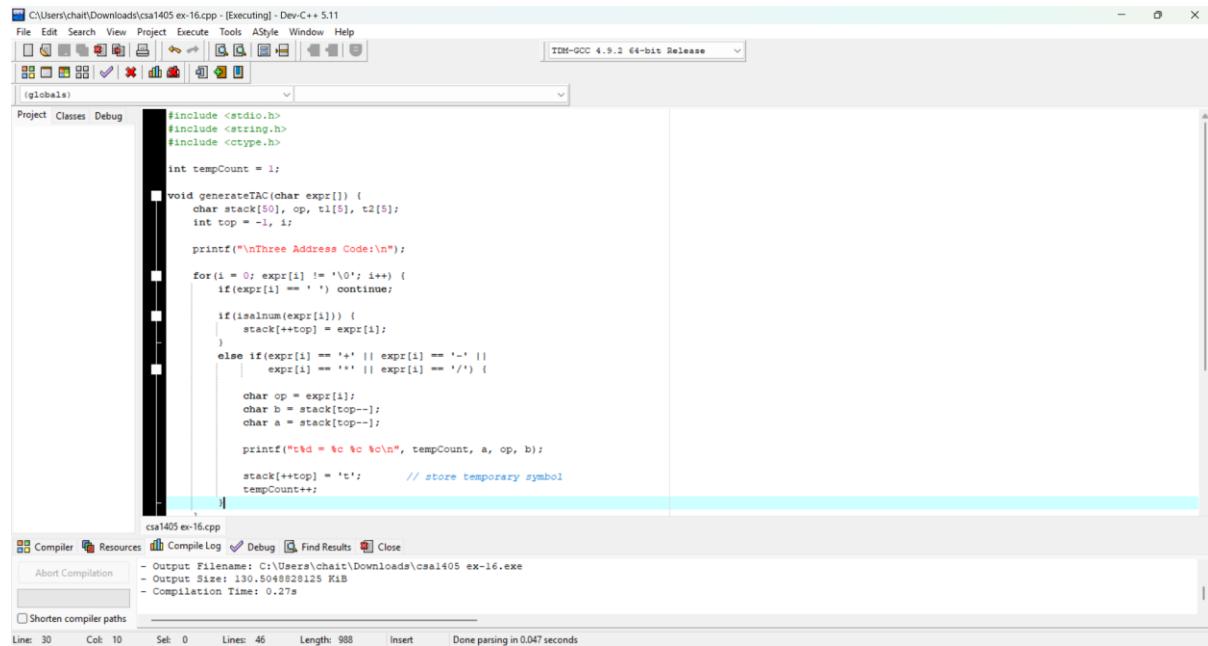


Exp. No. 16

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



The screenshot shows the Dev-C++ IDE interface. The main window displays a C++ source file named "ex-16.cpp". The code implements a function to generate Three Address Code (TAC) from a given expression string. It uses a stack to manage temporary symbols and tracks the current temporary symbol count. The code includes comments explaining the logic for handling different operators and operands. Below the code editor, the status bar shows the file name, output size, and compilation time. The bottom panel contains tabs for Compiler, Resources, Compile Log, Debug, Find Results, and Close, with the Debug tab currently selected.

```
#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;

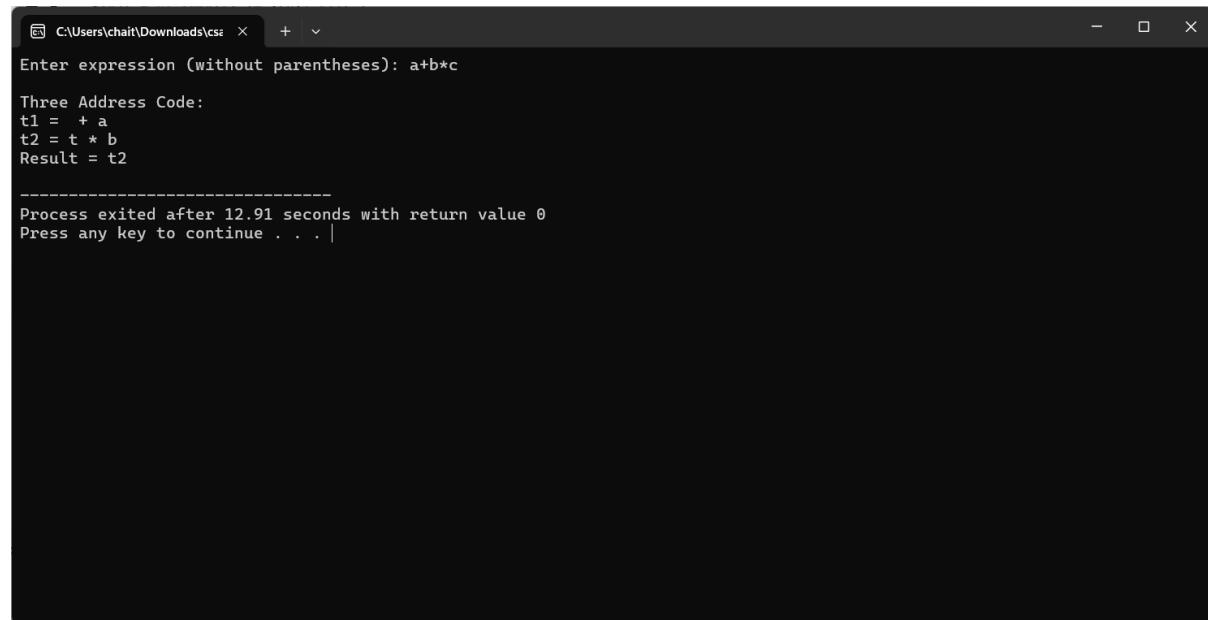
    printf("\nThree Address Code:\n");

    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++;
        }
    }
}
```

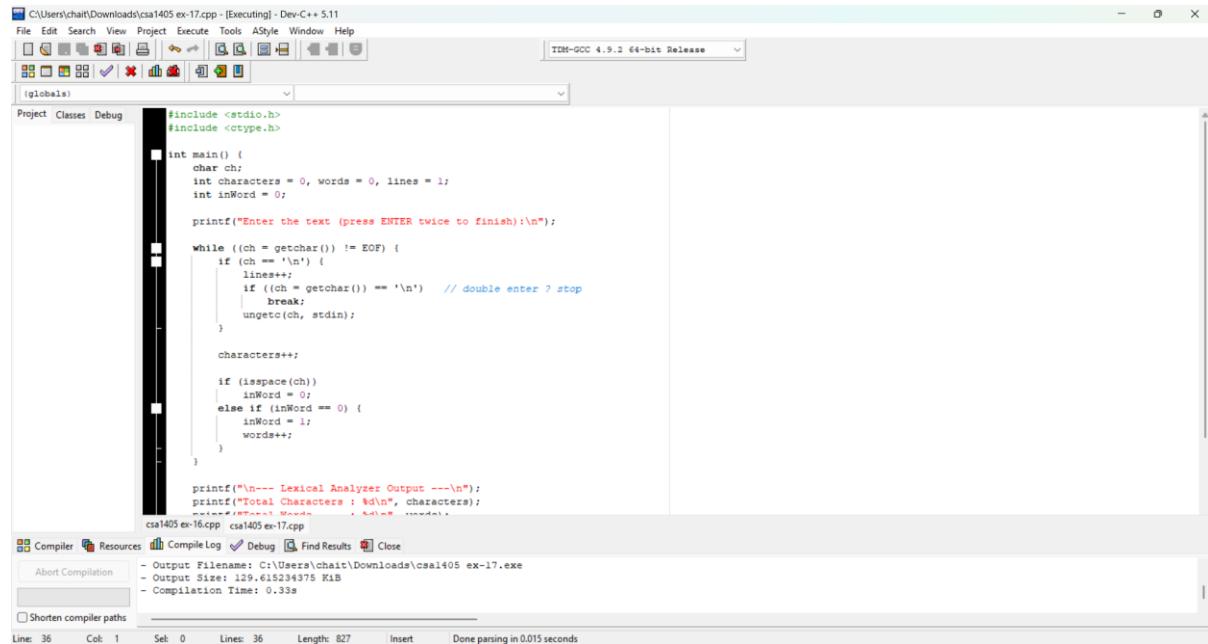


The screenshot shows a terminal window displaying the execution of the generated program. The user enters the expression "a+b*c" as input. The program outputs the Three Address Code generated for the expression. The output shows three temporaries (t1, t2, t3) being assigned values 'a', 'b', and 'c' respectively, followed by the addition and multiplication operations. The process exits after 12.91 seconds. The terminal window has a dark background and light-colored text.

```
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.



The screenshot shows the Dev-C++ IDE interface. The code editor window displays a C program named ex-17.cpp. The code implements a lexical analyzer to count characters, words, and lines from standard input. It uses `getchar()` to read characters, `isalpha()` to identify words, and handles newlines for line counting. The compiler output window at the bottom shows the compilation log, including the output filename, size, and time.

```
#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 (isalpha(ch))
            inWord = 0;
        else if (inWord == 0) {
            inWord = 1;
            words++;
        }
    }

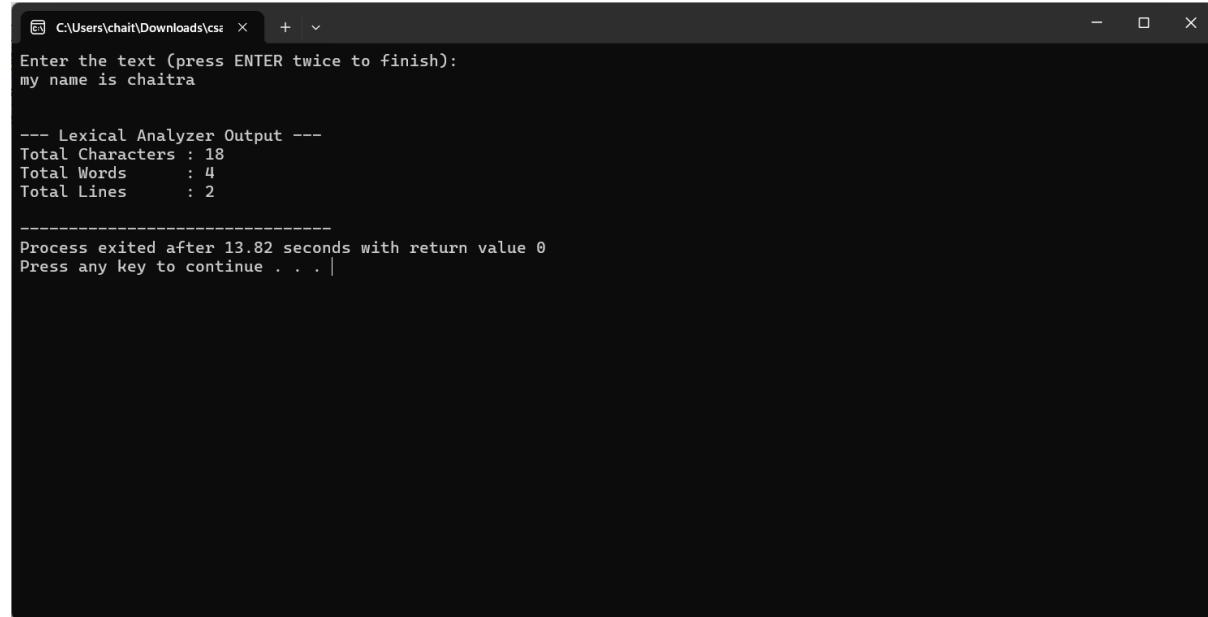
    printf("\n--- Lexical Analyzer Output ---\n");
    printf("Total Characters : %d\n", characters);
    printf("Total Words      : %d\n", words);
    printf("Total Lines       : %d\n", lines);
}

cs1405_ex-16.cpp  cs1405_ex-17.cpp
```

Compiler Resources Compile Log Debug Find Results Close

- Output Filename: C:\Users\chait\Downloads\cs1405_ex-17.exe
- Output Size: 129.615234375 Kib
- Compilation Time: 0.33s

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



The screenshot shows a terminal window titled 'C:\Users\chait\Downloads\css'. It displays the output of the lexical analyzer program. The user enters the text 'my name is chaitra' and the program responds with the analysis results: Total Characters: 18, Total Words: 4, and Total Lines: 2. The terminal also shows the execution time and a prompt to press any key to continue.

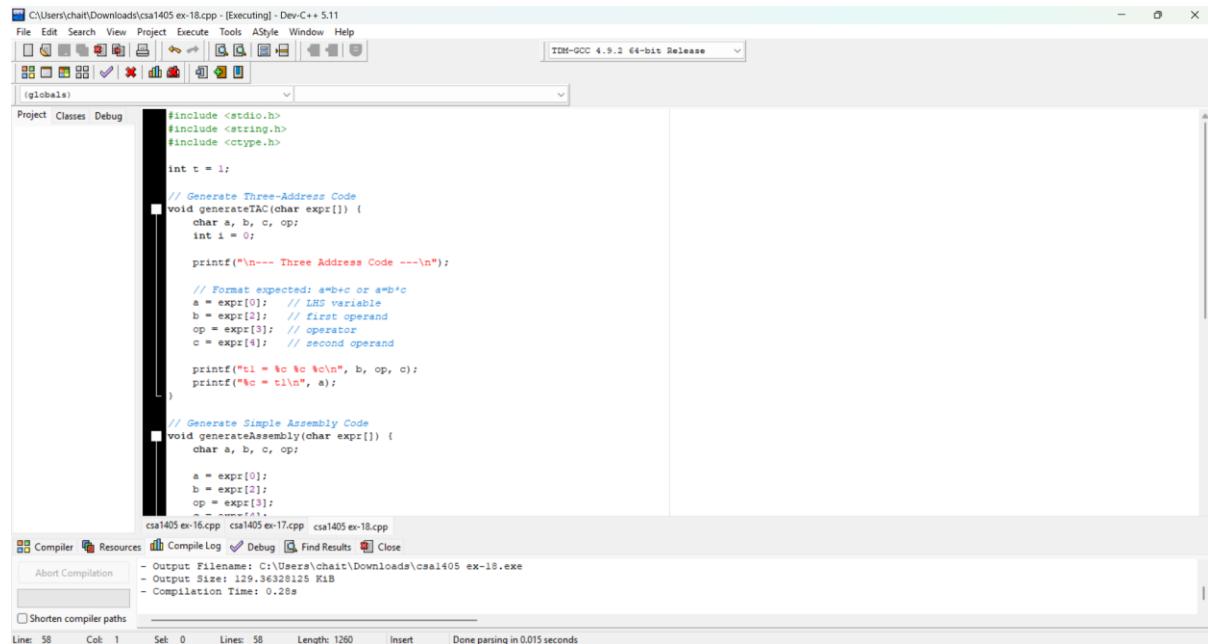
```
Enter the text (press ENTER twice to finish):
my name is chaitra

--- 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.



The screenshot shows the Dev-C++ IDE interface. The code editor displays a C program that generates Three-Address Code and Assembly Code. The code includes comments for generating TAC and assembly code, and it uses printf statements to output the generated code. The IDE's status bar at the bottom shows the file is being executed, with the output filename, size, and compilation time.

```
#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");

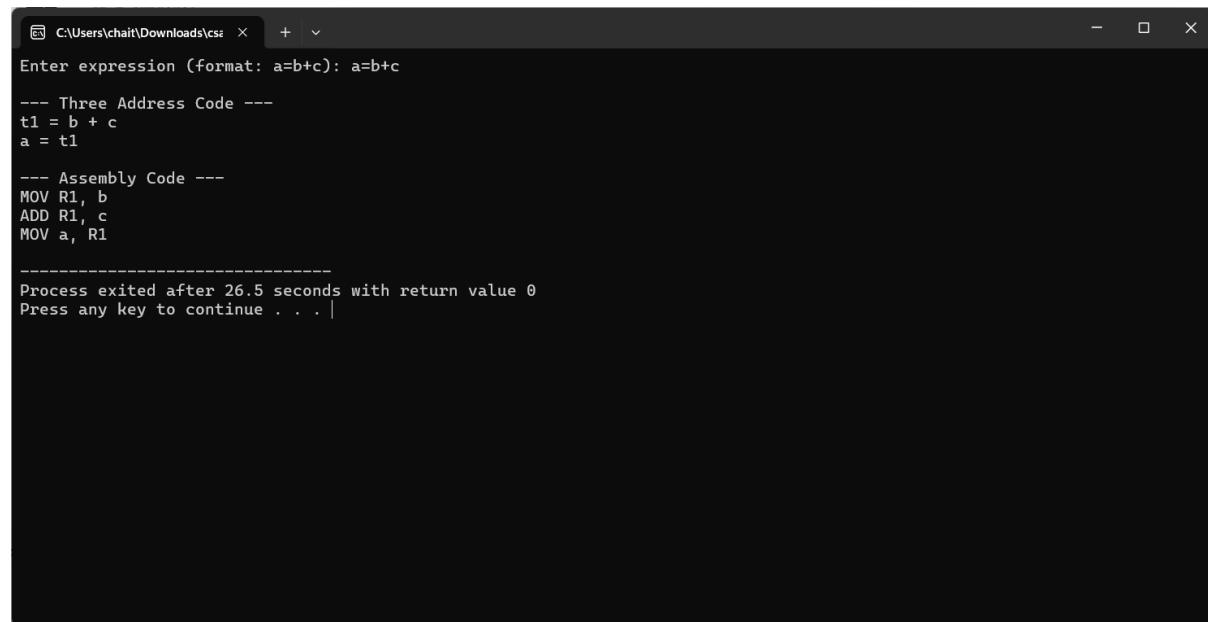
    // Format expected: a=b+c or a=b*c
    a = expr[0]; // LHS variable
    b = expr[1]; // first operand
    op = expr[2]; // operator
    c = expr[3]; // second operand

    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[1];
    op = expr[2];
    c = expr[3];
}

csai405 ex-16.cpp csai405 ex-17.cpp csai405 ex-18.cpp
```



The screenshot shows a terminal window displaying the execution of the generated assembly code. The user enters the expression "a=b+c", which is then processed by the compiler to produce the generated assembly code. The assembly code is shown in three parts: Three Address Code, Assembly Code, and the final assembly output. The terminal also shows the process exiting after 26.5 seconds.

```
C:\Users\chait\Downloads\csai> x + v
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 . . . |
```

Exp. No. 19

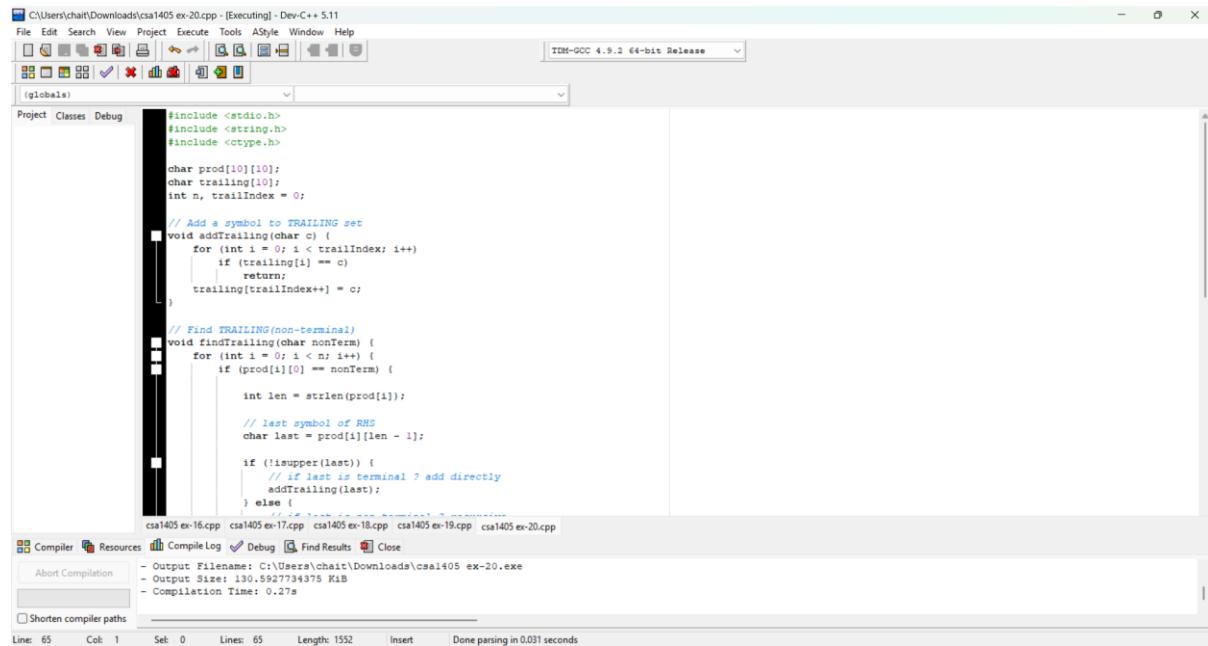
Write a C program to compute LEADING() – operator precedence parser for the given grammar

The screenshot shows the Dev-C++ IDE interface. The main window displays a C source code file named ex-19.cpp. The code implements a function to find the LEADING set of a non-terminal symbol. It includes headers for stdio.h, string.h, and ctype.h. The code uses arrays prod[10][10] and leading[10] to store productions and the LEADING set respectively. It defines two functions: addLeading, which adds a character to the LEADING set if it's not already present, and findLeading, which iterates through productions to find the LEADING set of a non-terminal. The IDE toolbar at the top includes File, Edit, Search, View, Project, Execute, Tools, ASyntax, Window, Help, and a build status bar indicating TDM-GCC 4.9.2 64-bit Release. Below the code editor is a status bar showing compilation details: Output Filename: C:\Users\chait\Downloads\csai405 ex-19.exe, Output Size: 130.58964375 Kib, Compilation Time: 0.33s.

The screenshot shows a terminal window titled 'C:\Users\chait\Downloads\csai405 ex-19' with a black background and white text. The user enters the number of productions (4) and the productions themselves (A->aB or A->a). They then enter a non-terminal to find its LEADING set (E). The output shows the LEADING set for E as an empty set {}, followed by a standard C-style exit message: 'Process exited after 14.44 seconds with return value 0'. The terminal prompt 'Press any key to continue . . .' is visible at the bottom.

Exp. No. 20

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



The screenshot shows the Dev-C++ IDE interface. The main window displays a C++ source code file named 'ex-20.cpp'. The code implements a parser for finding the trailing symbols of non-terminals in a grammar. It includes functions for adding symbols to a set and finding the trailing symbols of a non-terminal. The code uses character arrays 'prod' and 'trailing' to store grammar productions and trailing sets respectively. The IDE's status bar at the bottom shows compilation details: Output Filename: C:\Users\chait\Downloads\csai405 ex-20.exe, Output Size: 130.527734375 KB, Compilation Time: 0.27s.

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

char prod[10][10];
char trailing[10];
int n, trailIndex = 0;

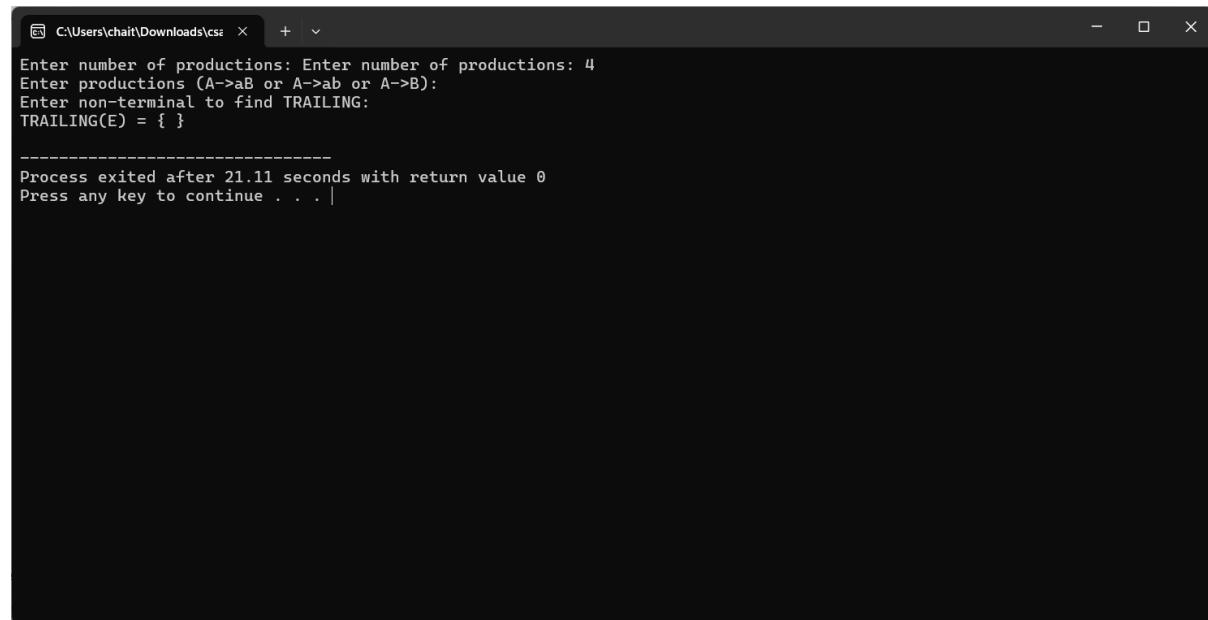
// Add a symbol to TRAILING set
void addTrailing(char c) {
    for (int i = 0; i < trailIndex; i++)
        if (trailing[i] == c)
            return;
    trailing[trailIndex++] = c;
}

// Find TRAILING(non-terminal)
void findTrailing(char nonTerm) {
    for (int i = 0; i < n; i++) {
        if (prod[i][0] == nonTerm) {

            int len = strlen(prod[i]);

            // last symbol of RHS
            char last = prod[i][len - 1];

            if (!isupper(last)) {
                // if last is terminal ? add directly
                addTrailing(last);
            } else {
                // if last is non-terminal
                findTrailing(last);
            }
        }
    }
}
```



The screenshot shows a terminal window with the command 'C:\Users\chait\Downloads\csai405 ex-20' entered. The program prompts for the number of productions (4), the productions themselves (A->aB or A->ab or A->B), and the non-terminal to find the trailing symbols for (TRAILING(E)). It then outputs the result: TRAILING(E) = { }. Finally, it shows the process exited after 21.11 seconds with a return value of 0.

```
Enter number of productions: Enter number of productions: 4
Enter productions (A->aB or A->ab or A->B):
Enter non-terminal to find TRAILING:
TRAILING(E) = { }

-----
Process exited after 21.11 seconds with return value 0
Press any key to continue . . . |
```