

## Ex-1

# Develop a lexical Analyzer to identify identifiers, constants, operators using C program.

Program:

The screenshot shows the Dev-C++ IDE interface. The main window displays a C program named 'ex-1.cpp'. The code implements a lexical analyzer to identify operators, identifiers, and constants from an input expression. It includes headers for stdio.h, ctype.h, and string.h. The main function reads an expression from standard input, tokenizes it, and prints the tokens. The code uses a global character array 'operators' containing '+-\*/=%' and a function 'isOperator' to check if a character is an operator. It also handles whitespace and digits to identify identifiers and constants. The code is annotated with comments explaining its functionality.

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
int isOperator(char c) {
    char operators[] = "+-*/=%";
    for (int i = 0; i < strlen(operators); i++) {
        if (c == operators[i])
            return 1;
    }
    return 0;
}
int main() {
    char input[100];
    int i = 0;
    printf("Enter an expression: ");
    fgets(input, 100, stdin);
    while (input[i] != '\0') {
        if (input[i] == ' ') {
            i++;
            continue;
        }
        if (isOperator(input[i])) {
            printf("Operator : %c\n", input[i]);
            i++;
        }
        // Identify Constants (numbers)
        else if (isdigit(input[i])) {
            printf("Constant : ");
            while (isdigit(input[i])) {
                printf("%c", input[i]);
                i++;
            }
        }
    }
}
```

Output:

The screenshot shows a terminal window titled 'C:\Users\chait\Downloads\css'. The user enters the expression 'a = b + 45'. The program outputs the tokens: Identifier : a, Operator : =, Identifier : b, Operator : +, Constant : 45, and Special Symbol :. The terminal then displays a standard C-style exit message: 'Process exited after 25.8 seconds with return value 0' and 'Press any key to continue . . .'

```
Enter an expression: a = b + 45
Identifier : a
Operator : =
Identifier : b
Operator : +
Constant : 45
Special Symbol :

-----
Process exited after 25.8 seconds with return value 0
Press any key to continue . . .
```

## Exp. No. 2

Develop a lexical Analyzer to identify whether a given line is a comment or not using C

The screenshot shows the Dev-C++ IDE interface. The main window displays a C++ source code file named 'ex-2.cpp'. The code implements a lexical analyzer to identify comments. It includes headers for stdio.h and string.h, defines a main function, reads a line from stdin, removes newlines, and then checks for single-line or multi-line comments based on the presence of // or /\* \*/. The code outputs the result to the console. Below the editor, the status bar shows the file name 'cs1405 ex-1.cpp' and the compilation results: Output Filename: C:\Users\chait\Downloads\cs1405 ex-2.exe, Output Size: 129.4423820125 Kib, Compilation Time: 0.23s.

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

int main() {
    char line[200];

    printf("Enter a line: ");
    fgets(line, sizeof(line), stdin);

    // Remove newline if present
    line[strcspn(line, "\n")] = 0;

    // Check single-line comment
    if (line[0] == '/' && line[1] == '/') {
        printf("It is a Single-line Comment.\n");
    }

    // Check multi-line comment
    else if (line[0] == '/' && line[1] == '*' && line[strlen(line)-2] == '*' && line[strlen(line)-1] == '/') {
        printf("It is a Multi-line Comment.\n");
    }

    // Otherwise, not a comment
    else {
        printf("It is NOT a Comment.\n");
    }

    return 0;
}
```

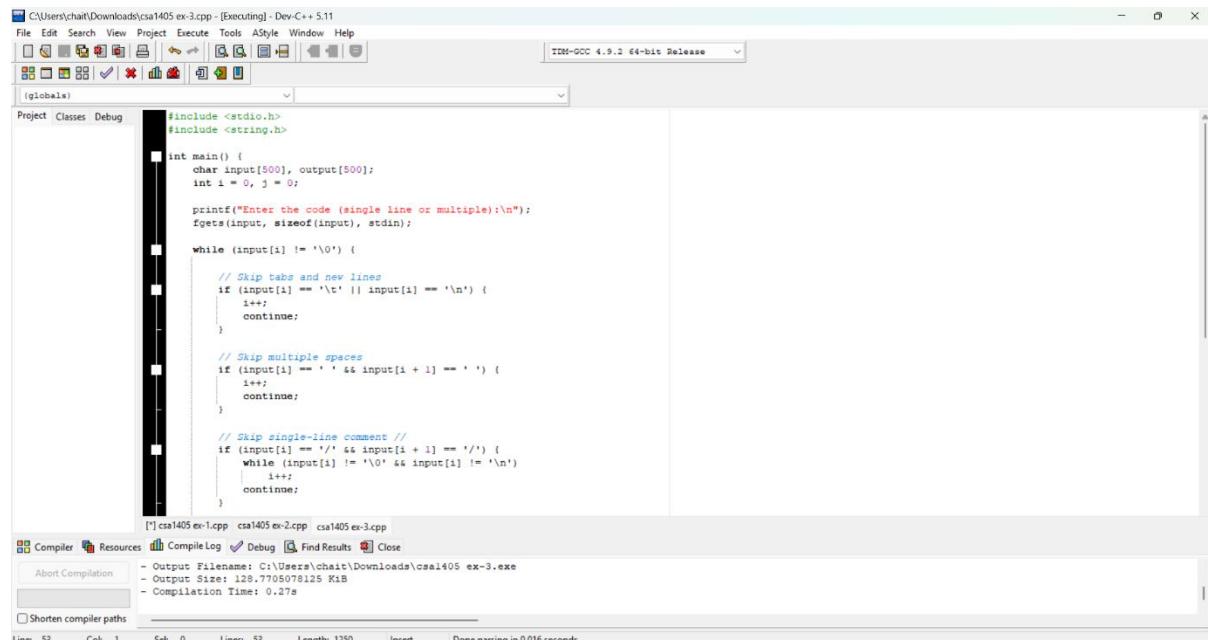
The screenshot shows a terminal window titled 'C:\Users\chait\Downloads\cs1405 ex-2.cpp'. It displays the output of the program when run with the input '/\* hello \*//\* hello \*/'. The program correctly identifies it as a multi-line comment. At the bottom, the terminal shows the process exit message: 'Process exited after 34.36 seconds with return value 0' and a prompt to press any key to continue.

```
Enter a line: /* hello *//* hello */
It is a Multi-line Comment.

-----
Process exited after 34.36 seconds with return value 0
Press any key to continue . . . |
```

## Exp. No. 3

**Design a lexical Analyzer for given language should ignore the redundant spaces, tabs and new lines and ignore comments using C**



The screenshot shows the Dev-C++ IDE interface. The main window displays a C program for a lexical analyzer. The code includes #include <stdio.h> and #include <string.h>. It defines a main() function that reads input from stdin. The program uses a while loop to iterate through the input. Inside the loop, it checks for tabs and new lines, and skips them if found. It also handles multiple spaces by skipping consecutive whitespace characters. Additionally, it ignores single-line comments starting with //.

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

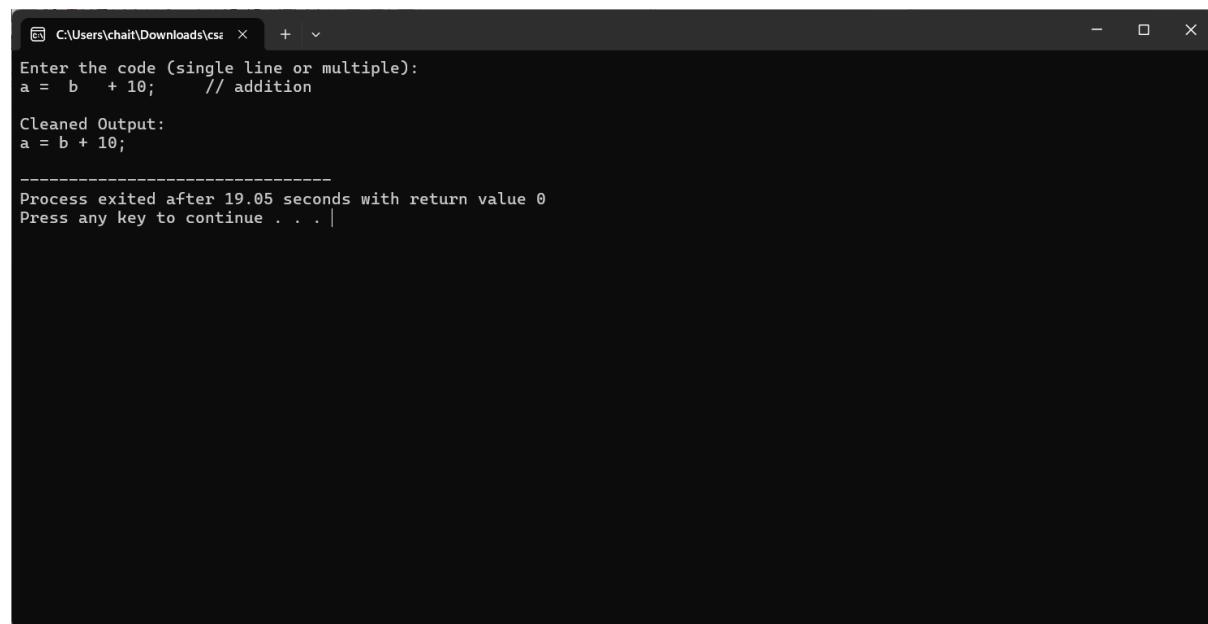
int main() {
    char input[500], output[500];
    int i = 0, j = 0;

    printf("Enter the code (single line or multiple):\n");
    fgets(input, sizeof(input), stdin);

    while (input[i] != '\0') {
        // Skip tabs and new lines
        if (input[i] == '\t' || input[i] == '\n') {
            i++;
            continue;
        }

        // Skip multiple spaces
        if (input[i] == ' ' && input[i + 1] == ' ') {
            i++;
            continue;
        }

        // Skip single-line comment //
        if (input[i] == '/' && input[i + 1] == '/') {
            while (input[i] != '\0' && input[i] != '\n') {
                i++;
            }
            continue;
        }
    }
}
```



The screenshot shows a terminal window titled "C:\Users\chait\Downloads\csa". The user enters the command "a = b + 10; // addition". The program outputs "Cleaned Output: a = b + 10;" followed by a separator line and the process exit message. The terminal window has a dark theme.

```
Enter the code (single line or multiple):
a = b + 10; // addition

Cleaned Output:
a = b + 10;

-----
Process exited after 19.05 seconds with return value 0
Press any key to continue . . . |
```

## Exp. No. 4

# Design a lexical Analyzer to validate operators to recognize the operators +,-,\*,/ using regular arithmetic operators using C

The screenshot shows the Dev-C++ IDE interface. The main window displays a C++ source code file named 'ex-4.cpp'. The code implements a simple lexical analyzer to validate arithmetic operators. It includes a header file inclusion, a main function, and a switch statement for operators '+', '-', '\*', and '/'. The code is highlighted in different colors (black, red, green) to indicate syntax. Below the editor, the status bar shows the current line (Line: 28), column (Col: 1), and other details. The bottom panel shows the compiler output, indicating a successful compilation of 'ex-4.cpp' into 'ex-4.exe'.

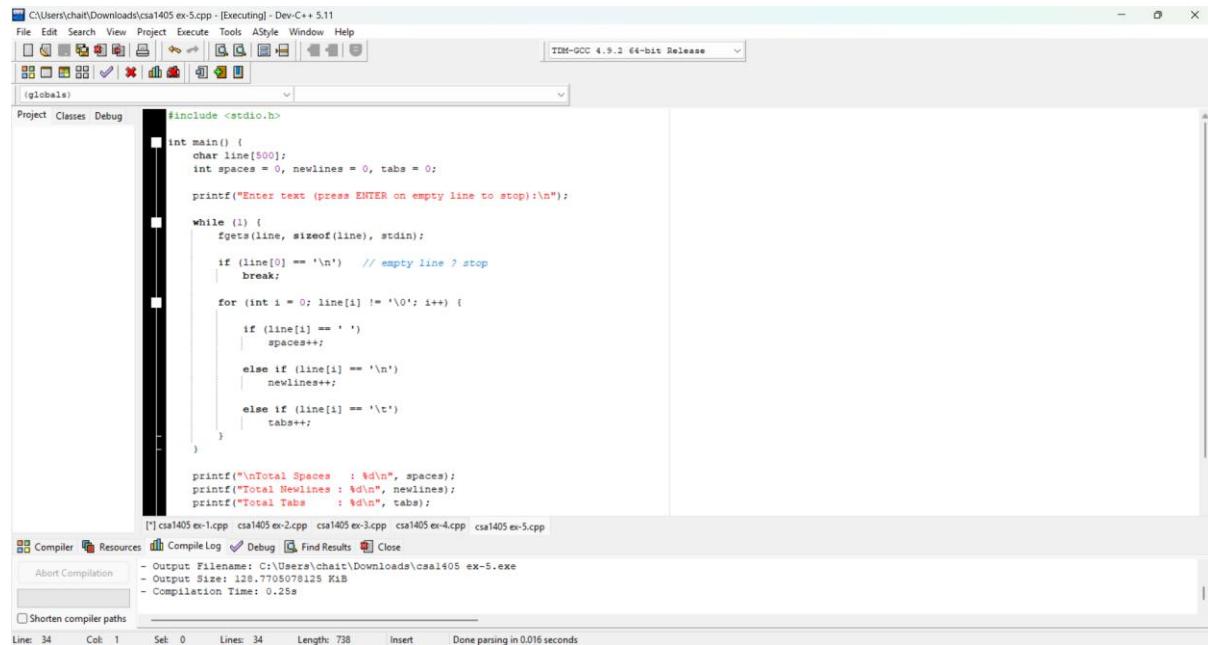
```
#include <stdio.h>
int main() {
    char op;
    printf("Enter an operator: ");
    scanf("%c", &op);
    switch (op) {
        case '+':
            printf("Valid Operator: +\n");
            break;
        case '-':
            printf("Valid Operator: -\n");
            break;
        case '*':
            printf("Valid Operator: *\n");
            break;
        case '/':
            printf("Valid Operator: /\n");
            break;
        default:
            printf("Invalid Operator\n");
    }
    return 0;
}
```

The screenshot shows a terminal window titled 'C:\Users\chait\Downloads\csa'. The window displays the output of the program 'ex-4.exe'. The user enters the character '+' as input, and the program outputs 'Valid Operator: +' followed by a new line. At the end of the output, there is a standard Windows-style message: 'Process exited after 20.41 seconds with return value 0' and 'Press any key to continue . . . |'.

```
Enter an operator: +
Valid Operator: +
```

## Exp. No. 5

# Design a lexical Analyzer to find the number of whitespaces and newline characters using C.



The screenshot shows the Dev-C++ IDE interface. The main window displays a C++ source code file named 'csa1405 ex-5.cpp'. The code implements a lexical analyzer to count spaces, newlines, and tabs from user input. The IDE's status bar at the bottom shows the current line (Line: 34), column (Col: 1), and other build-related information.

```
#include <stdio.h>

int main() {
    char line[500];
    int spaces = 0, newlines = 0, tabs = 0;

    printf("Enter text (press ENTER on empty line to stop):\n");

    while (1) {
        fgets(line, sizeof(line), stdin);

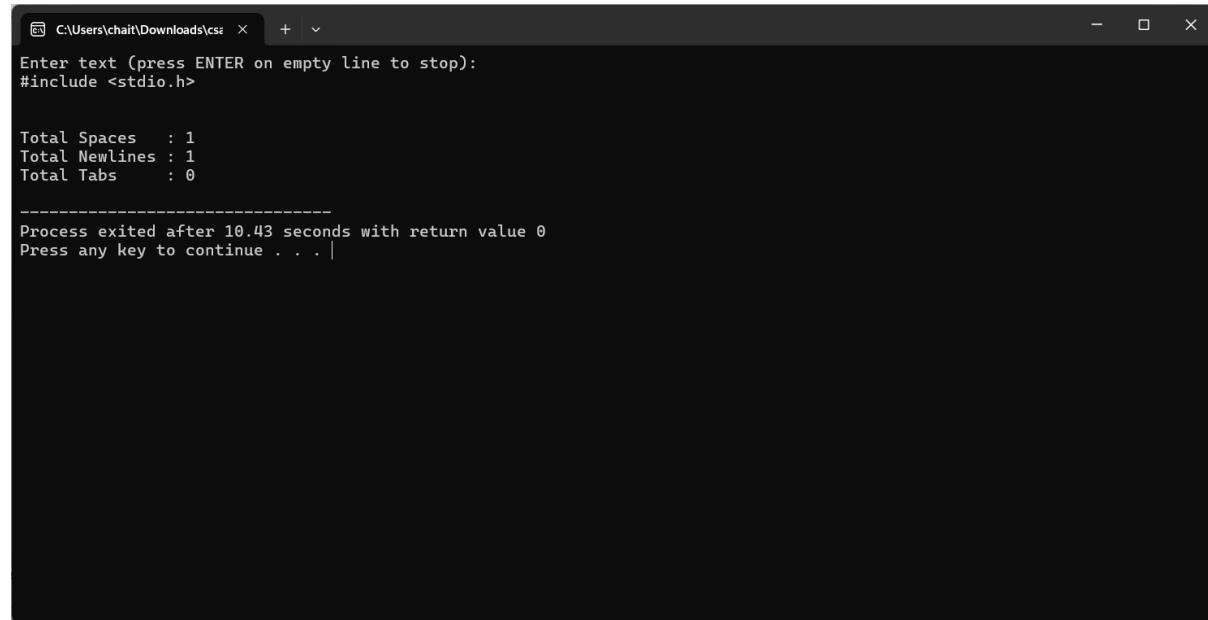
        if (line[0] == '\n') // empty line ? stop
            break;

        for (int i = 0; line[i] != '\0'; i++) {
            if (line[i] == ' ')
                spaces++;

            else if (line[i] == '\n')
                newlines++;

            else if (line[i] == '\t')
                tabs++;
        }

        printf("\nTotal Spaces : %d\n", spaces);
        printf("Total Newlines : %d\n", newlines);
        printf("Total Tabs : %d\n", tabs);
    }
}
```



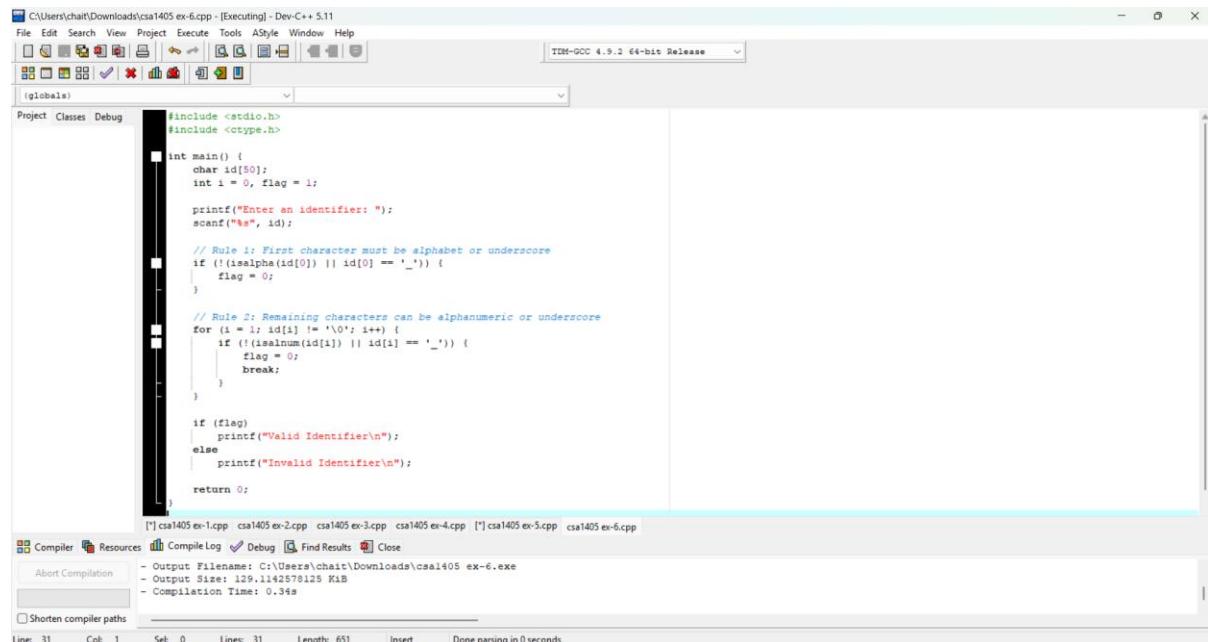
The screenshot shows a terminal window titled 'C:\Users\chait\Downloads\css'. It displays the output of the program, which counts the number of spaces, newlines, and tabs in the input text. The terminal also shows the compilation command and the execution time.

```
Enter text (press ENTER on empty line to stop):
#include <stdio.h>

Total Spaces : 1
Total Newlines : 1
Total Tabs : 0
-----
Process exited after 10.43 seconds with return value 0
Press any key to continue . . . |
```

## Exp. No. 6

Develop a lexical Analyzer to test whether a given identifier is valid or not using C.



The screenshot shows the Dev-C++ IDE interface. The main window displays a C++ source code file named 'ex-6.cpp'. The code implements a lexical analyzer to validate identifiers based on two rules:

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

int main() {
    char id[50];
    int i = 0, flag = 1;

    printf("Enter an identifier: ");
    scanf("%s", id);

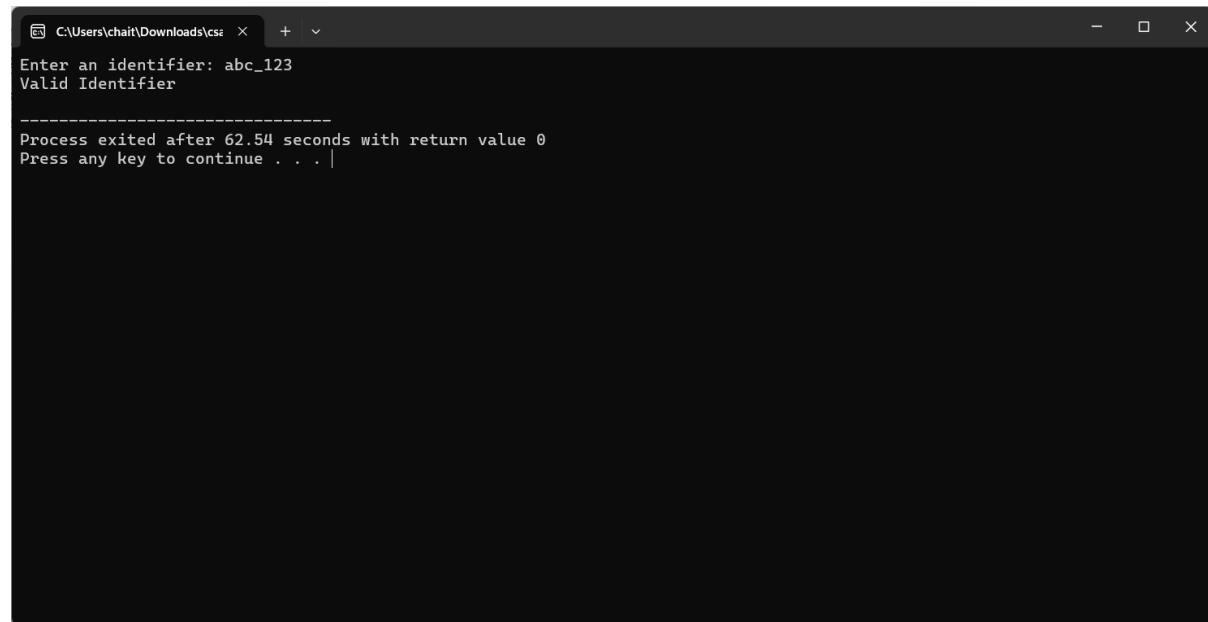
    // Rule 1: First character must be alphabet or underscore
    if (!isalpha(id[0]) || id[0] == '_') {
        flag = 0;
    }

    // Rule 2: Remaining characters can be alphanumeric or underscore
    for (i = 1; id[i] != '\0'; i++) {
        if (!isalnum(id[i]) || id[i] == '_') {
            flag = 0;
            break;
        }
    }

    if (flag)
        printf("Valid Identifier\n");
    else
        printf("Invalid Identifier\n");
}

return 0;
}
```

The status bar at the bottom shows the command line: '[] csa1405 ex-1.cpp csa1405 ex-2.cpp csa1405 ex-3.cpp csa1405 ex-4.cpp [!] csa1405 ex-5.cpp csa1405 ex-6.cpp'.



The screenshot shows a terminal window titled 'C:\Users\chait\Downloads\csa'. The user enters 'abc\_123' and the program outputs 'Valid Identifier'. The terminal also displays the process exit information and a prompt to press any key.

```
Enter an identifier: abc_123
Valid Identifier

-----
Process exited after 62.54 seconds with return value 0
Press any key to continue . . . |
```

## Exp. No. 7

**Write a C program to find FIRST( ) - predictive parser for the given grammar**

The screenshot shows the Dev-C++ IDE interface. The main window displays a C++ source code file named 'csa1405 ex-7.cpp'. The code implements a predictive parser for finding FIRST sets. It includes headers for stdio.h, ctype.h, and string.h, and defines arrays prod and first. The function findFirst takes a firstSet array and a symbol, then iterates through productions to find the FIRST set for the symbol. The IDE's status bar at the bottom shows the compilation was successful with a size of 130.7001953125 KB and a time of 0.33s.

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

char prod[10][10];
char first[10][10];
int nr;

void findFirst(char firstSet[], char symbol) {
    int i, j;

    // If terminal ? FIRST is the terminal itself
    if (!isupper(symbol)) {
        firstSet[strlen(firstSet)] = symbol;
        return;
    }

    for (i = 0; i < nr; i++) {
        if (prod[i][0] == symbol) { // match A ?
            if (prod[i][2] == '\0') { // terminal or epsilon
                firstSet[strlen(firstSet)] = prod[i][2];
            } else { // non-terminal ? recursion
                findFirst(firstSet, prod[i][2]);
            }
        }
    }
}

int main() {
    // Your main function code here
}
```

The screenshot shows a terminal window titled 'C:\Users\chait\Downloads\csa'. The user enters the number of productions (4) and provides the productions: A->aB, A->Bc, B->b. They then enter the symbol to find FIRST (B->#) and receive the output FIRST(B) = { > }. Finally, the terminal displays the process exit message and a prompt to press any key.

```
Enter number of productions: 4
Enter productions (Example: A->aB):
4
A->aB
A->Bc
B->b
Enter the symbol to find FIRST: B->#
FIRST(B) = { > }

-----
Process exited after 11.08 seconds with return value 0
Press any key to continue . . .
```

## Exp. No. 8

**Write a C program to find FOLLOW( ) - predictive parser for the given grammar**

The screenshot shows the Dev-C++ IDE interface. The main window displays a C source code file named 'ex-8.cpp'. The code implements a predictive parser for a grammar with productions like S->AB, A->aA, and A->#. It includes functions for finding first sets and follow sets. The code uses arrays 'prod' and 'firstSet' to store grammar rules and first sets respectively. The 'firstSet' array is initialized with '#'. The 'findFirst' function adds symbols to the first set based on grammar rules. The 'findFollow' function adds symbols to the follow set of a non-terminal. The 'main' function prompts the user for the number of productions and the productions themselves, then asks for a symbol to find its FOLLOW set. The output window at the bottom shows the command-line interaction and the generated executable's output.

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

char prod[10][10];
int n;

void findFirst(char[], char);
void findFollow(char[], char);

void findFirst(char firstSet[], char symbol) {
    int i;

    if (!isupper(symbol)) {
        int len = strlen(firstSet);
        firstSet[len] = symbol;
        return;
    }

    for (i = 0; i < n; i++) {
        if (prod[i][0] == symbol) {
            if (prod[i][3] == '#') {
                int len = strlen(firstSet);
                firstSet[len] = '#';
            }
            else if (!isupper(prod[i][3])) {
                int len = strlen(firstSet);
                firstSet[len] = prod[i][3];
            }
            else {
                findFirst(firstSet, prod[i][3]);
            }
        }
    }
}

void findFollow(char firstSet[], char symbol) {
    int i;

    for (i = 0; i < n; i++) {
        if (prod[i][0] == symbol) {
            if (prod[i][3] == '#') {
                int len = strlen(firstSet);
                firstSet[len] = '#';
            }
            else if (!isupper(prod[i][3])) {
                int len = strlen(firstSet);
                firstSet[len] = prod[i][3];
            }
            else {
                findFollow(firstSet, prod[i][3]);
            }
        }
    }
}

int main() {
    int numProd;
    char prodStr[100];
    char symbol;
    char firstSet[100];

    printf("Enter number of productions: ");
    scanf("%d", &numProd);

    printf("Enter productions (Example: A->aB):\n");
    for (i = 0; i < numProd; i++) {
        gets(prodStr);
        sscanf(prodStr, "%s->%s", prod[i], prod[i]+3);
    }

    printf("Enter the symbol to find FOLLOW: ");
    scanf(" %c", &symbol);

    findFollow(firstSet, symbol);

    printf("FOLLOW(%c) = { %s }\n", symbol, firstSet);

    return 0;
}
```

The screenshot shows a terminal window titled 'C:\Users\chait\Downloads\css'. The window displays the command-line interaction for running the program. The user enters the number of productions (4), the productions (S->AB, A->aA, A->#), and the symbol to find the FOLLOW set for (B). The program outputs the FOLLOW set for B as {} and then exits. The terminal also shows the process exit time and a prompt to press any key to continue.

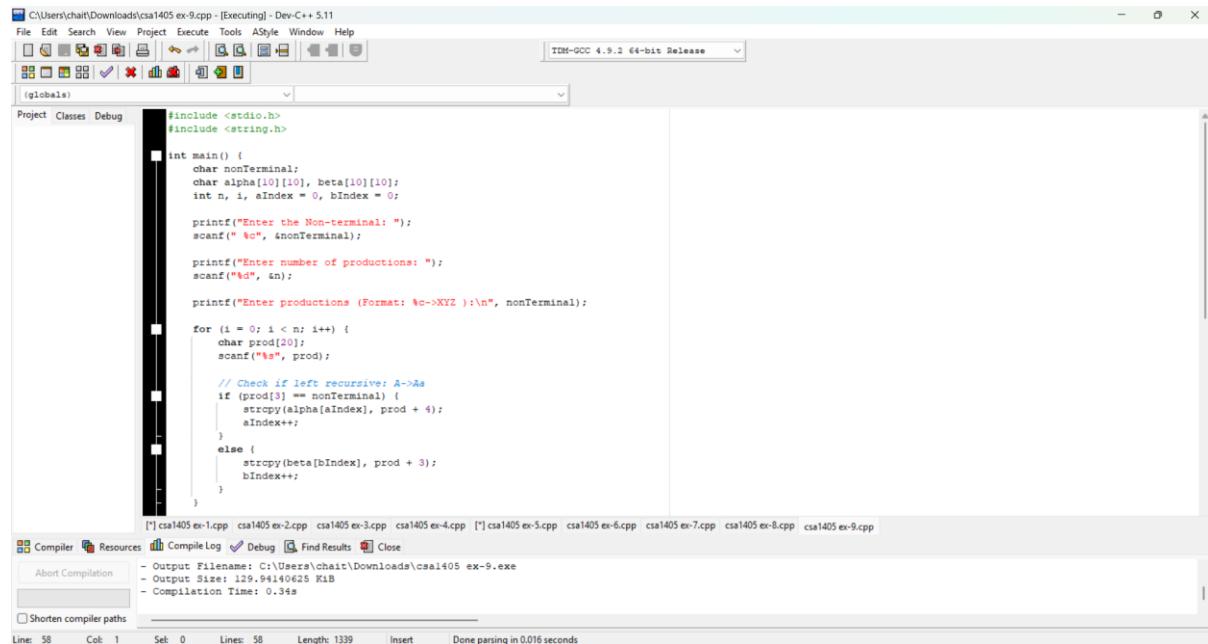
```
C:\Users\chait\Downloads\css > + v

Enter number of productions: 4
Enter productions (Example: A->aB):
4
S->AB
A->aA
A->#
Enter the symbol to find FOLLOW: B->b
FOLLOW(B) = { }

-----
Process exited after 32.89 seconds with return value 0
Press any key to continue . . . |
```

## Exp. No. 9

# Implement a C program to eliminate left recursion from a given CFG.



The screenshot shows the Dev-C++ IDE interface. The code editor window displays a C program named ex-9.cpp. The code reads input from the user to determine a non-terminal character, the number of productions, and the productions themselves. It then checks for left recursion in each production rule and eliminates it by transforming it into a non-left-recursive form. The compiler log at the bottom shows the compilation process, output filename (csa1405 ex-9.exe), output size (129.94 KB), and compilation time (0.34s). The status bar at the bottom indicates the current line (Line: 58), column (Col: 1), selection (Sel: 0), and total lines (Lines: 58).

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

int main() {
    char nonTerminal;
    char alpha[10][10], beta[10][10];
    int n, i, aIndex = 0, bIndex = 0;

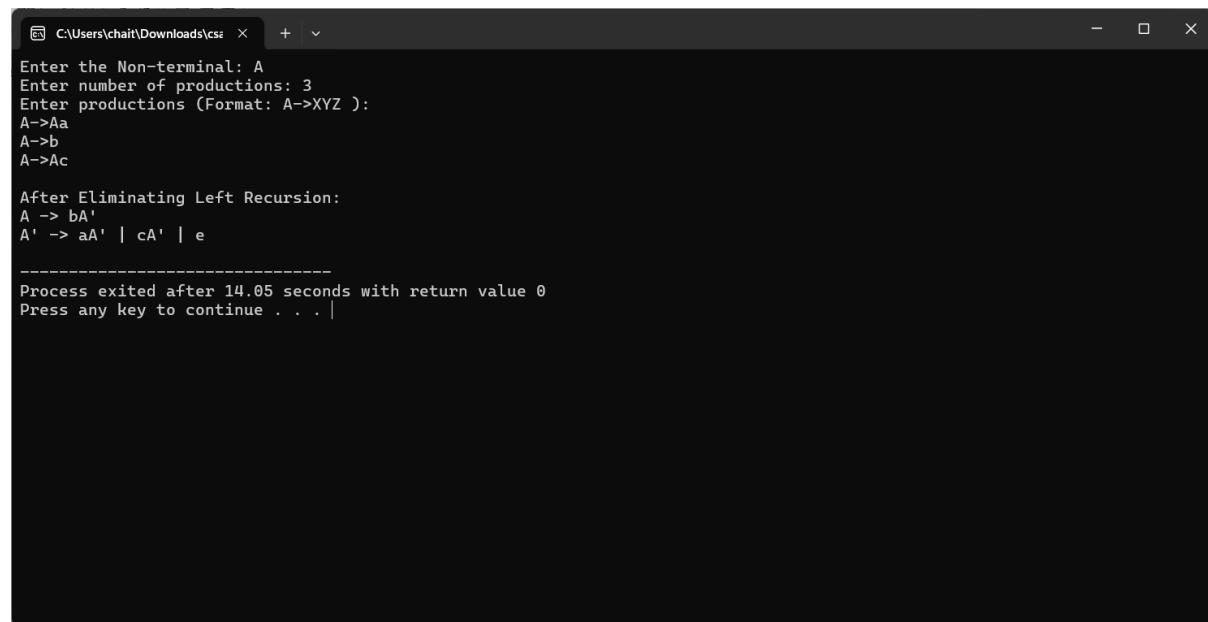
    printf("Enter the Non-terminal: ");
    scanf(" %c", &nonTerminal);

    printf("Enter number of productions: ");
    scanf("%d", &n);

    printf("Enter productions (Format: %c->XYZ ): \n", nonTerminal);

    for (i = 0; i < n; i++) {
        char prod[20];
        scanf("%s", prod);

        // Check if left recursive: A->Aa
        if (prod[3] == nonTerminal) {
            strcpy(alpha[aIndex], prod + 4);
            aIndex++;
        }
        else {
            strcpy(beta[bIndex], prod + 3);
            bIndex++;
        }
    }
}
```



The screenshot shows a terminal window with the command C:\Users\chait\Downloads\csa1405 ex-9.exe entered. The program prompts the user for a non-terminal character (A), the number of productions (3), and the productions themselves (A->Aa, A->b, A->Ac). After processing, it outputs the result of eliminating left recursion: A -> bA' and A' -> aA' | cA' | e. Finally, it displays the process exit message and a prompt to press any key to continue.

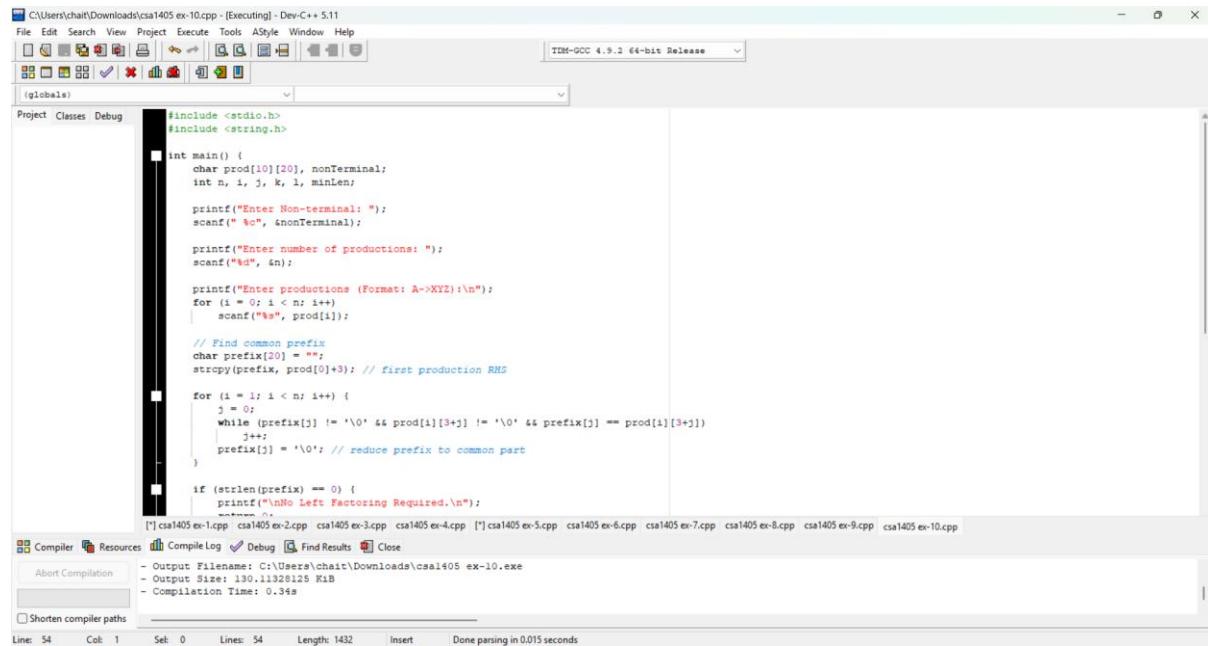
```
Enter the Non-terminal: A
Enter number of productions: 3
Enter productions (Format: A->XYZ ):
A->Aa
A->b
A->Ac

After Eliminating Left Recursion:
A -> bA'
A' -> aA' | cA' | e

-----
Process exited after 14.05 seconds with return value 0
Press any key to continue . . . |
```

## Exp. No. 10

# Implement a C program to eliminate left factoring from a given CFG.



The screenshot shows the Dev-C++ IDE interface. The code editor displays a C program named ex-10.cpp. The code reads input for a non-terminal symbol and the number of productions, then asks for productions in the format A->XYZ. It then performs left factorization by finding common prefixes among the RHS of the productions. The compiler log at the bottom shows the compilation process and output file details.

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

int main() {
    char prod[10][20], nonTerminal;
    int n, i, j, k, l, minLen;

    printf("Enter Non-terminal: ");
    scanf(" %c", &nonTerminal);

    printf("Enter number of productions: ");
    scanf("%d", &n);

    printf("Enter productions (Format: A->XYZ):\n");
    for (i = 0; i < n; i++)
        scanf("%s", prod[i]);

    // Find common prefix
    char prefix[20] = "";
    strcpy(prefix, prod[0]+3); // first production RHS

    for (i = 1; i < n; i++) {
        j = 0;
        while (prefix[j] != '\0' && prod[i][3+j] != '\0' && prefix[j] == prod[i][3+j])
            j++;
        prefix[j] = '\0'; // reduce prefix to common part
    }

    if (strlen(prefix) == 0) {
        printf("\nNo Left Factoring Required.\n");
        return 0;
    }
}

[1] csa1405 ex-1.cpp csa1405 ex-2.cpp csa1405 ex-3.cpp csa1405 ex-4.cpp [1] csa1405 ex-5.cpp csa1405 ex-6.cpp csa1405 ex-7.cpp csa1405 ex-8.cpp csa1405 ex-9.cpp csa1405 ex-10.cpp
```

Compiler Resources Compile Log Debug Find Results Close

Abort Compilation

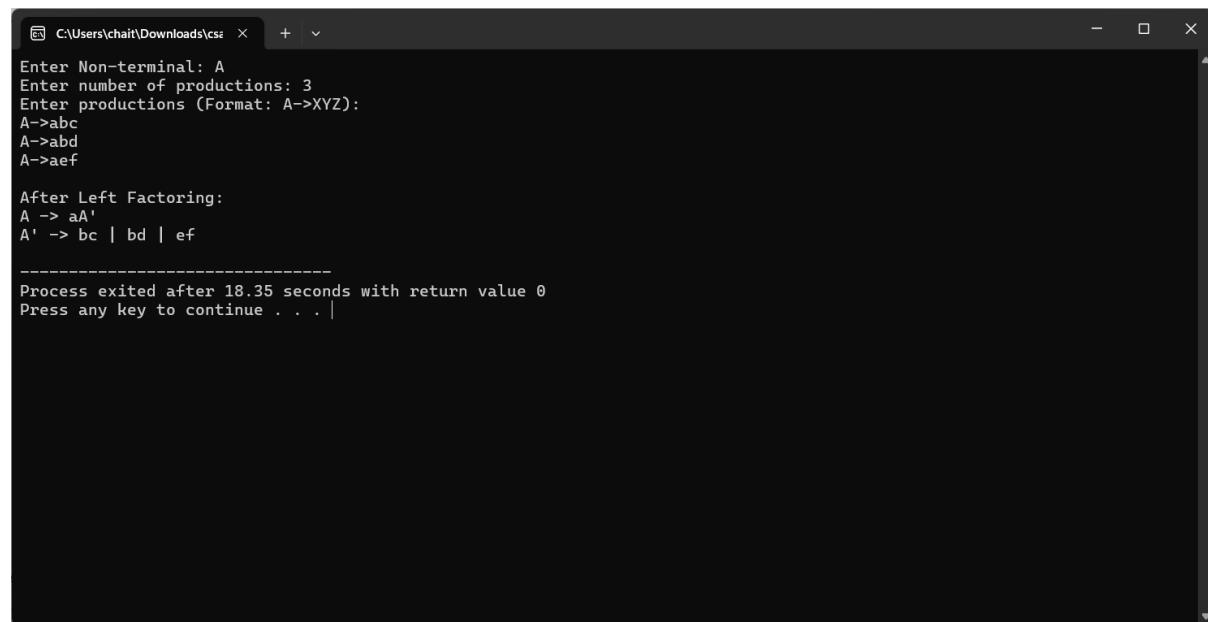
Output Filename: C:\Users\chait\Downloads\csa1405 ex-10.exe

Output Size: 130.11328125 Kib

Compilation Time: 0.34s

Shorten compiler paths

Line: 54 Col: 1 Sel: 0 Lines: 54 Length: 1432 Insert Done parsing in 0.015 seconds



The screenshot shows a terminal window with the following interaction:

```
Enter Non-terminal: A
Enter number of productions: 3
Enter productions (Format: A->XYZ):
A->abc
A->abd
A->aef

After Left Factoring:
A -> aA'
A' -> bc | bd | ef

-----
Process exited after 18.35 seconds with return value 0
Press any key to continue . . . |
```

## Exp. No. 11

# Implement a C program to perform symbol table operations.

The screenshot shows the Dev-C++ IDE interface. The code editor displays a C program for a symbol table. The code includes a struct for symbols, a main function with a menu loop, and logic for inserting, displaying, searching, and exiting. The status bar at the bottom shows compilation details: Output Filename: C:\Users\chait\Downloads\csai405 ex-11.exe, Output Size: 129,941,406,251 Kib, Compilation Time: 0.36s.

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

struct Symbol {
    char name[20];
    char type[10];
};

int main() {
    struct Symbol table[50];
    int n = 0; // number of symbols
    int choice;

    while (1) {
        printf("\n--- Symbol Table Operations ---");
        printf("1. Insert Symbol\n");
        printf("2. Display Symbol Table\n");
        printf("3. Search Symbol\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            printf("Enter Symbol Name: ");
            scanf("%s", table[n].name);
            printf("Enter Symbol Type: ");
            scanf("%s", table[n].type);
            n++;
        }
        else if (choice == 2) {
            printf("Symbol Table:\n");
            for (int i = 0; i < n; i++) {
                printf("%d. %s %s\n", i + 1, table[i].name, table[i].type);
            }
        }
        else if (choice == 3) {
            printf("Enter Symbol Name: ");
            scanf("%s", table[n].name);
            printf("Symbol found: %s\n", table[n].name);
        }
        else if (choice == 4) {
            break;
        }
    }
}
```

The screenshot shows a terminal window displaying the execution of the program. It starts with the menu, then inserts a symbol ('X' of type 'int'). It then displays the symbol table, which shows one entry: X of type int. Finally, it exits. The terminal window also shows the exit message and the process exit time.

```
--- Symbol Table Operations ---
1. Insert Symbol
2. Display Symbol Table
3. Search Symbol
4. Exit
Enter your choice: 1
Enter Symbol Name: X
Enter Symbol Type: int

--- Symbol Table Operations ---
1. Insert Symbol
2. Display Symbol Table
3. Search Symbol
4. Exit
Enter your choice: 2

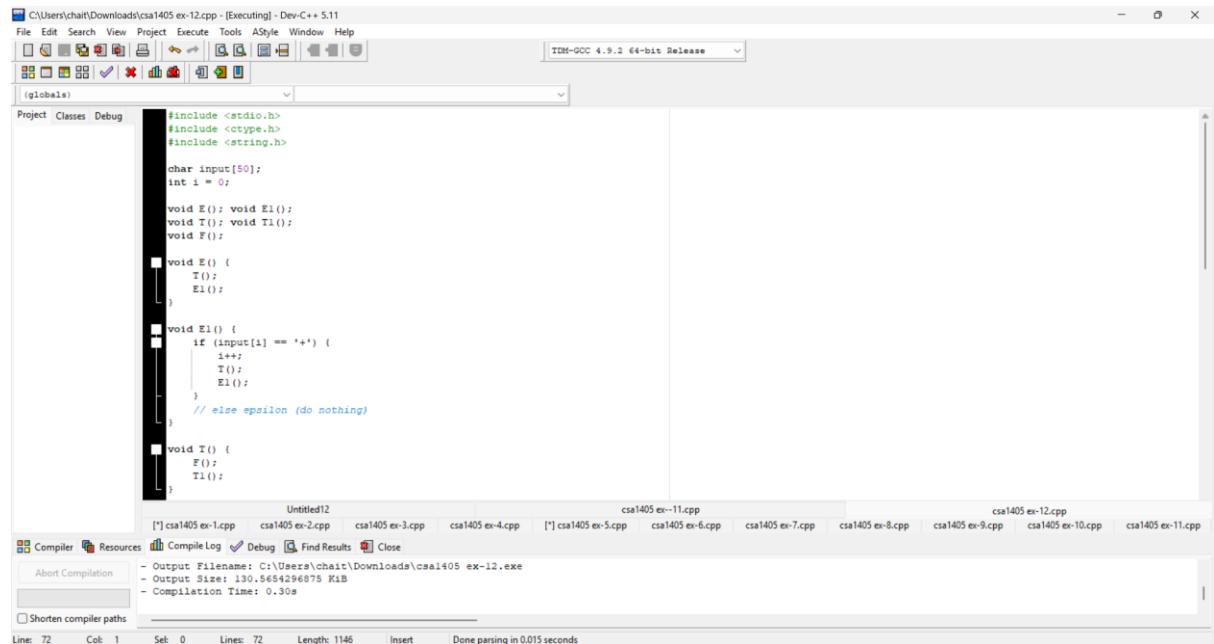
Symbol Table:
S.No      Name      Type
1          X         int

--- Symbol Table Operations ---
1. Insert Symbol
2. Display Symbol Table
3. Search Symbol
4. Exit
Enter your choice: 4

-----
Process exited after 52.32 seconds with return value 0
```

## Exp. No. 12

# Write a C program to construct recursive descent parsing for the given grammar



The screenshot shows the Dev-C++ IDE interface. The main window displays a C source code file named 'csa1405 ex-12.cpp'. The code implements a recursive descent parser for a grammar. It includes declarations for stdio.h, ctype.h, and string.h, and defines functions E(), E1(), T(), T1(), and F(). The E() function calls T() and E1(). The E1() function checks if the next character is '+', and if so, increments the index and calls T() followed by E1(). If not, it does nothing. The T() function calls F() and T1(). The compiler log at the bottom shows the compilation process and successful execution of the program.

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

char input[50];
int i = 0;

void E(); void E1();
void T(); void T1();
void F();

void E() {
    T();
    E1();
}

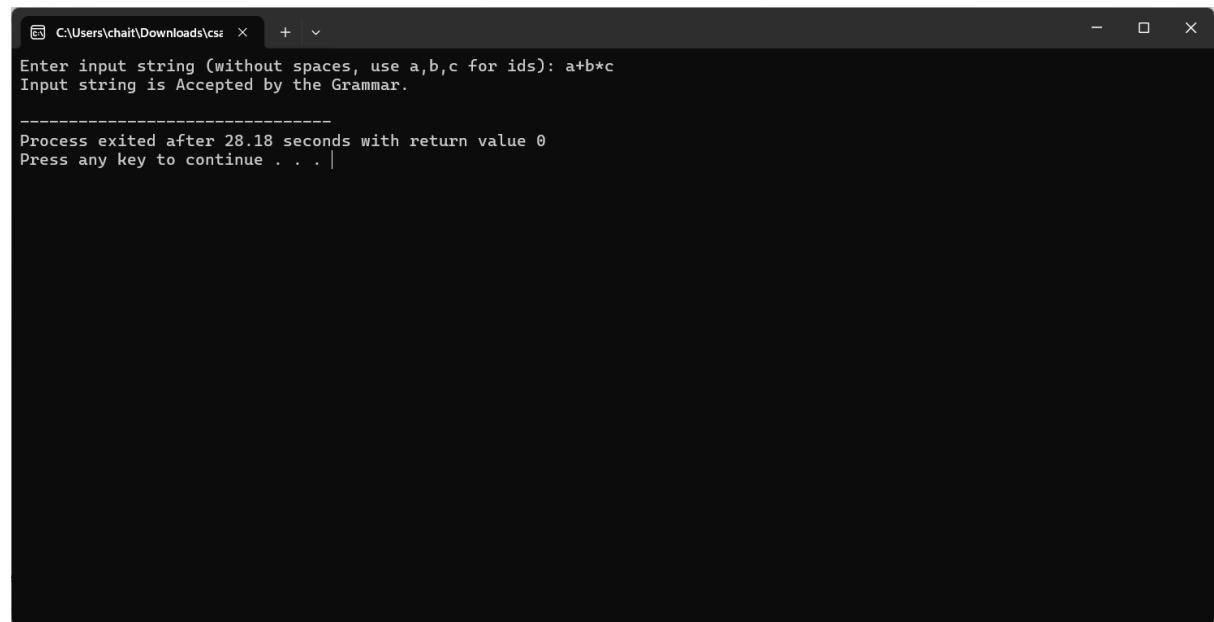
void E1() {
    if (input[i] == '+') {
        i++;
        T();
        E1();
    }
    // else epsilon (do nothing)
}

void T() {
    F();
    T1();
}

void T1() {
}
```

Compiler Log:

- Output Filename: C:\Users\chait\Downloads\csa1405 ex-12.exe
- Output Size: 130.654296875 KiB
- Compilation Time: 0.30s



The screenshot shows a terminal window with the command prompt 'C:\Users\chait\Downloads\csa1405 ex-12.exe'. The user enters the input string 'a+b\*c' and receives the message 'Input string is Accepted by the Grammar.' The terminal also displays the process exit information and a prompt to press any key to continue.

```
Enter input string (without spaces, use a,b,c for ids): a+b*c
Input string is Accepted by the Grammar.

-----
Process exited after 28.18 seconds with return value 0
Press any key to continue . . . |
```

## Exp. No. 13

**Write a C program to implement either Top Down parsing technique or Bottom Up Parsing technique to check whether the given input string is satisfying the grammar or not.**

The screenshot shows the Dev-C++ IDE interface. The code editor contains a C program for implementing a Top Down parser. The code includes declarations for `char input[50]`, `int i = 0;`, and three functions: `E()`, `T()`, and `F()`. The `E()` function handles the base case and recursive cases for  $E \rightarrow T E'$  and  $E' \rightarrow + T E' \mid \epsilon$ . The `T()` and `F()` functions handle the terminal symbols. The code uses nested if statements and assignment operators to manage the state and input index `i`.

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

char input[50];
int i = 0;

void E(); void E1();
void T(); void T1();
void F();

// E -> T E'
void E() {
    T();
    E1();
}

// E' -> + T E' | ε
void E1() {
    if (input[i] == '+') {
        i++;
        T();
        E1();
    }
}

// T -> F T'
void T() {
    F();
}
```

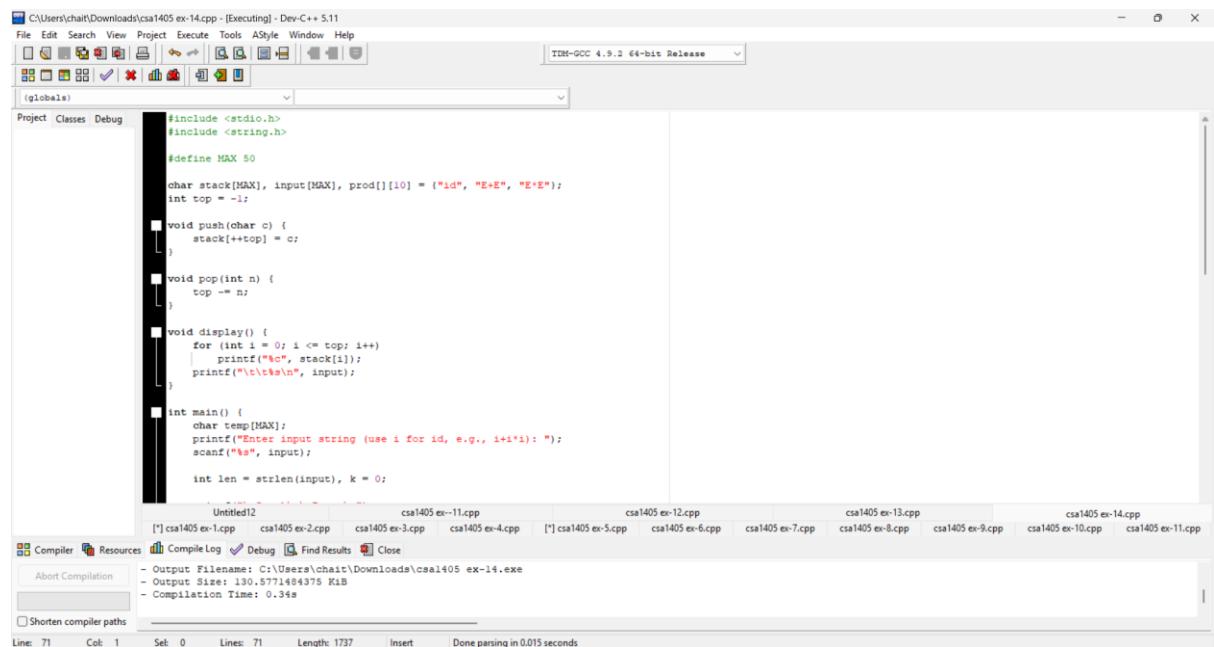
The screenshot shows a terminal window titled "C:\Users\chait\Downloads\csa". The user has entered the command "a+b\*c" and the program has accepted it. The terminal also displays the compilation log and execution time.

```
Enter input string (without spaces, use single letters for id): a+b*c
Input string is Accepted by the Grammar.

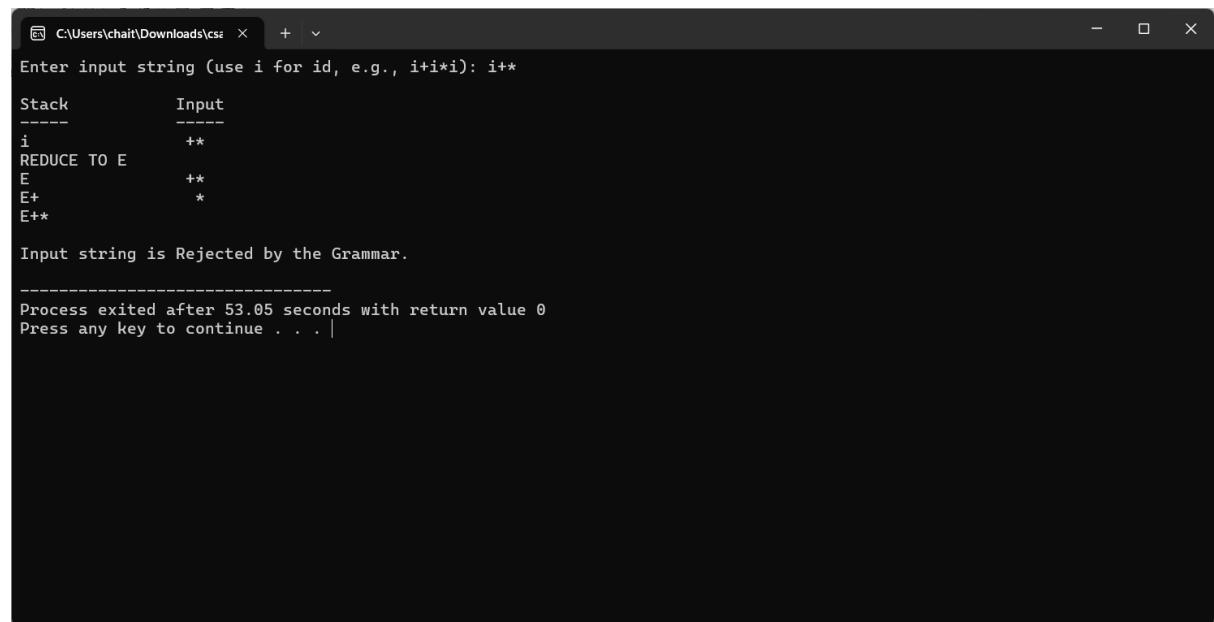
-----
Process exited after 43.05 seconds with return value 0
Press any key to continue . . .
```

## Exp. No. 14

### Implement the concept of Shift reduce parsing in C Programming.



The screenshot shows the Dev-C++ IDE interface. The main window displays a C++ source file named 'Untitled12.cpp' containing code for a parser. The code includes #include directives for stdio.h and string.h, defines MAX as 50, and implements stack operations (push, pop, display) and a main function. The main function reads input from the user and processes it using the defined functions. Below the code editor is a status bar showing compilation details: Output Filename: C:\Users\chait\Downloads\csa1405 ex-14.exe, Output Size: 130.591484375 KIB, Compilation Time: 0.34s. The bottom of the screen shows the Windows taskbar.



The screenshot shows a terminal window with the command 'C:\Users\chait\Downloads\csa1405 ex-14.exe' entered. The output shows the parser's internal state (Stack and Input) and the reduction steps for the input 'i\*'. The output concludes with 'Input string is Rejected by the Grammar.' and 'Process exited after 53.05 seconds with return value 0'. The terminal window has a dark background.

Exp. No. 15

Write a C Program to implement the operator precedence parsing.

```
C:\Users\chait\Downloads\cse x + v
Enter expression (use i for id, e.g., i+i*i): i*i

Stack      Input      Action
----      ----      -----
$i          *i      Shift
$i          *i      Reduce
$*          i       Shift
$*i

Error: Invalid Expression

-----
Process exited after 18.56 seconds with return value 0
Press any key to continue . . . |
```