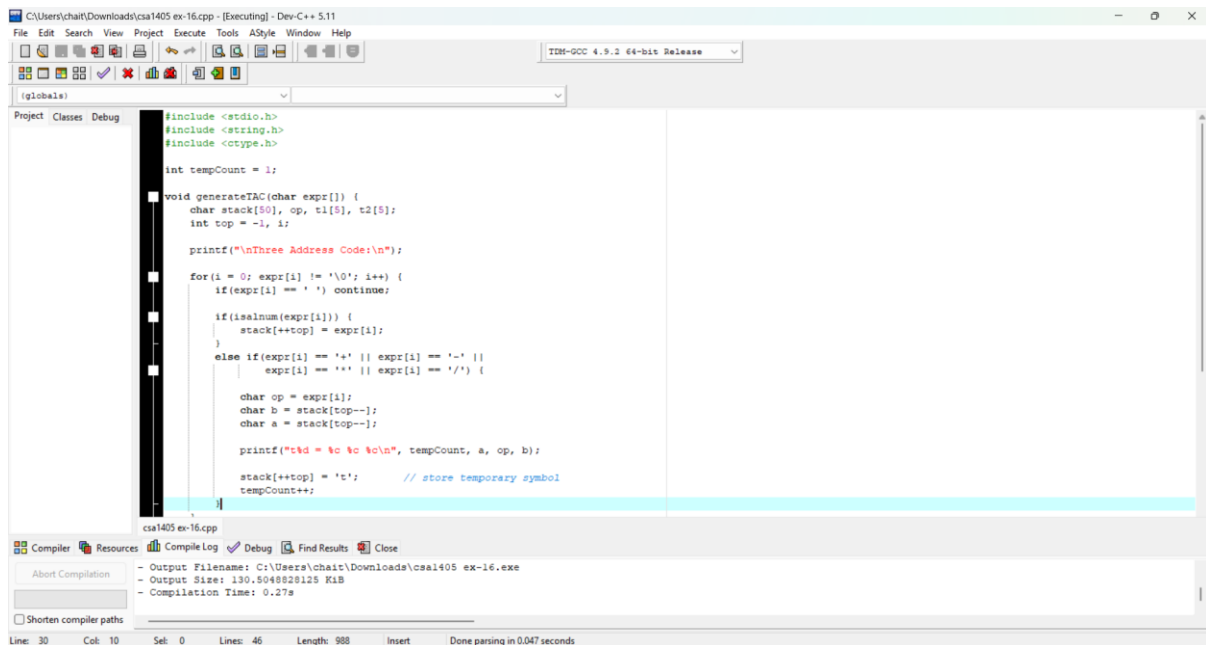


## Exp. No. 16

**Write a C Program to Generate the Three address code representation for the given input statement.**



```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

int tempCount = 1;

void generateTAC(char expr[]) {
    char stack[50], op, t1[5], t2[5];
    int top = -1, i;

    printf("\nThree Address Code:\n");

    for(i = 0; expr[i] != '\0'; i++) {
        if(expr[i] == ' ') continue;

        if(isalnum(expr[i])) {
            stack[++top] = expr[i];
        }
        else if(expr[i] == '+' || expr[i] == '-' ||
                expr[i] == '*' || expr[i] == '/') {

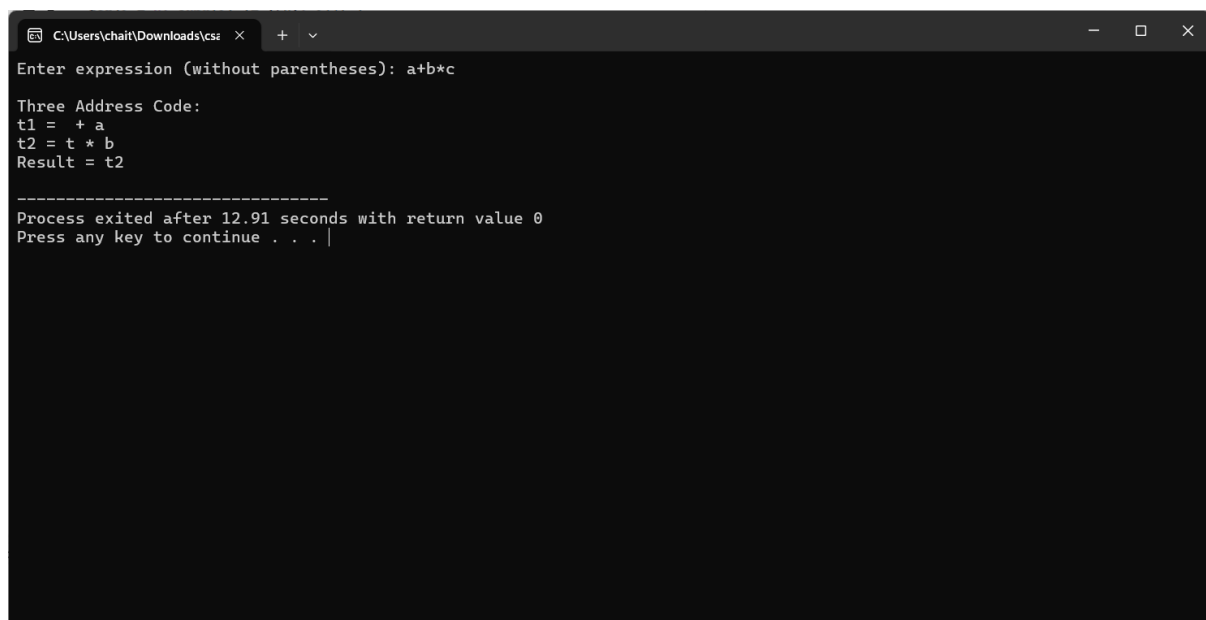
            char op = expr[i];
            char b = stack[top--];
            char a = stack[top--];

            printf("t%d = %c %c %c\n", tempCount, a, op, b);

            stack[++top] = 't'; // store temporary symbol
            tempCount++;
        }
    }
}
```

Compiler Output:

```
- Output Filename: C:\Users\chait\Downloads\csa1405 ex-16.exe
- Output Size: 130.5048828125 KiB
- Compilation Time: 0.27s
```



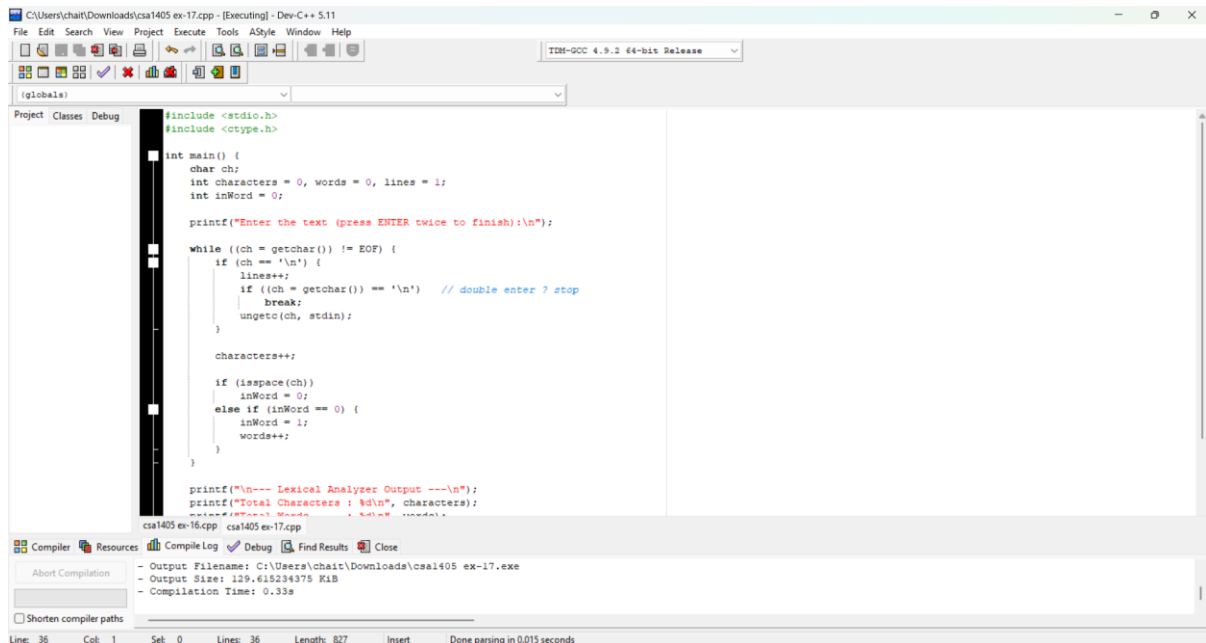
```
C:\Users\chait\Downloads\csa1405 ex-16.exe
Enter expression (without parentheses): a+b*c

Three Address Code:
t1 = + a
t2 = t * b
Result = t2

-----
Process exited after 12.91 seconds with return value 0
Press any key to continue . . .
```

## Exp. No. 17

**Write a C program for implementing a Lexical Analyzer to Scan and Count the number of characters, words, and lines in a file.**



```
#include <stdio.h>
#include <ctype.h>

int main() {
    char ch;
    int characters = 0, words = 0, lines = 1;
    int inWord = 0;

    printf("Enter the text (press ENTER twice to finish):\n");

    while ((ch = getchar()) != EOF) {
        if (ch == '\n') {
            lines++;
            if ((ch = getchar()) == '\n') // double enter ? stop
                break;
            ungetc(ch, stdin);
        }
        characters++;

        if (isspace(ch))
            inWord = 0;
        else if (inWord == 0) {
            inWord = 1;
            words++;
        }
    }

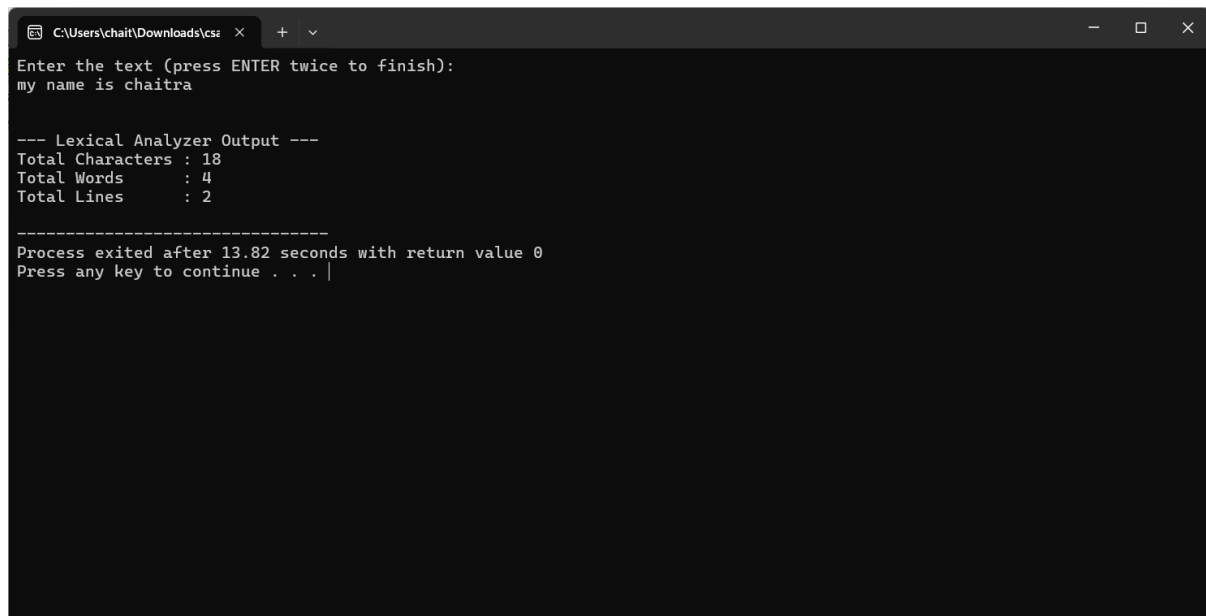
    printf("\n--- Lexical Analyzer Output ---\n");
    printf("Total Characters : %d\n", characters);
    printf("Total Words : %d\n", words);
    printf("Total Lines : %d\n", lines);
}
```

Compiler: csa1405 ex-16.cpp csa1405 ex-17.app

Compiler Output:

- Output Filename: C:\Users\chait\Downloads\csa1405 ex-17.exe
- Output Size: 129.615234375 KiB
- Compilation Time: 0.33s

Line: 36 Col: 1 Set: 0 Lines: 36 Length: 827 Insert Done parsing in 0.015 seconds



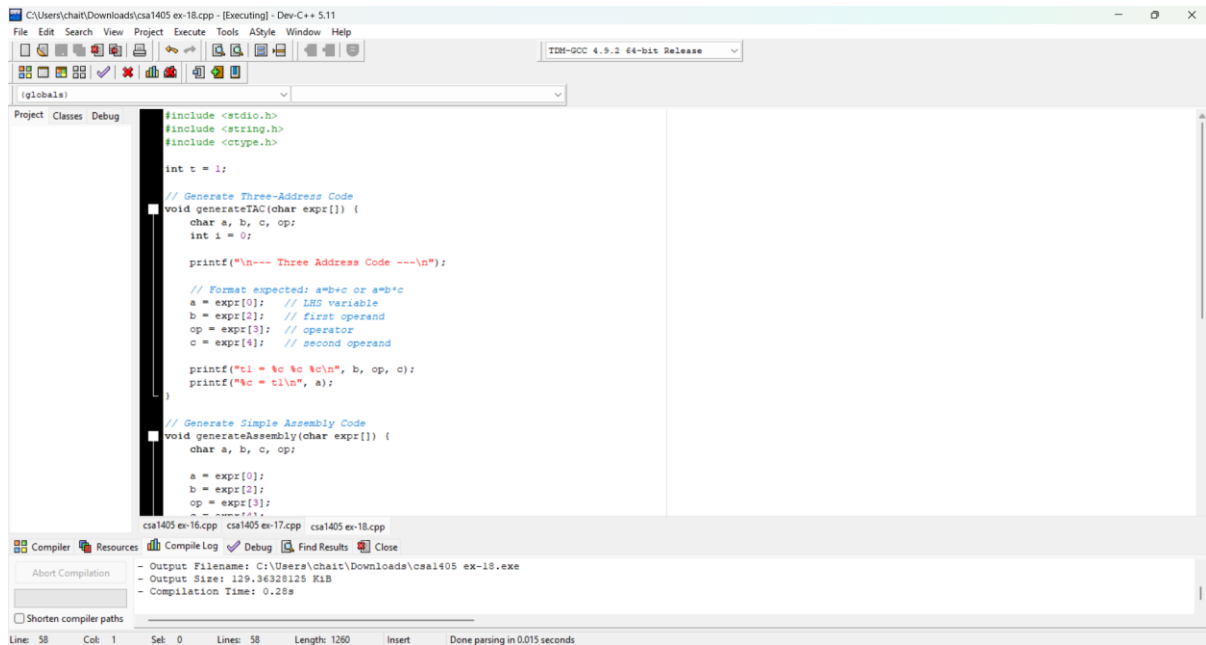
```
C:\Users\chait\Downloads\csa1405 ex-17.exe
Enter the text (press ENTER twice to finish):
my name is chaitra

--- Lexical Analyzer Output ---
Total Characters : 18
Total Words      : 4
Total Lines      : 2

-----
Process exited after 13.82 seconds with return value 0
Press any key to continue . . .
```

## Exp. No. 18

Write a C program to implement the back end of the compiler.



```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

int t = 1;

// Generate Three-Address Code
void generateTAC(char expr[]) {
    char a, b, c, op;
    int i = 0;

    printf("\n--- Three Address Code ---\n");

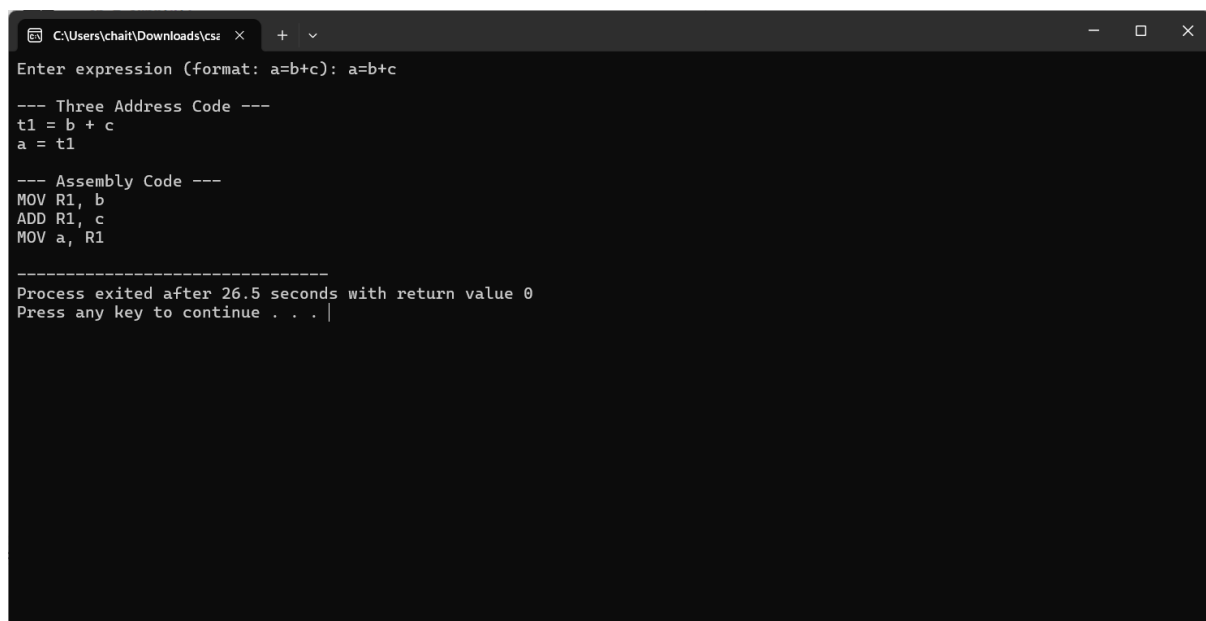
    // Format expected: a=b+c or a=b*c
    a = expr[0]; // LHS variable
    b = expr[2]; // first operand
    op = expr[3]; // operator
    c = expr[4]; // second operand

    printf("t1 = %c %c %c\n", b, op, c);
    printf("%c = t1\n", a);
}

// Generate Simple Assembly Code
void generateAssembly(char expr[]) {
    char a, b, c, op;

    a = expr[0];
    b = expr[2];
    op = expr[3];
    c = expr[4];

    printf("MOV R1, %c\n", b);
    printf("ADD R1, %c\n", c);
    printf("MOV %c, R1\n", a);
}
```



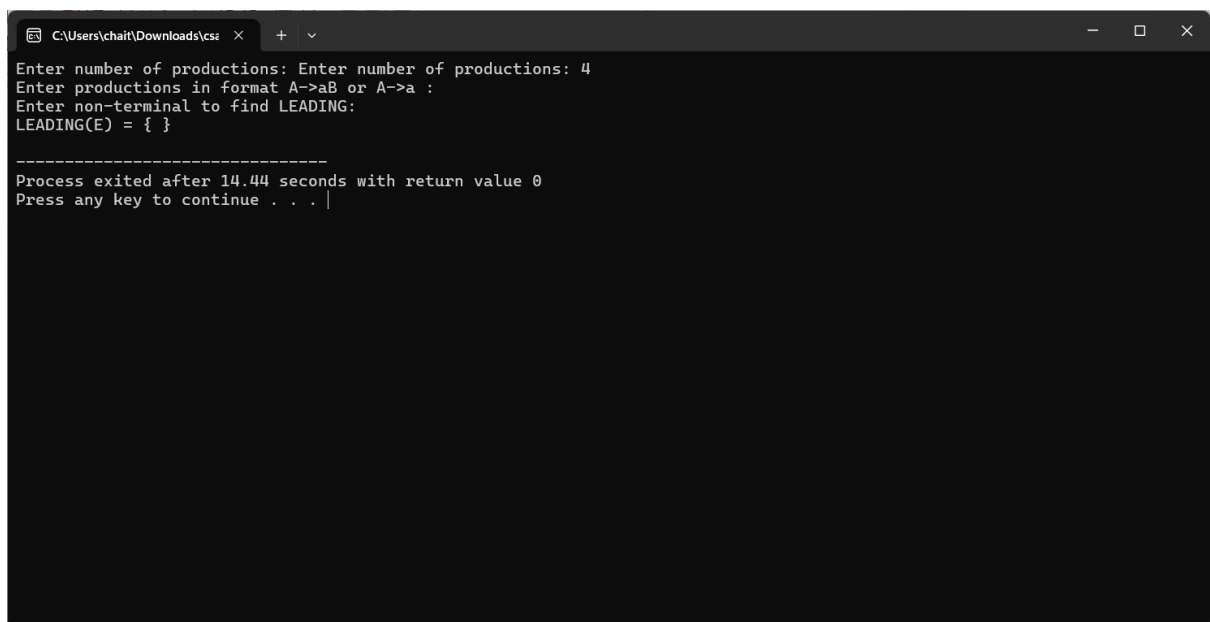
```
C:\Users\chait\Downloads\csa1405 ex-18.exe
Enter expression (format: a=b+c): a=b+c

--- Three Address Code ---
t1 = b + c
a = t1

--- Assembly Code ---
MOV R1, b
ADD R1, c
MOV a, R1

Process exited after 26.5 seconds with return value 0
Press any key to continue . . .
```

**Write a C program to compute LEADING( ) – operator precedence parser for the given grammar**



**Write a C program to compute TRAILING( ) – operator precedence parser for the given grammar**

