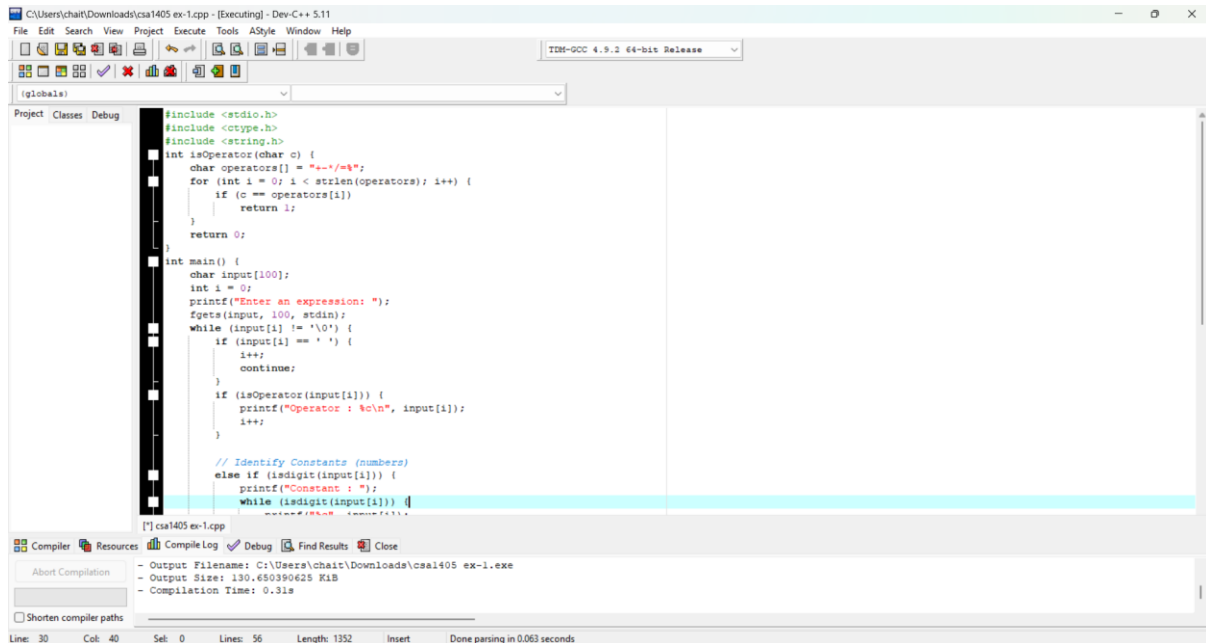


Ex-1

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

Program:



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

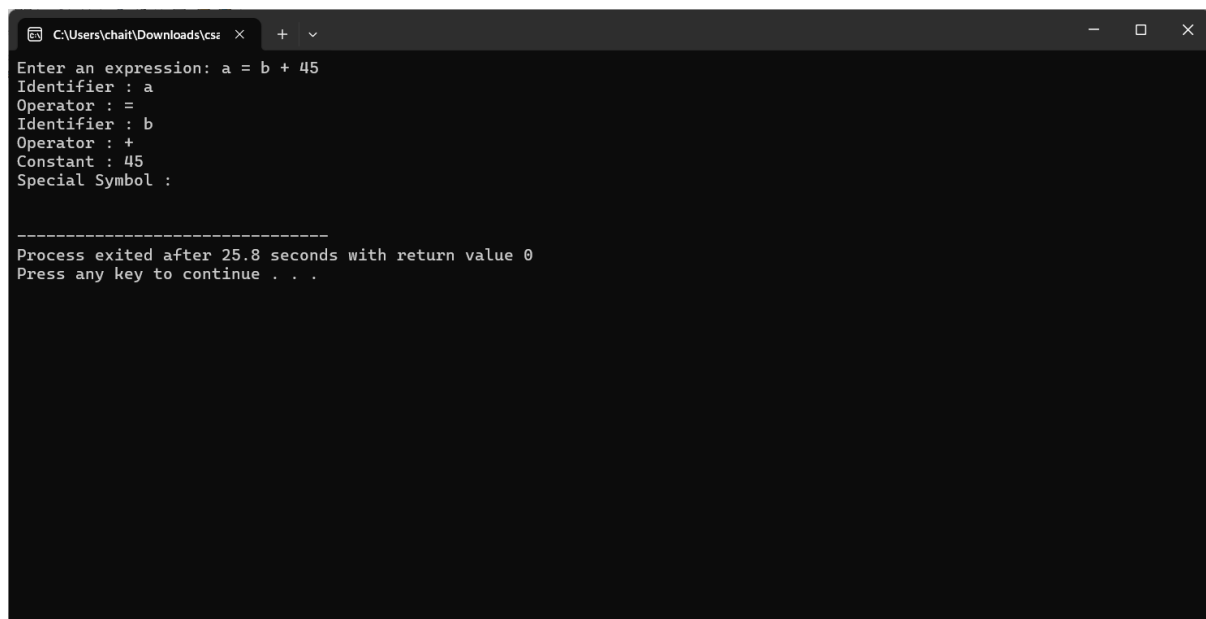
int isOperator(char c) {
    char operators[] = "+-*/%";
    for (int i = 0; i < strlen(operators); i++) {
        if (c == operators[i])
            return 1;
    }
    return 0;
}

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("%s", input[i]);
                i++;
            }
            printf("\n");
        }
    }
}
```

Compiler Output:

```
- Output Filename: C:\Users\chait\Downloads\csal405 ex-1.exe
- Output Size: 130.650390625 KiB
- Compilation Time: 0.31s
```

Output:

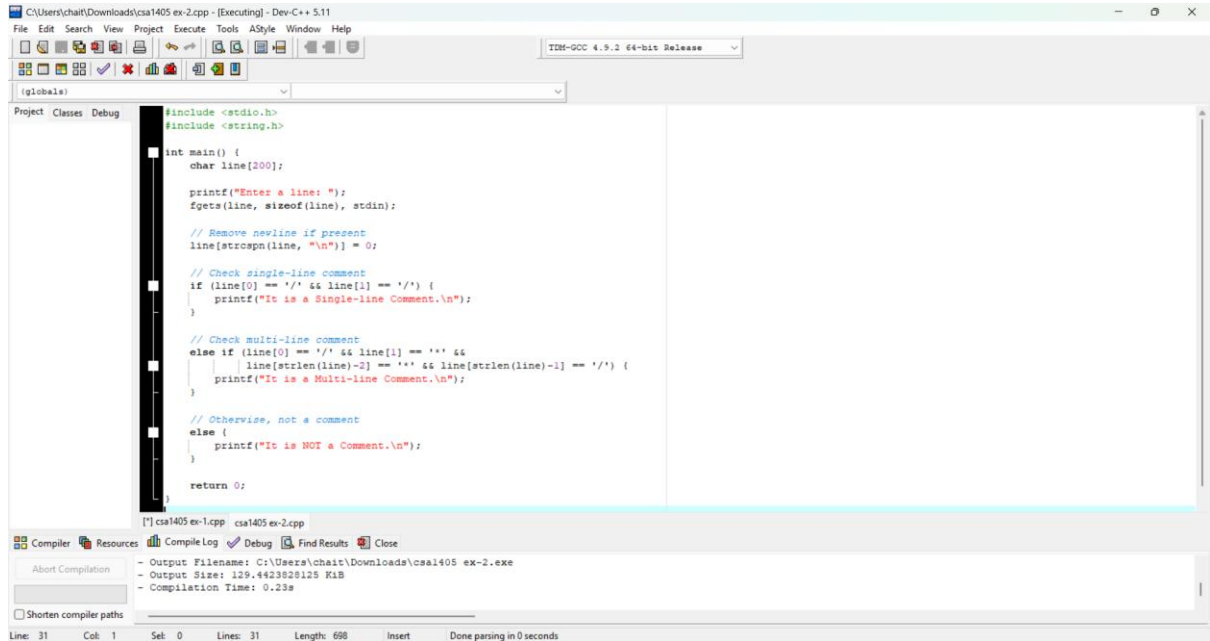


```
C:\Users\chait\Downloads\csa1405 ex-1>
Enter an expression: a = b + 45
Identifier : a
Operator : =
Identifier : b
Operator : +
Constant : 45
Special Symbol :

-----
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 with a C program open. The program is designed to check if a line of code is a single-line comment, a multi-line comment, or not a comment at all. It uses `fgets` to read a line and `strcspn` to find the first non-whitespace character. It then checks for `/*` and `*/` patterns.

```
#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;

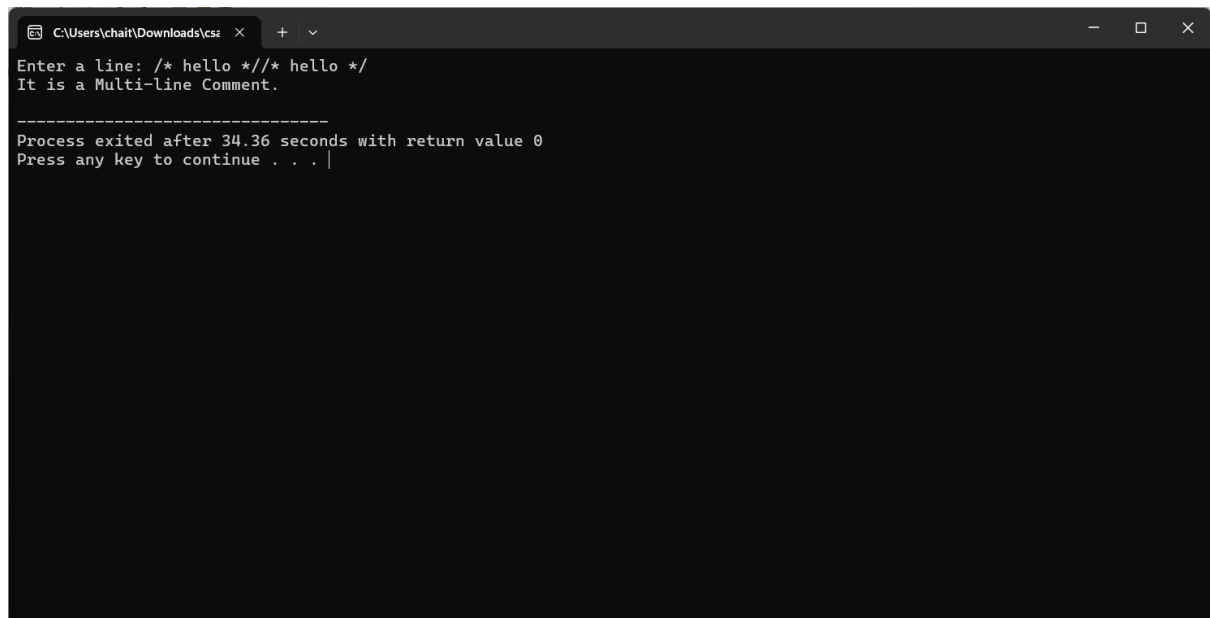
    // Check single-line comment
    if (line[0] == '/' && line[1] == '/') {
        printf("It is a Single-line Comment.\n");
    }

    // 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 bottom status bar shows the compiler output: "Output Filename: C:\Users\chait\Downloads\csa1405 ex-2.exe", "Output Size: 129.4423828125 KiB", and "Compilation Time: 0.23s".



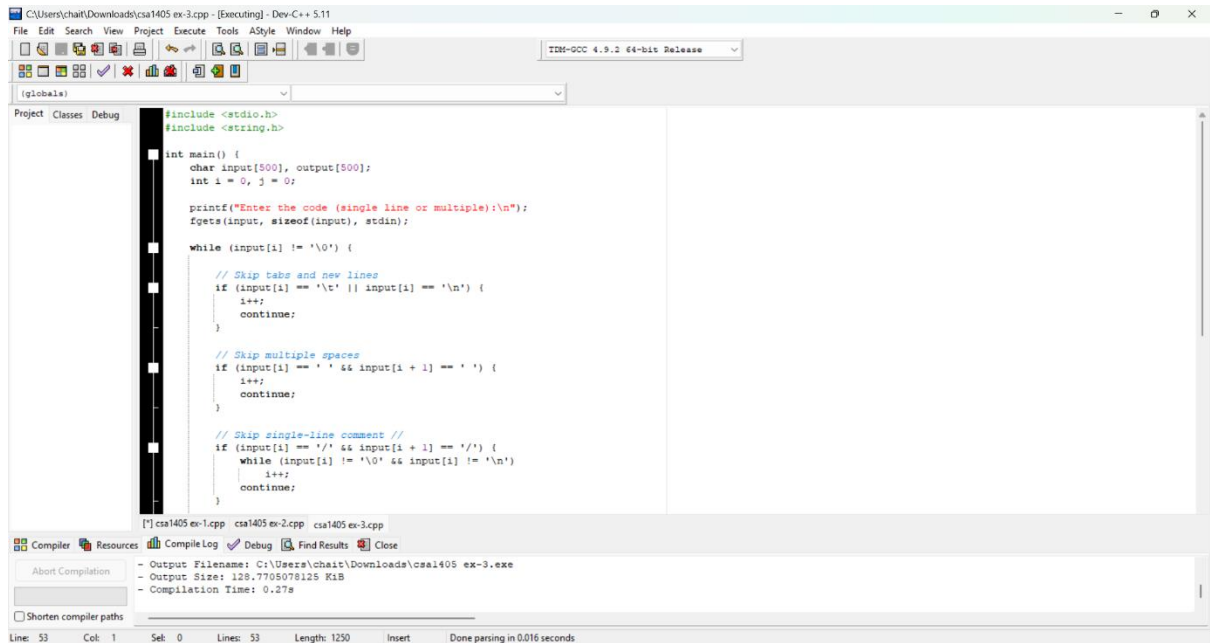
The screenshot shows a terminal window where the program has been executed. The user entered a multi-line comment, and the program correctly identified it as such. The terminal output is as follows:

```
C:\Users\chait\Downloads\csa1405 ex-2.exe
Enter a line: /* hello */
It is a Multi-line Comment.

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

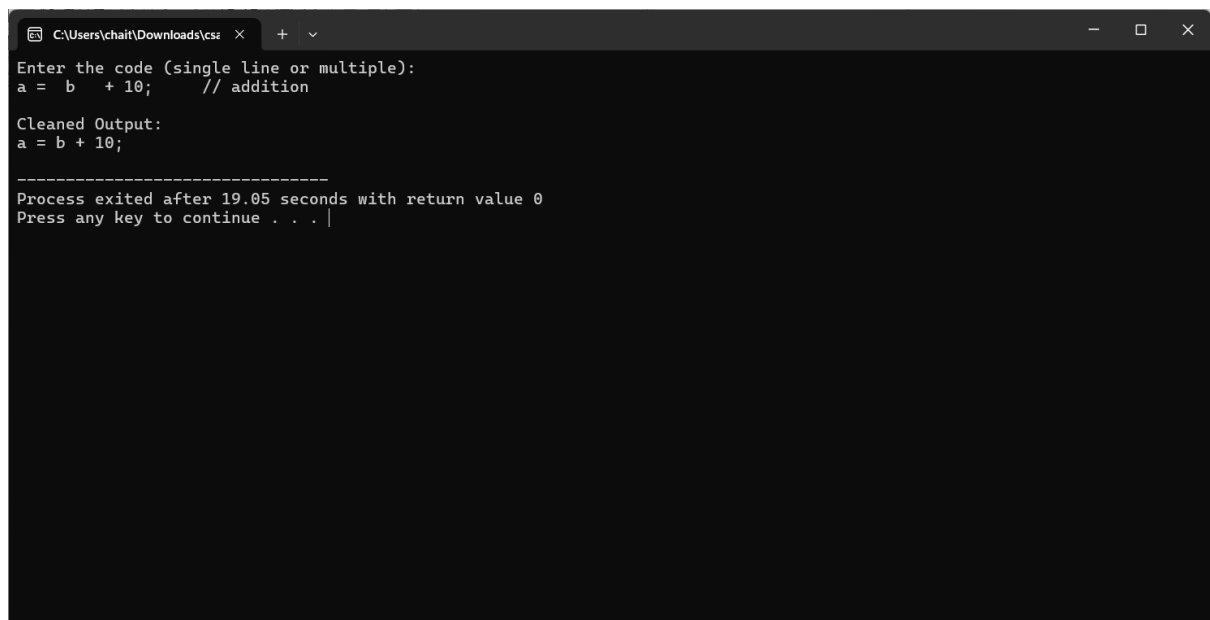
Exp. No. 3

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



The screenshot shows a C++ IDE with a project named 'globals'. The main code file is 'csa1405 ex-3.cpp'. The code implements a lexical analyzer that reads input from 'stdin' and processes it character by character. It uses a while loop to skip tabs, new lines, multiple spaces, and single-line comments. The output is stored in an array 'output' of size 500. The compiler is 'TDM-GCC 4.9.2 64-bit Release'. The output window shows the following compilation details:

```
Compiler: TDM-GCC 4.9.2 64-bit Release
Output Filename: C:\Users\chait\Downloads\csa1405 ex-3.exe
Output Size: 128.7705078125 KiB
Compilation Time: 0.27s
```



The screenshot shows a terminal window with the following output:

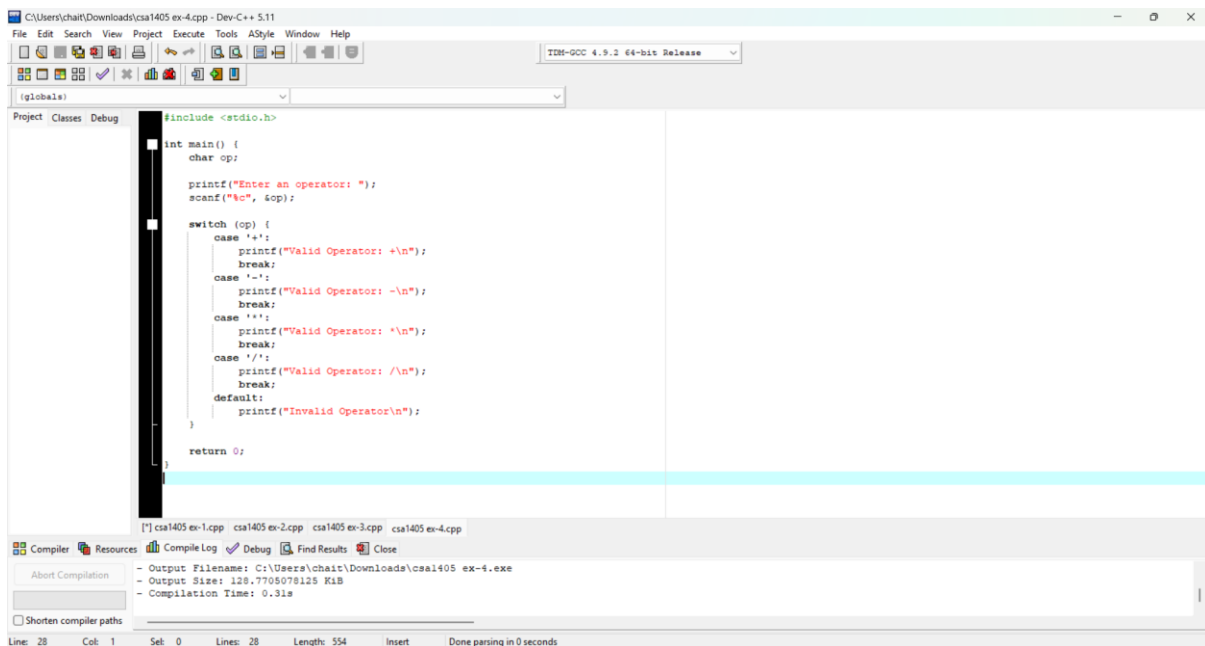
```
C:\Users\chait\Downloads\csa1405 ex-3.exe
Enter the code (single line or multiple):
a = b + 10; // addition

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



```
#include <stdio.h>

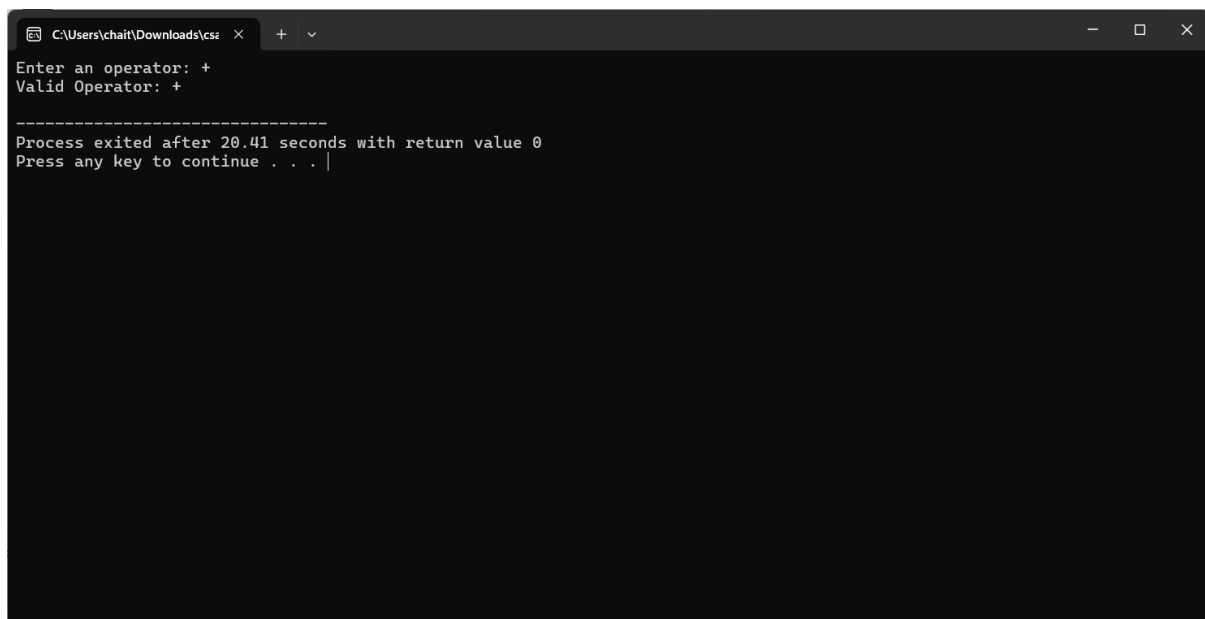
int main() {
    char op;

    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 C++ IDE with the source code for a lexical analyzer. The code prompts the user to enter an operator and then checks if it is a valid arithmetic operator (+, -, *, /). If the operator is valid, it prints a message; otherwise, it prints "Invalid Operator". The IDE also shows the compilation output, indicating that the program compiled successfully.



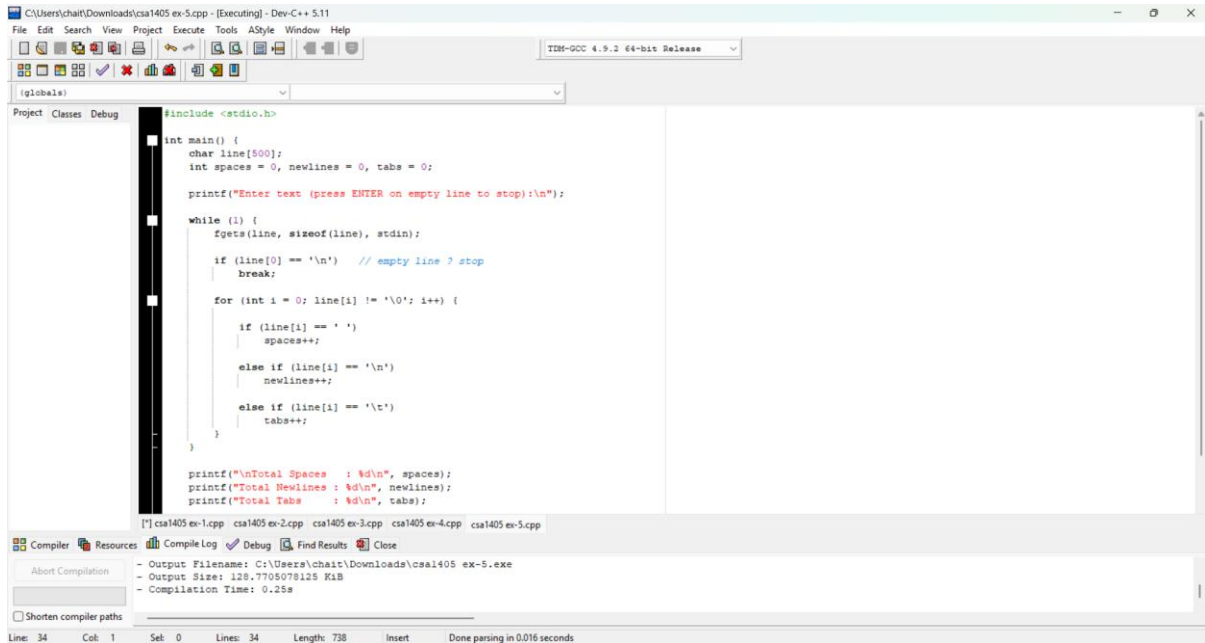
```
C:\Users\chait\Downloads\csa1405 ex-4.exe
Enter an operator: +
Valid Operator: +

-----
Process exited after 20.41 seconds with return value 0
Press any key to continue . . . |
```

The screenshot shows a terminal window where the program has been executed. The user entered the '+' operator, and the program correctly identified it as a valid operator. The terminal also shows the process exit message and a prompt to press any key to continue.

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 with a C program open. The program is designed to count the number of spaces, newlines, and tabs in a user-input string. The code is as follows:

```
#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");

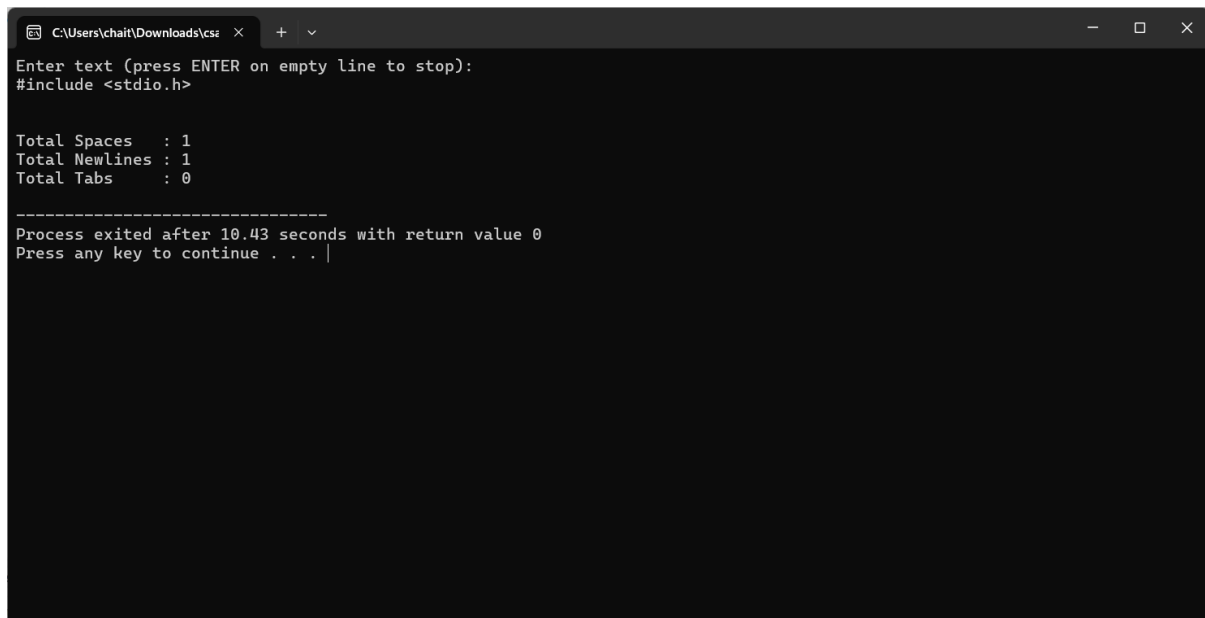
    while (1) {
        fgets(line, sizeof(line), stdin);

        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 compiler output at the bottom shows the program was compiled successfully into an executable file named 'csal405 ex-5.exe'.



The screenshot shows the output of the program in a terminal window. The user has entered the text 'include <stdio.h>' followed by an empty line to stop the input. The program has counted 1 space, 1 newline, and 0 tabs. The output is as follows:

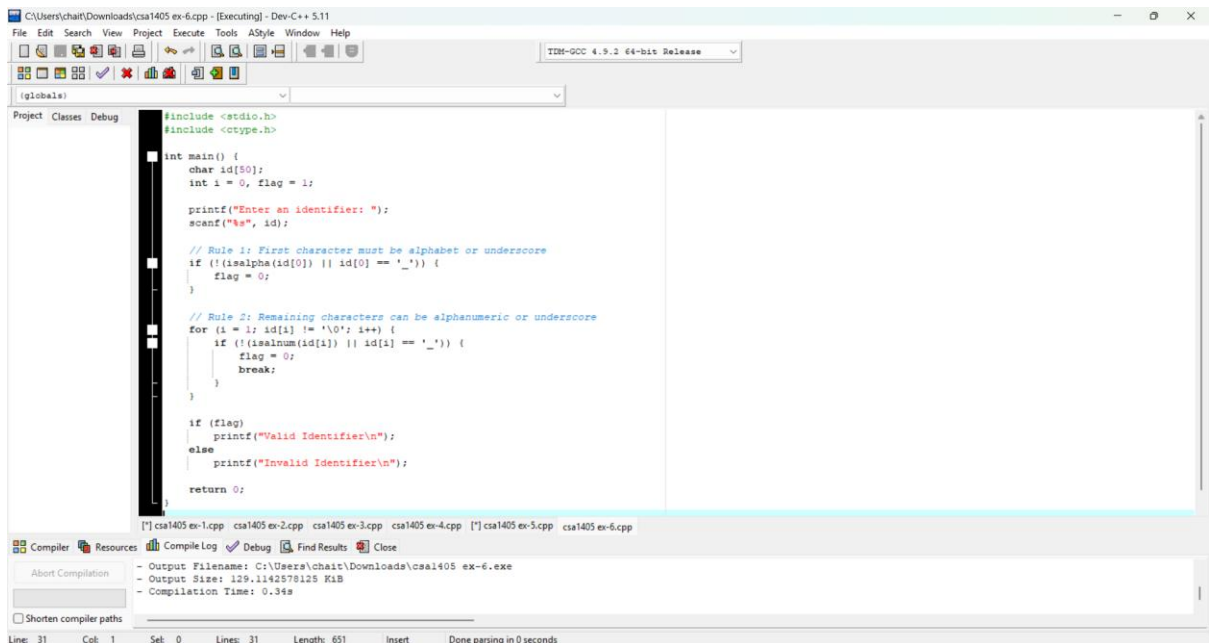
```
Enter text (press ENTER on empty line to stop):
#include <stdio.h>

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.



```
#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");

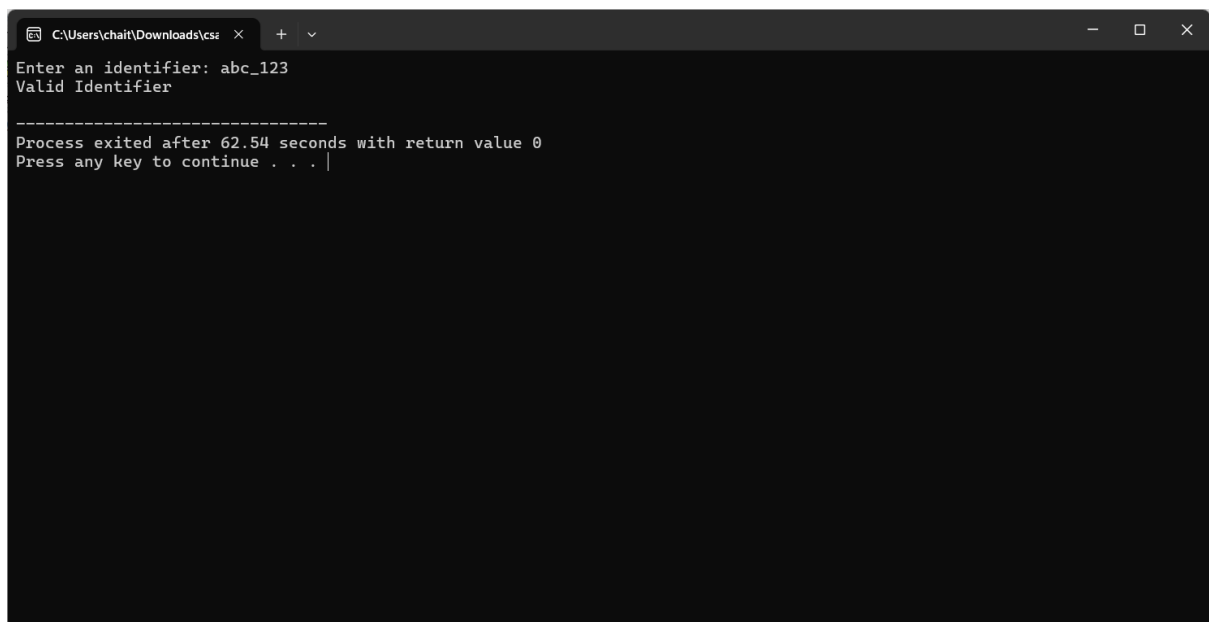
    return 0;
}
```

Compiler: TDM-GCC 4.9.2 64-bit Release

Output File: c:\Users\chait\Downloads\csa1405 ex-6.exe

Output Size: 129.1143578125 Kib

Compilation Time: 0.34s

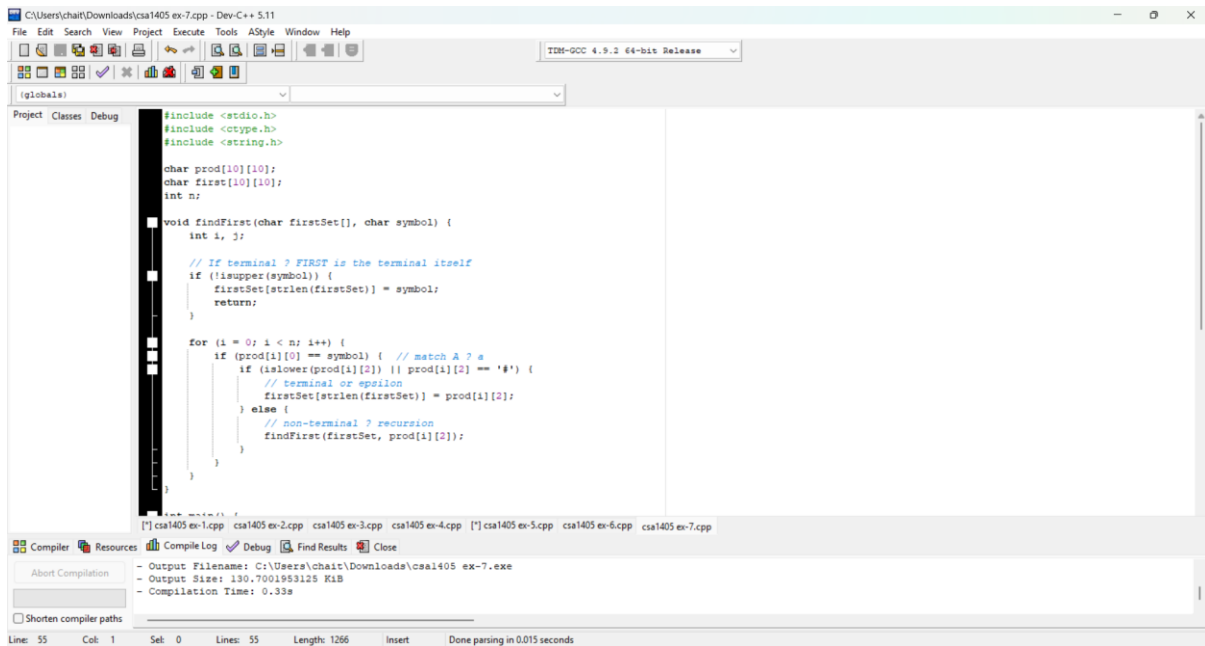


```
C:\Users\chait\Downloads\csa1405 ex-6.exe
Enter an identifier: abc_123
Valid Identifier

-----
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



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

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

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

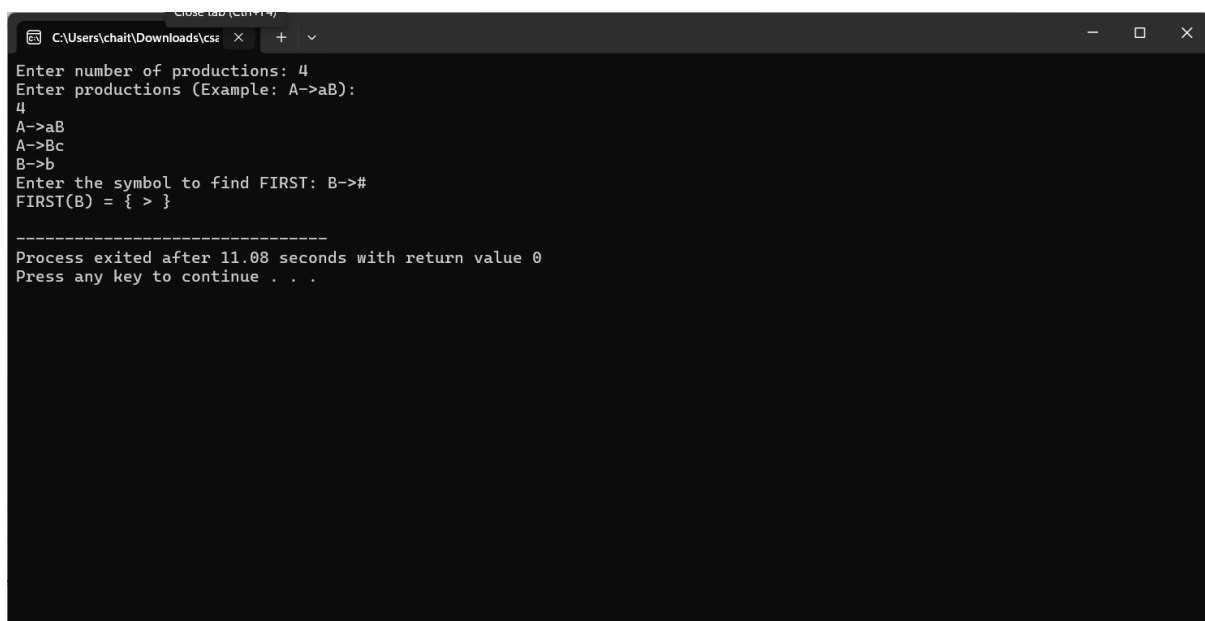
    // If terminal ? FIRST is the terminal itself
    if (!isupper(symbol)) {
        firstSet[strlen(firstSet)] = symbol;
        return;
    }

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

Compiler: TDM-GCC 4.9.2 64-bit Release

Output File Name: C:\Users\chait\Downloads\csa1405 ex-7.exe
Output Size: 130.7001953125 KiB
Compilation Time: 0.33s

Line: 55 Col: 1 Set: 0 Lines: 55 Length: 1266 Insert Done parsing in 0.015 seconds

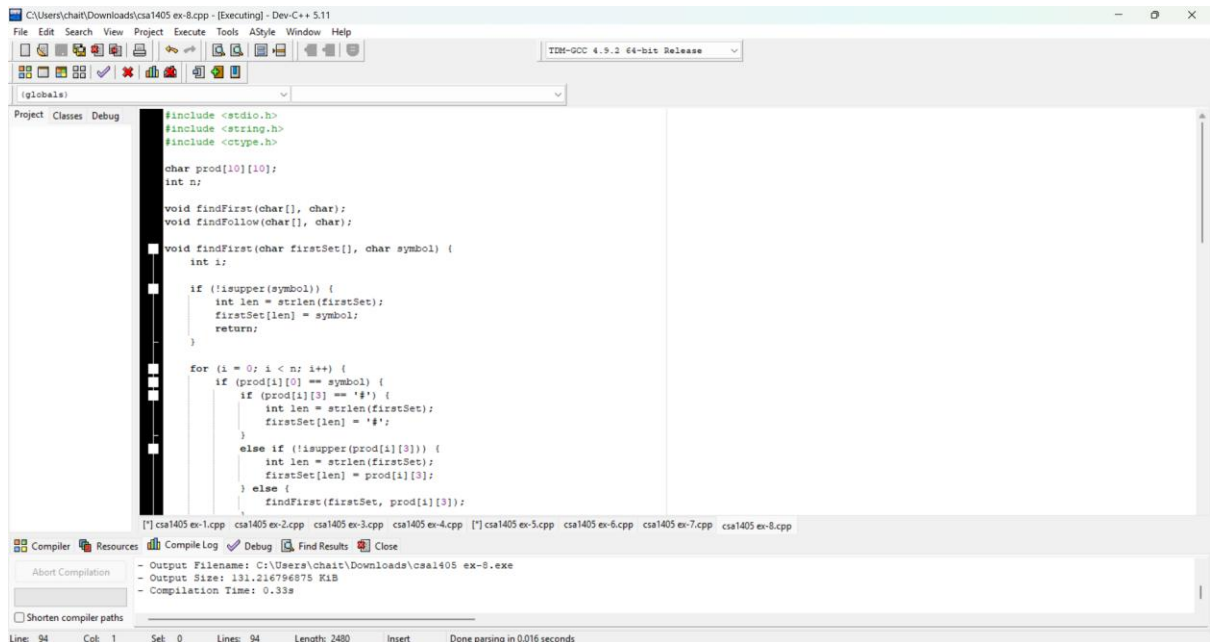


```
C:\Users\chait\Downloads\csa1405 ex-7.exe
Enter number of productions: 4
Enter productions (Example: A->aB):
4
A->aB
A->Bc
B->b
Enter the symbol to find FIRST: B->#
FIRST(B) = { > }

-----
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



```
#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]);
            }
        }
    }
}
```

Compiler: csa1405 ex-1.cpp csa1405 ex-2.cpp csa1405 ex-3.cpp csa1405 ex-4.cpp [*] csa1405 ex-5.cpp csa1405 ex-6.cpp csa1405 ex-7.cpp csa1405 ex-8.cpp

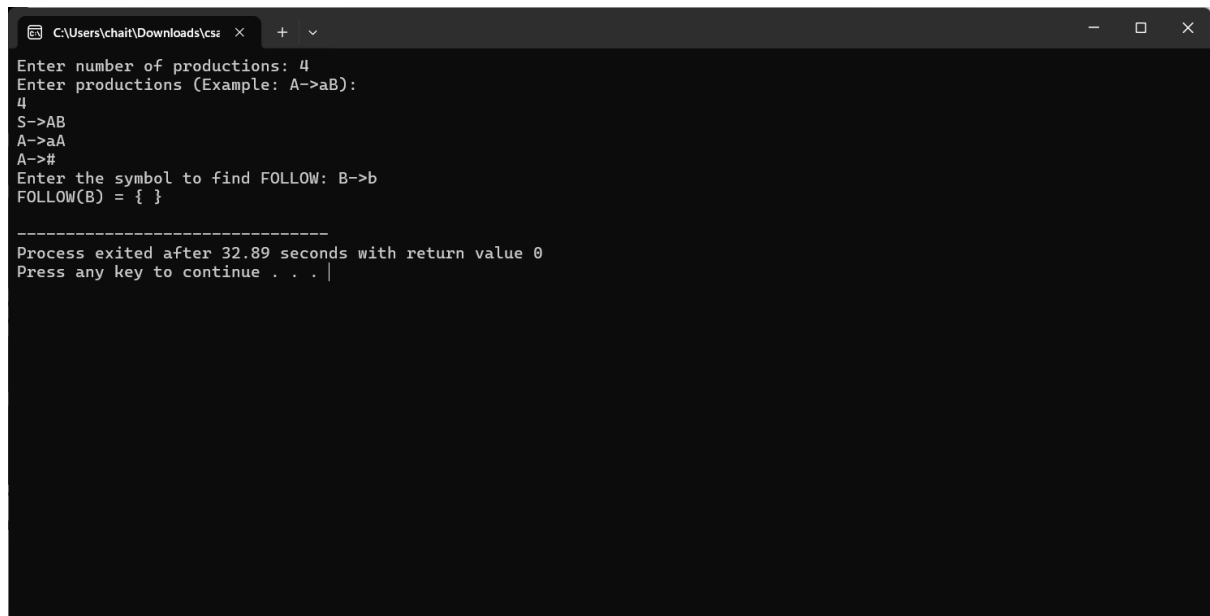
Compiler Resources: Compile Log Debug Find Results Close

Abort Compilation

- Output Filename: C:\Users\chait\Downloads\csa1405 ex-8.exe
- Output Size: 131.216796875 KiB
- Compilation Time: 0.33s

Shorten compiler paths

Line: 94 Col: 1 Set: 0 Lines: 94 Length: 2480 Insert Done parsing in 0.016 seconds

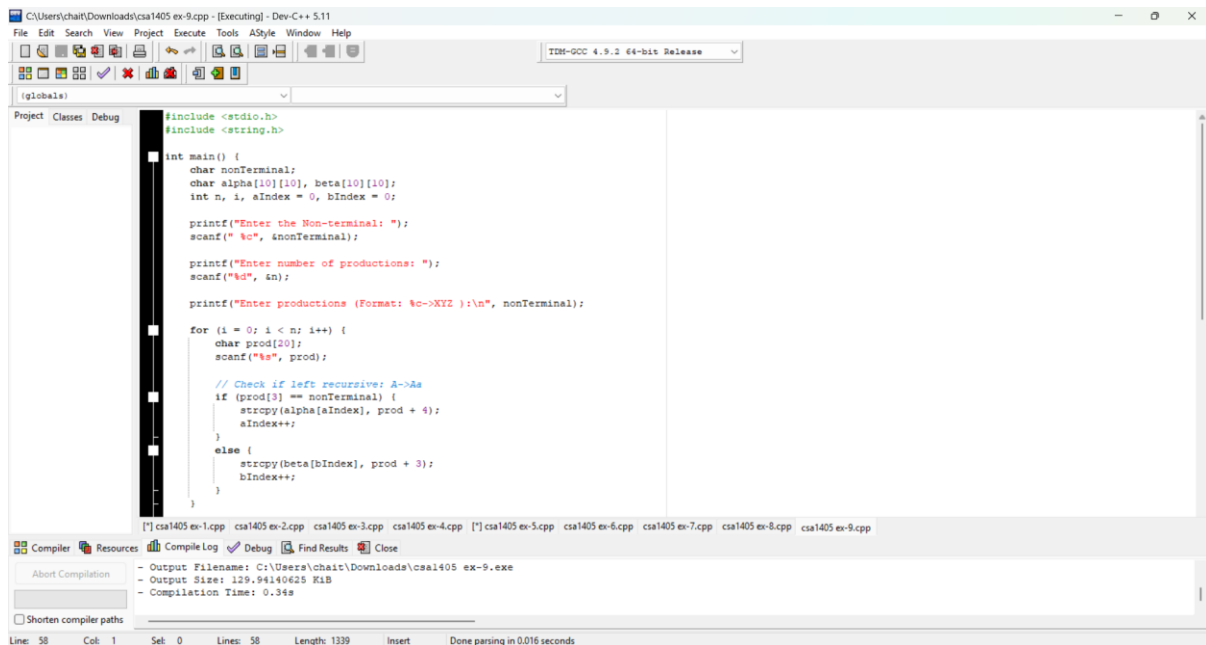


```
C:\Users\chait\Downloads\csa1405 ex-8.exe
Enter number of productions: 4
Enter productions (Example: A->aB):
4
S->AB
A->aA
A->#
Enter the symbol to find FOLLOW: B->b
FOLLOW(B) = { }

-----
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 with a C program open. The program prompts the user to enter a non-terminal, the number of productions, and the productions themselves. It then processes the productions to eliminate left recursion. The code is as follows:

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

int main() {
    char nonTerminal;
    char alpha[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);

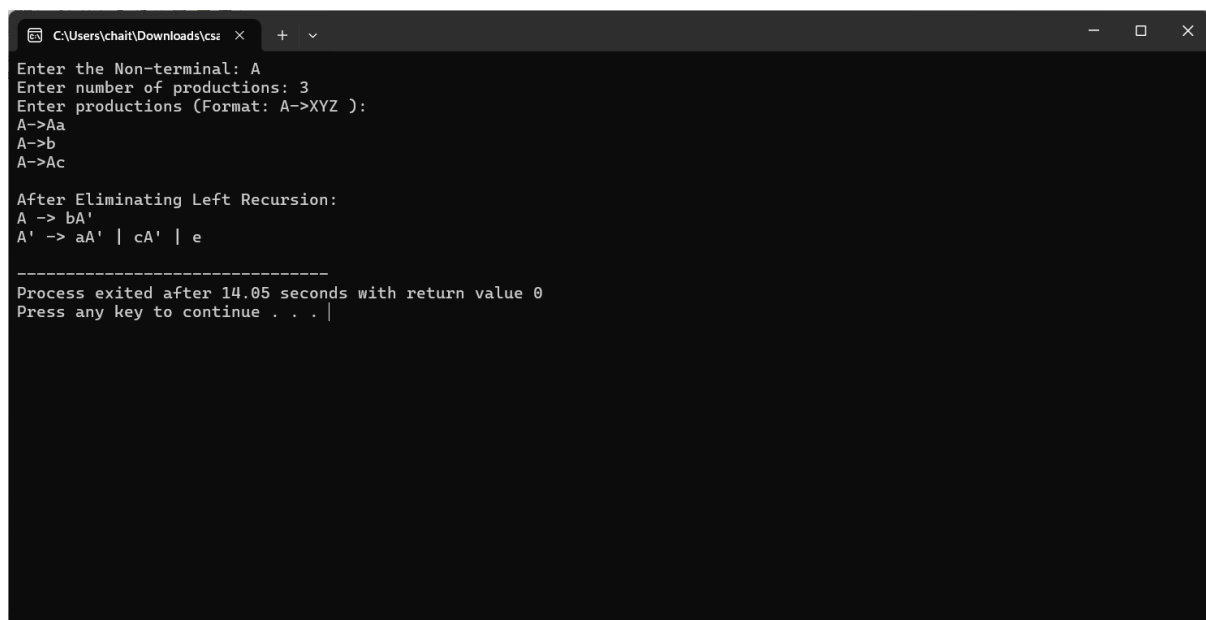
    printf("Enter productions (Format: %c->XYZ):\n", nonTerminal);

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

        // Check if left recursive: A->Aa
        if (prod[0] == nonTerminal) {
            strcpy(alpha[aIndex], prod + 1);
            aIndex++;
        }
        else {
            strcpy(beta[bIndex], prod + 1);
            bIndex++;
        }
    }
}
```

The IDE shows the compilation output at the bottom:

```
Compiler: gcc
Resources:
Compile Log:
Debug:
Find Results:
Close:
- Output Filename: C:\Users\chait\Downloads\csa1405 ex-9.exe
- Output Size: 129.94140625 KiB
- Compilation Time: 0.36s
```



The screenshot shows a terminal window with the following output:

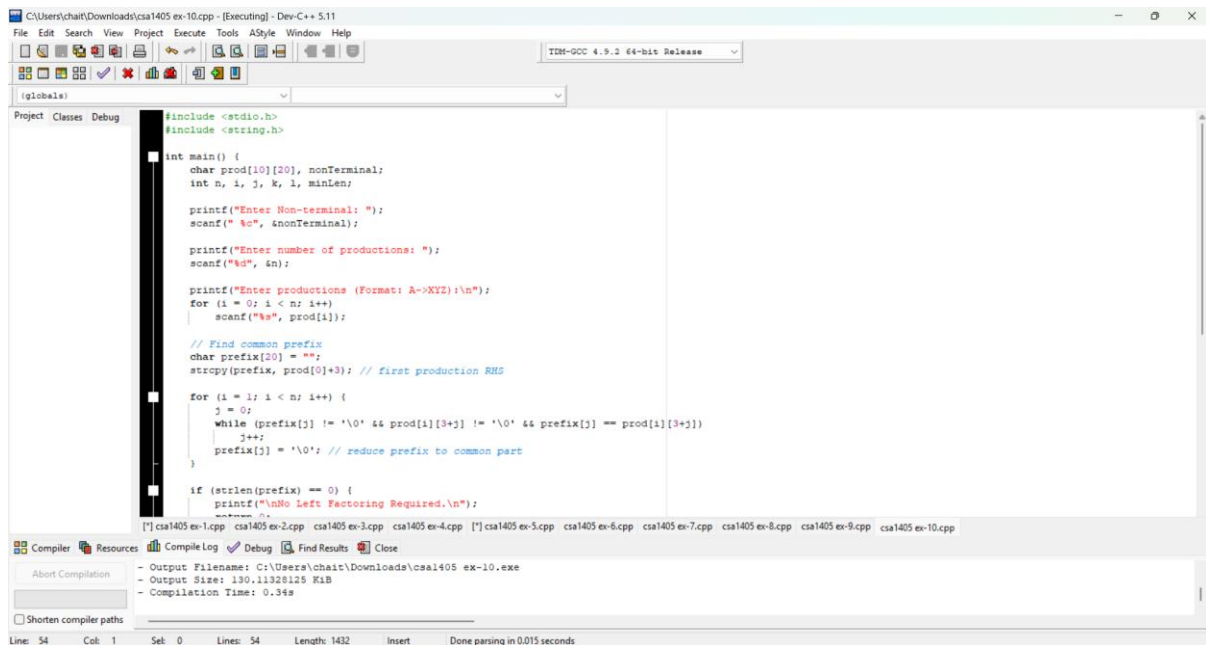
```
C:\Users\chait\Downloads\csa1405 ex-9.exe
Enter the Non-terminal: A
Enter number of productions: 3
Enter productions (Format: A->XYZ ):
A->Aa
A->b
A->Ac

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.



```
#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]); // first production RHS

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

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

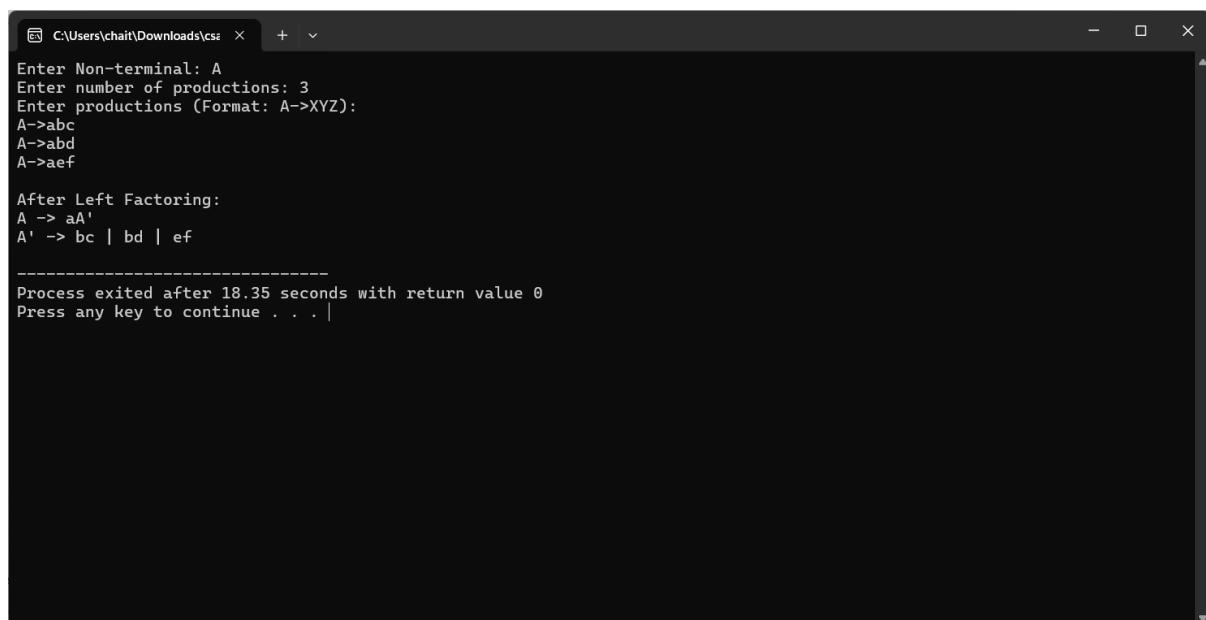
Compiler: Resources | Compile Log | Debug | Find Results | Close

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 Set: 0 Lines: 54 Length: 1432 Insert Done parsing in 0.015 seconds



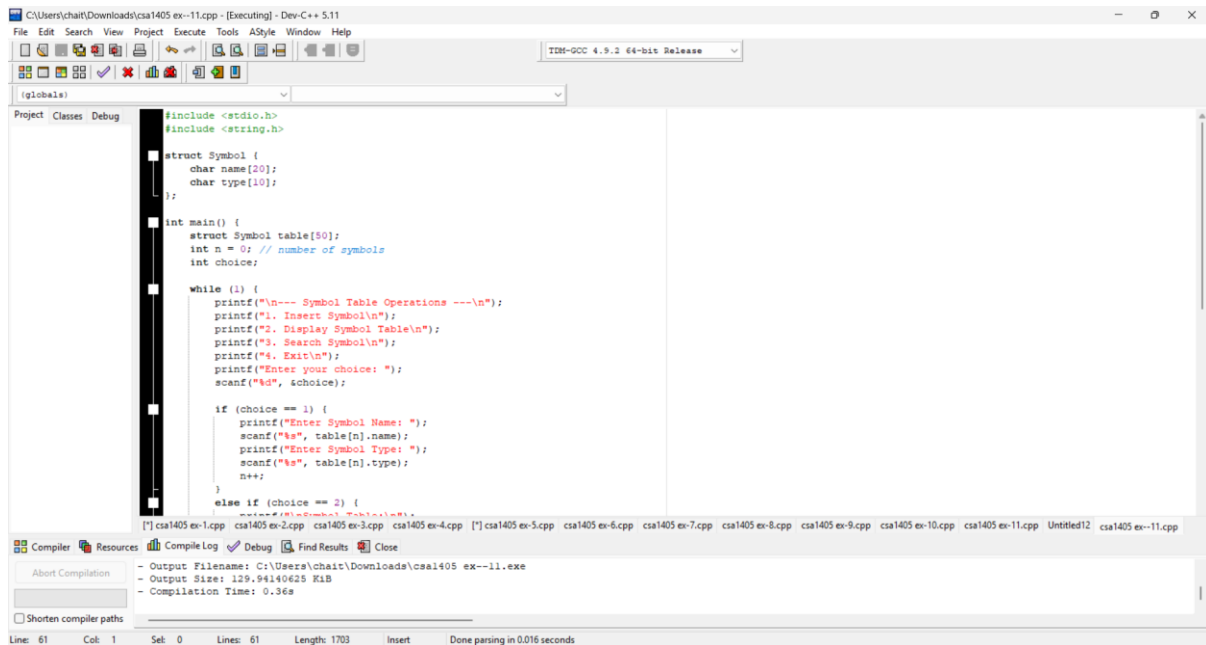
```
C:\Users\chait\Downloads\csa1405 ex-10.exe
Enter Non-terminal: A
Enter number of productions: 3
Enter productions (Format: A->XYZ):
A->abc
A->abd
A->aef

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 a C++ IDE with the following code:

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

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

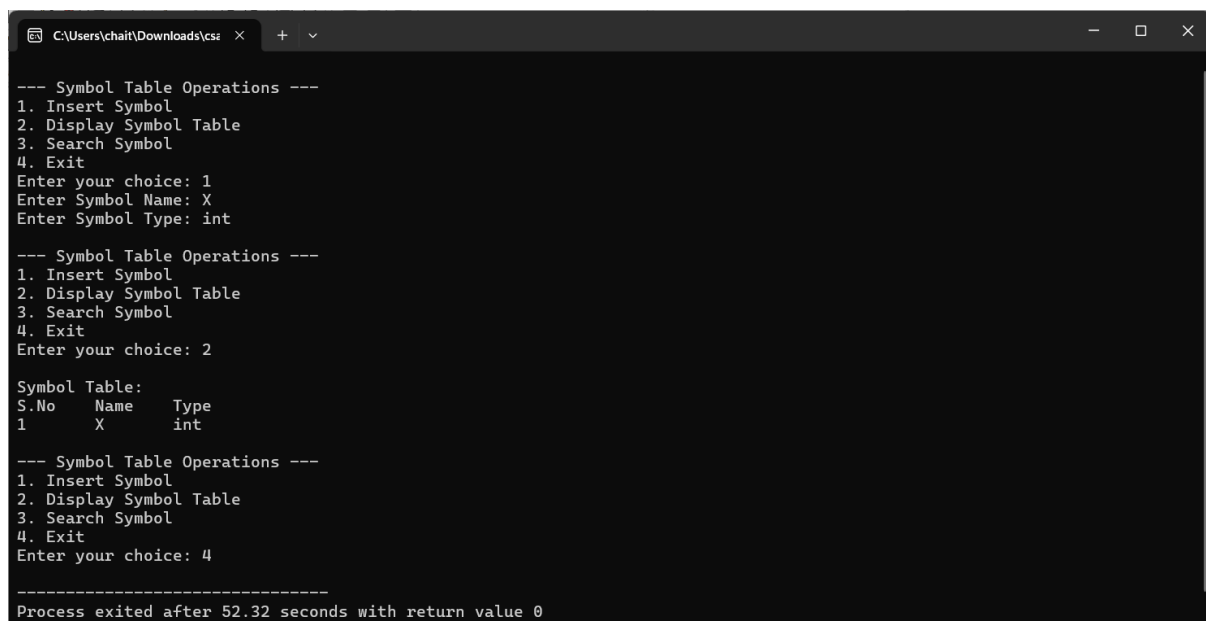
int main() {
    struct Symbol table[50];
    int n = 0; // number of symbols
    int choice;

    while (1) {
        printf("\n--- Symbol Table Operations ---\n");
        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("\nSymbol Table\n");
            for (int i = 0; i < n; i++) {
                printf("%d\t%s\t%s\n", i+1, table[i].name, table[i].type);
            }
        }
        else if (choice == 3) {
            printf("Enter Symbol Name to Search: ");
            char search_name[20];
            scanf("%s", search_name);
            for (int i = 0; i < n; i++) {
                if (strcmp(table[i].name, search_name) == 0) {
                    printf("Symbol found at index %d\n", i);
                    printf("Name: %s, Type: %s\n", table[i].name, table[i].type);
                }
            }
        }
        else if (choice == 4) {
            break;
        }
    }
}
```

The IDE also shows the compiler output at the bottom:

```
Compiler: g++
Resources:
- Output Filename: C:\Users\chait\Downloads\csa1405 ex--11.exe
- Output Size: 129.94140625 KiB
- Compilation Time: 0.36s
```



The screenshot shows the execution of the program in a terminal window. The output is as follows:

```
--- 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

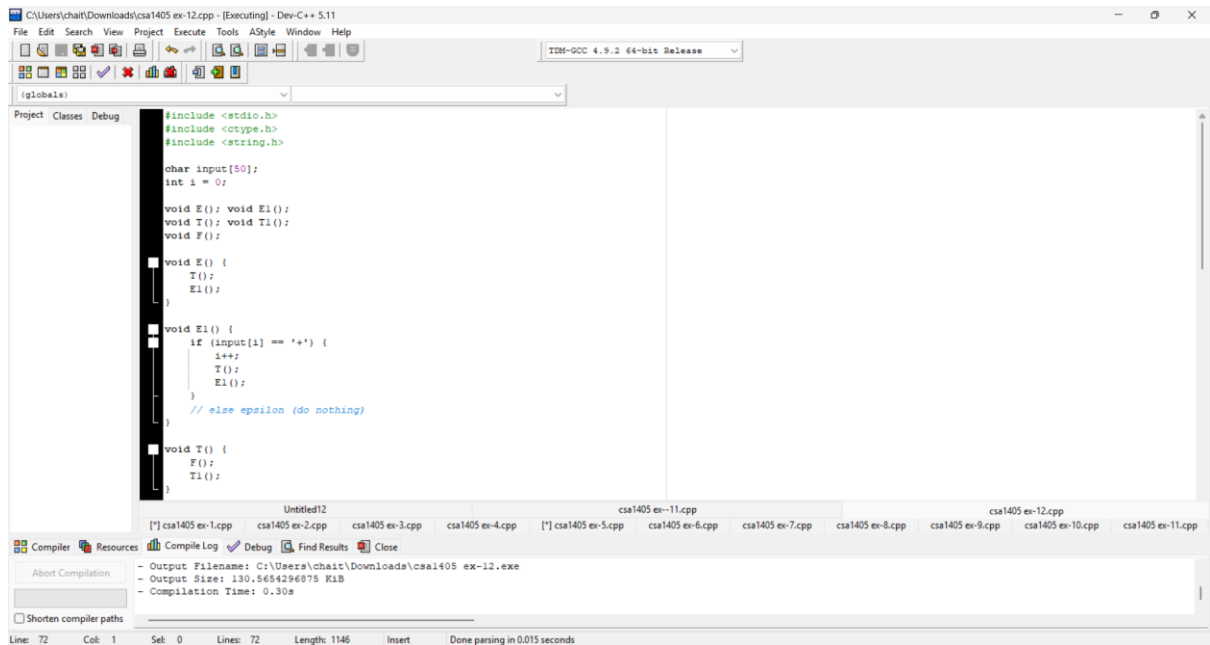
Symbol Table:
S.No  Name  Type
1     X    int

--- 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 a C++ IDE with the following code in the main editor:

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

char input[50];
int i = 0;

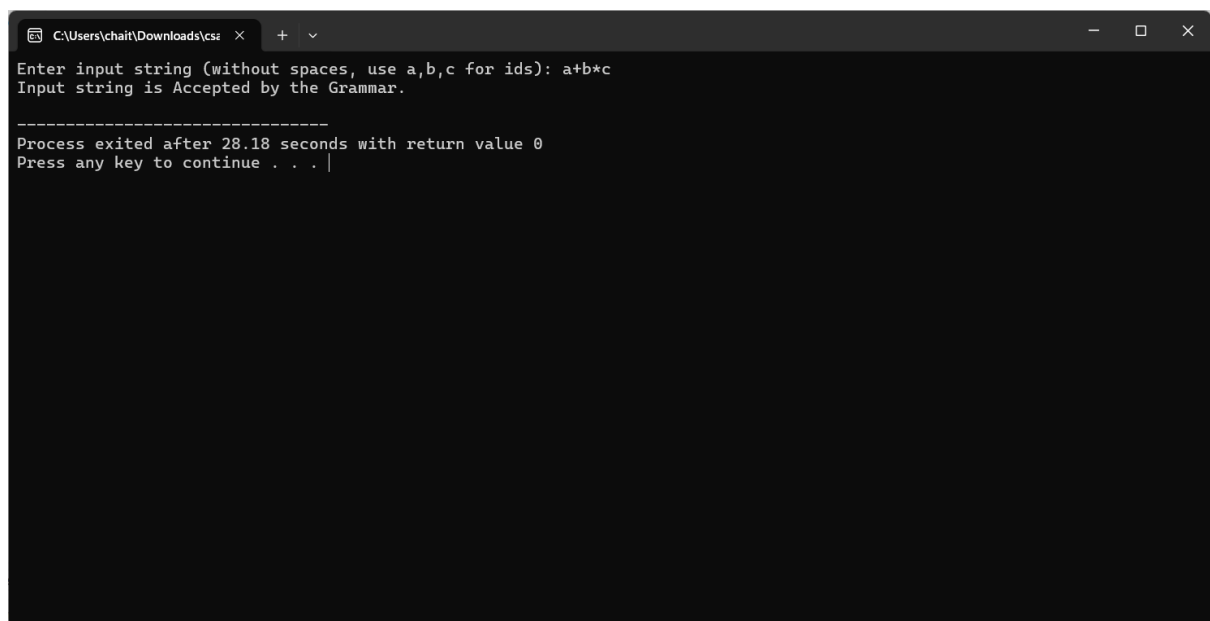
void E(); void E1();
void T(); void T1();
void F();

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

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

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

The IDE's status bar at the bottom indicates: Line: 72, Col: 1, Sel: 0, Lines: 72, Length: 1146, Insert, Done parsing in 0.015 seconds.



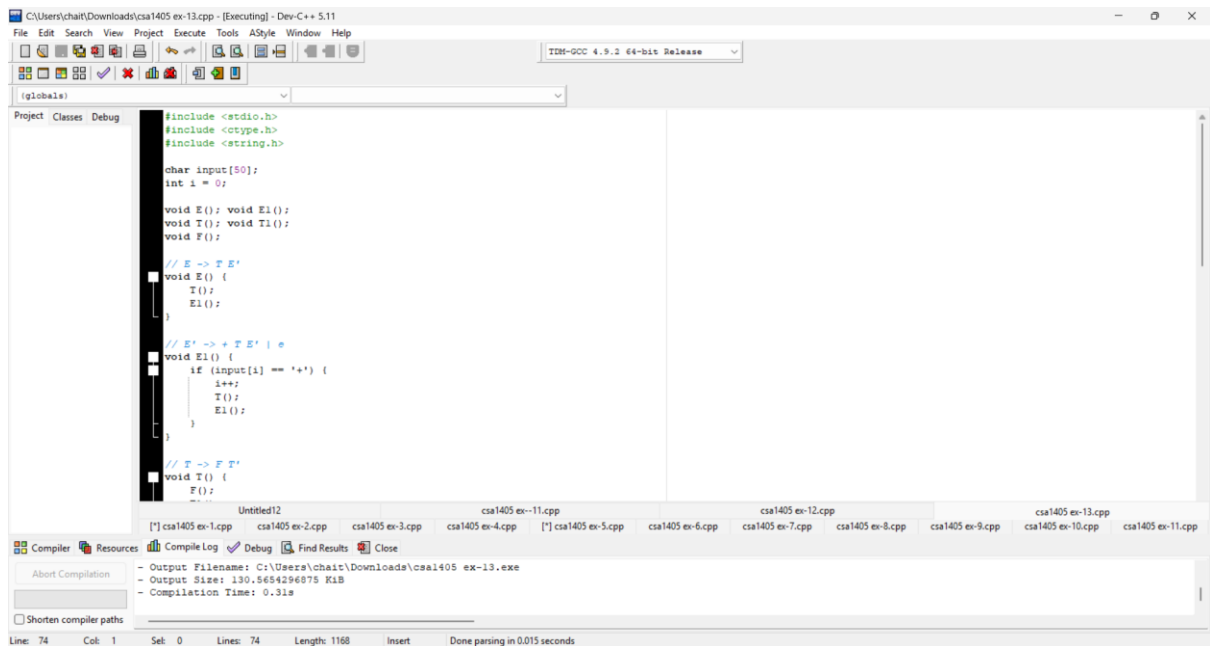
The screenshot shows a terminal window with the following output:

```
C:\Users\chait\Downloads\csa1405 ex-12.exe
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 a C++ IDE with the following code in the main editor:

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

char input[50];
int i = 0;

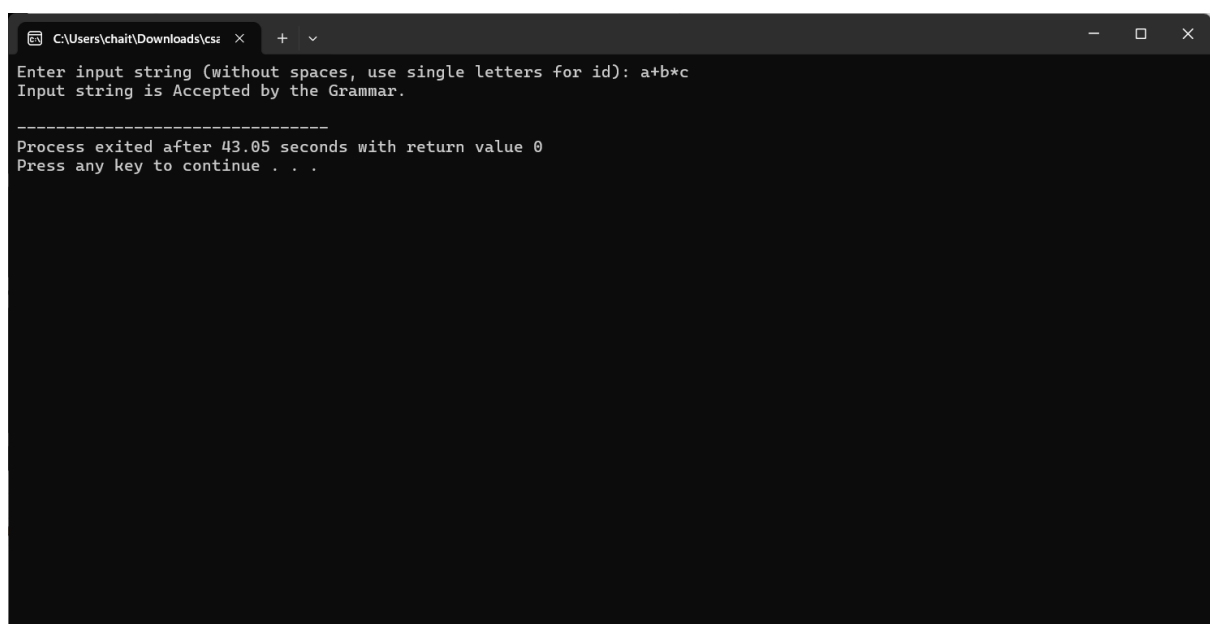
void E(); void E1();
void T(); void T1();
void F();

// 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();
    T1();
}
```

The IDE's status bar at the bottom indicates: Line: 74, Col: 1, Set: 0, Lines: 74, Length: 1168, Insert, Done parsing in 0.015 seconds.



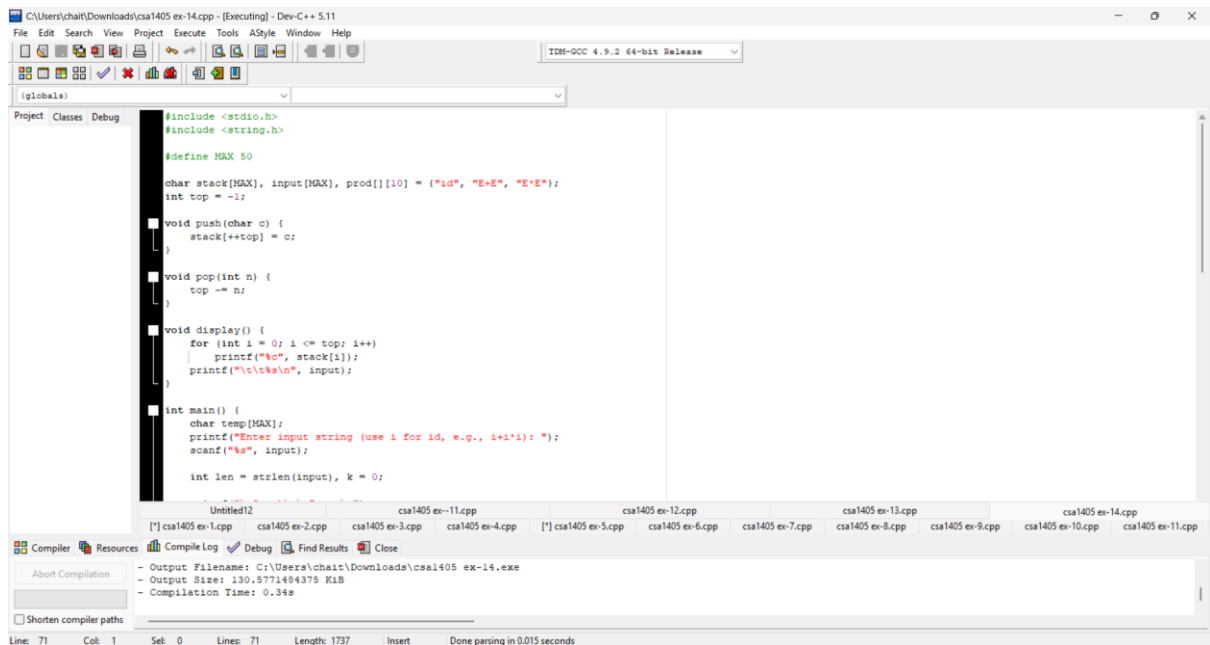
The terminal window shows the following output:

```
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.



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

#define MAX 50

char stack[MAX], input[MAX], prod[10] = {"id", "E+E", "E-E"};
int top = -1;

void push(char c) {
    stack[++top] = c;
}

void pop(int n) {
    top -= n;
}

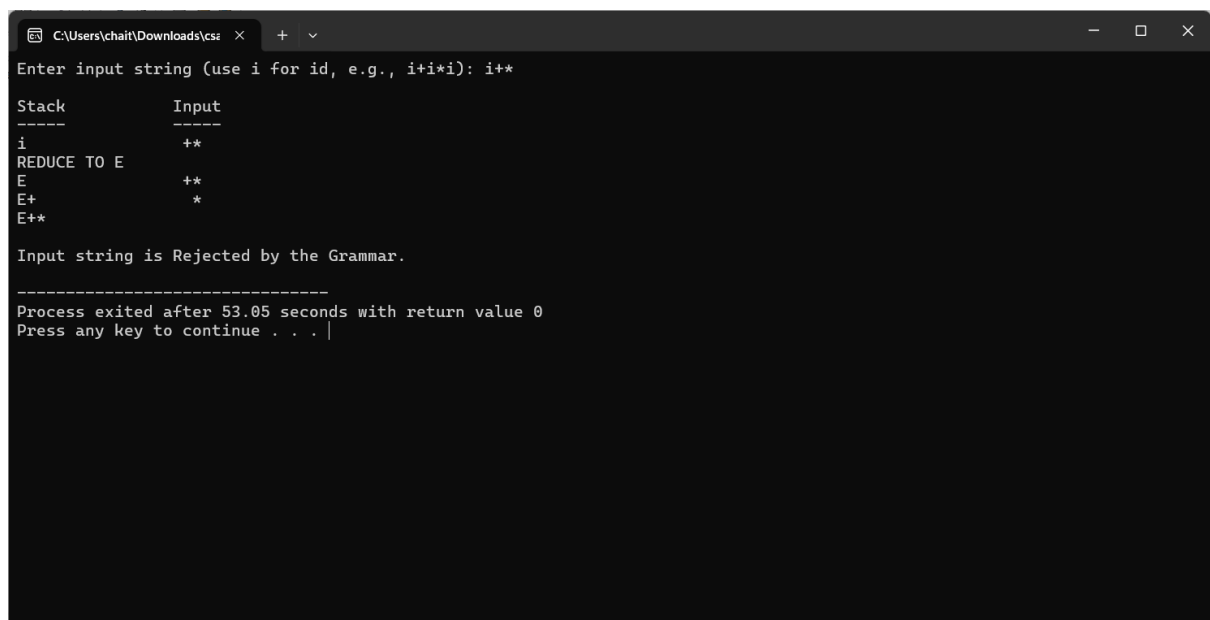
void display() {
    for (int i = 0; i <= top; i++)
        printf("%c", stack[i]);
    printf("\n", input);
}

int main() {
    char temp[MAX];
    printf("Enter input string (use i for id, e.g., i+i*i): ");
    scanf("%s", input);

    int len = strlen(input), k = 0;
```

The screenshot shows a C++ IDE with the following components:

- Editor:** Contains the C++ code for the shift-reduce parser.
- Compiler:** Shows the compilation output: "Output Filename: C:\Users\chait\Downloads\csl1405 ex-14.exe", "Output Size: 130.5771484375 KiB", and "Compilation Time: 0.34s".
- Debugger:** Shows the execution status: "Line: 71 Col: 1 Set: 0 Lines: 71 Length: 1737 Insert Done parsing in 0.015 seconds".



```
Enter input string (use i for id, e.g., i+i*i): i++

Stack      Input
-----
i          ++
REDUCE TO E
E          ++
E+         *
E+         *
```

Input string is Rejected by the Grammar.

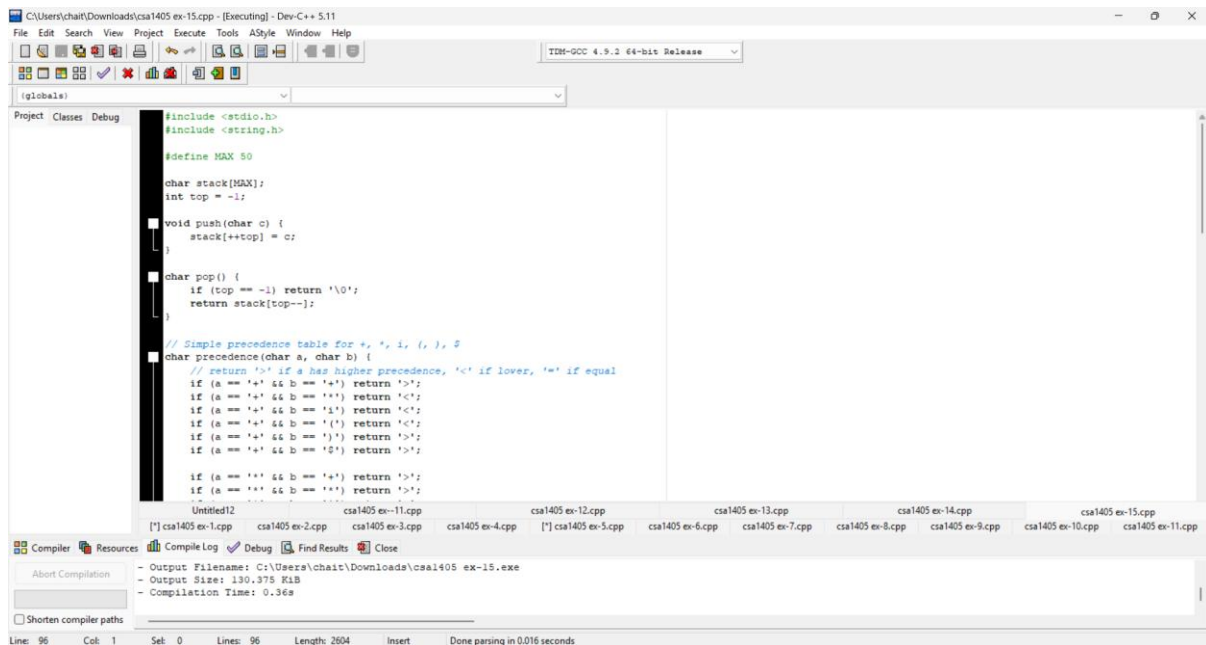
Process exited after 53.05 seconds with return value 0
Press any key to continue . . . |

The screenshot shows a terminal window with the following output:

- The user enters the input string "i++".
- The program displays the stack and input at each step of the shift-reduce process.
- The input string is rejected by the grammar.
- The process exits after 53.05 seconds with a return value of 0.

Exp. No. 15

Write a C Program to implement the operator precedence parsing.



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

#define MAX 50

char stack[MAX];
int top = -1;

void push(char c) {
    stack[++top] = c;
}

char pop() {
    if (top == -1) return '\0';
    return stack[top--];
}

// Simple precedence table for +, *, i, (, ), $
char precedence(char a, char b) {
    // return '>' if a has higher precedence, '<' if lower, '=' if equal
    if (a == '+' && b == '+') return '>';
    if (a == '+' && b == '*') return '<';
    if (a == '+' && b == 'i') return '<';
    if (a == '+' && b == '(') return '<';
    if (a == '+' && b == ')') return '>';
    if (a == '+' && b == '$') return '>';

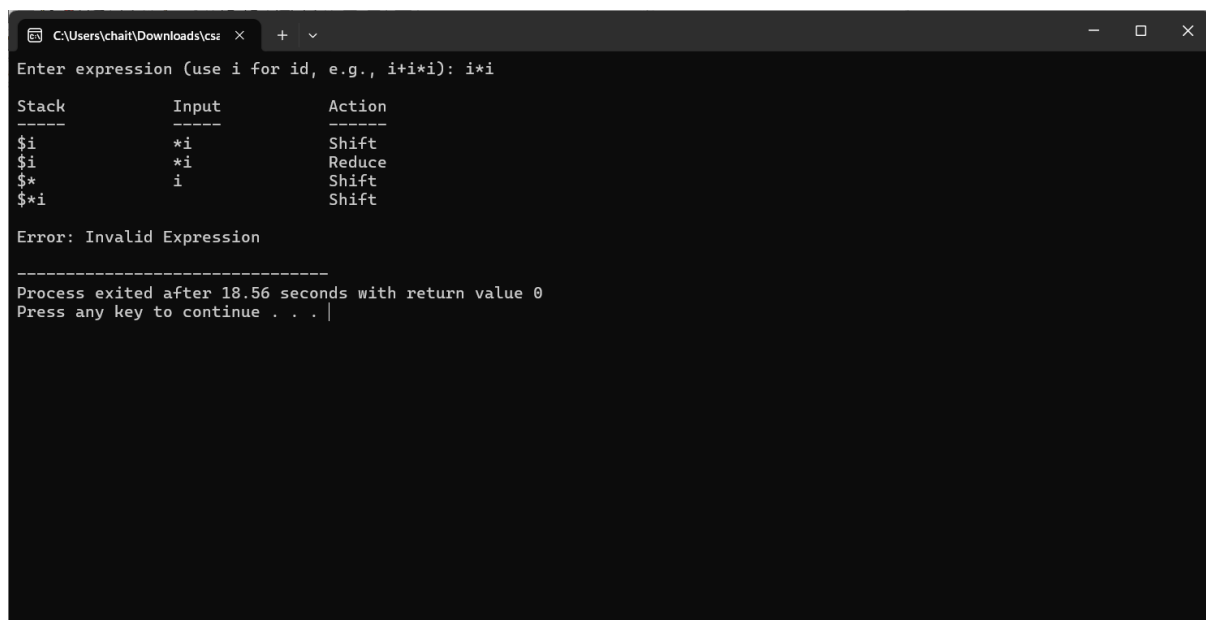
    if (a == '*' && b == '+') return '>';
    if (a == '*' && b == '*') return '>';
    if (a == '*' && b == 'i') return '>';
    if (a == '*' && b == '(') return '<';
    if (a == '*' && b == ')') return '>';
    if (a == '*' && b == '$') return '>';

    if (a == 'i' && b == '+') return '<';
    if (a == 'i' && b == '*') return '<';
    if (a == 'i' && b == 'i') return '>';
    if (a == 'i' && b == '(') return '<';
    if (a == 'i' && b == ')') return '>';
    if (a == 'i' && b == '$') return '>';

    if (a == '(' && b == '+') return '<';
    if (a == '(' && b == '*') return '<';
    if (a == '(' && b == 'i') return '<';
    if (a == '(' && b == '(') return '<';
    if (a == '(' && b == ')') return '>';
    if (a == '(' && b == '$') return '>';

    if (a == ')' && b == '+') return '>';
    if (a == ')' && b == '*') return '>';
    if (a == ')' && b == 'i') return '>';
    if (a == ')' && b == '(') return '<';
    if (a == ')' && b == ')') return '>';
    if (a == ')' && b == '$') return '>';

    if (a == '$' && b == '+') return '>';
    if (a == '$' && b == '*') return '>';
    if (a == '$' && b == 'i') return '>';
    if (a == '$' && b == '(') return '>';
    if (a == '$' && b == ')') return '>';
    if (a == '$' && b == '$') return '>';
}
```



```
C:\Users\chait\Downloads\csa1405-ex-15.exe
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*i        $         Shift

Error: Invalid Expression

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