



**SIMATS SCHOOL OF ENGINEERING**  
**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES**  
**CHENNAI-602105**



## **LAB PROGRAMS (11-20)**

**ON**

**CSA1445-COMPILER DESIGN FOR POLYMORPHIC  
FUNCTIONS**

**SLOT C**

**Submitted by**

**192321072– P.PANEENDRA**

**To**

**DR.ANITHA G**

**Saveetha School of Engineering**  
**SIMATS, Thandalam.**

## 11)Implement a C program to perform symbol table operations.

### Aim:

To implement a symbol table in C that supports insertion, search, and display operations.

### Code:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define SIZE 100
struct Symbol {
    char name[50];
    char type[20];
    int address;
} table[SIZE];
int count = 0;
void insert(char name[], char type[], int address) {
    strcpy(table[count].name, name);
    strcpy(table[count].type, type);
    table[count].address = address;
    count++;
}
int search(char name[]) {
    for (int i = 0; i < count; i++) {
        if (strcmp(table[i].name, name) == 0)
            return i;
    }
    return -1;
}
```

```

void display() {
    printf("\nSymbol Table:\n");
    printf("Name\tType\tAddress\n");
    for (int i = 0; i < count; i++) {
        printf("%s\t%s\t%d\n", table[i].name, table[i].type, table[i].address);
    }
}

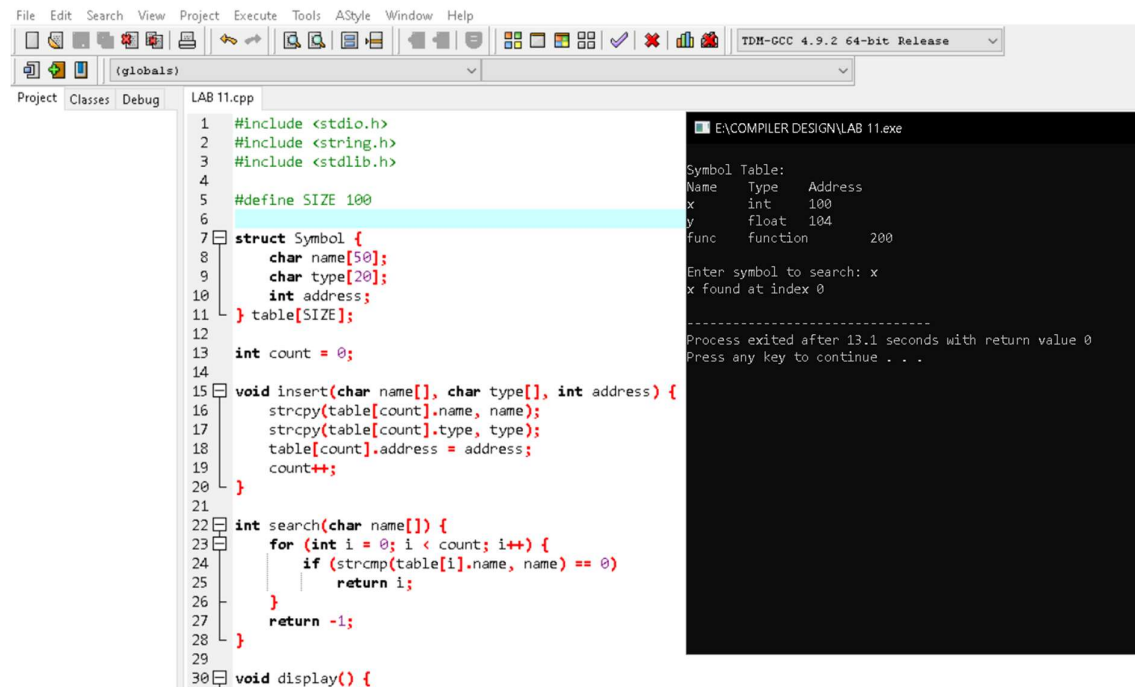
int main() {
    insert("x", "int", 100);
    insert("y", "float", 104);
    insert("func", "function", 200);
    display();
    char searchName[50];
    printf("\nEnter symbol to search: ");
    scanf("%s", searchName);

    int pos = search(searchName);
    if (pos != -1)
        printf("%s found at index %d\n", searchName, pos);
    else
        printf("%s not found in symbol table.\n", searchName);

    return 0;
}

```

## Output:



```
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug LAB 11.cpp
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 #define SIZE 100
6
7 struct Symbol {
8     char name[50];
9     char type[20];
10    int address;
11 } table[SIZE];
12
13 int count = 0;
14
15 void insert(char name[], char type[], int address) {
16     strcpy(table[count].name, name);
17     strcpy(table[count].type, type);
18     table[count].address = address;
19     count++;
20 }
21
22 int search(char name[]) {
23     for (int i = 0; i < count; i++) {
24         if (strcmp(table[i].name, name) == 0)
25             return i;
26     }
27     return -1;
28 }
29
30 void display() {
```

```
E:\COMPILER DESIGN\LAB 11.exe
Symbol Table:
Name Type Address
x int 100
y float 104
func function 200
Enter symbol to search: x
x found at index 0
-----
Process exited after 13.1 seconds with return value 0
Press any key to continue . . .
```

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

### Aim:

To implement a recursive descent parser for the given grammar.

### Code:

```
#include <stdio.h>
#include <string.h>
char input[100];
int index = 0;
void E(), T(), F(), E_prime(), T_prime();
void match(char expected) {
    if (input[index] == expected) {
        index++;
    }
}
```

```
    } else {  
        printf("Error in parsing\n");  
        exit(0);  
    }  
}  
void E() {  
    T();  
    E_prime();  
}  
void E_prime() {  
    if (input[index] == '+') {  
        match('+');  
        T();  
        E_prime();  
    }  
}  
void T() {  
    F();  
    T_prime();  
}  
void T_prime() {  
    if (input[index] == '*') {  
        match('*');  
        F();  
        T_prime();  
    }  
}
```

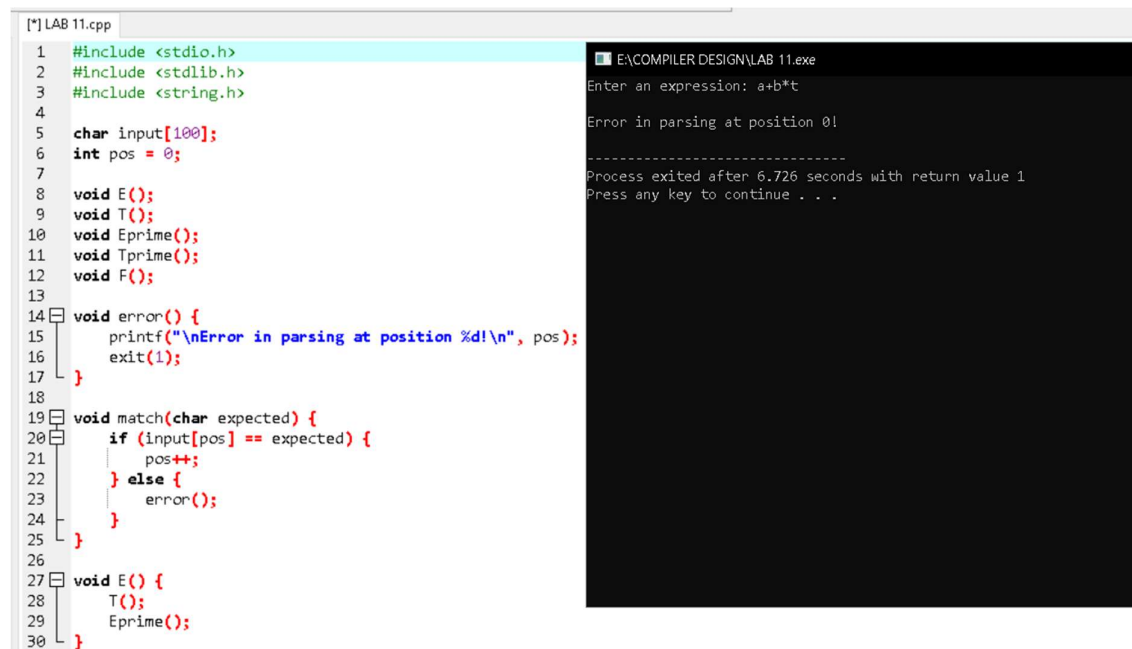
```

void F() {
    if (input[index] == '(') {
        match('(');
        E();
        match(')');
    } else if (input[index] == 'i') {
        match('i'); // Assuming 'i' represents an identifier
    } else {
        printf("Error in parsing\n");
        exit(0);
    }
}

int main() {
    printf("Enter input string: ");
    scanf("%s", input);
    E();
    if (input[index] == '\0') {
        printf("Parsing successful\n");
    } else {
        printf("Error: Unexpected input\n");
    }
    return 0;
}

```

## Output:



The screenshot shows a C program in a text editor on the left and its execution output in a terminal window on the right. The C program, named LAB 11.cpp, includes `<stdio.h>`, `<stdlib.h>`, and `<string.h>`. It defines a character array `input` of size 100 and an integer `pos` initialized to 0. It declares several functions: `E()`, `T()`, `Eprime()`, `Tprime()`, `F()`, `error()`, and `match()`. The `error()` function prints an error message and exits with a return value of 1. The `match()` function checks if the current character in `input` matches the expected character. The `E()` function calls `T()` and `Eprime()`. The terminal window shows the program being executed as `E:\COMPILER DESIGN\LAB 11.exe`. It prompts the user to "Enter an expression: a+b\*t". The program then outputs "Error in parsing at position 0!". A separator line follows, and the terminal reports "Process exited after 6.726 seconds with return value 1" and "Press any key to continue . . .".

```
[*] LAB 11.cpp
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  char input[100];
6  int pos = 0;
7
8  void E();
9  void T();
10 void Eprime();
11 void Tprime();
12 void F();
13
14 void error() {
15     printf("\nError in parsing at position %d!\n", pos);
16     exit(1);
17 }
18
19 void match(char expected) {
20     if (input[pos] == expected) {
21         pos++;
22     } else {
23         error();
24     }
25 }
26
27 void E() {
28     T();
29     Eprime();
30 }
```

```
E:\COMPILER DESIGN\LAB 11.exe
Enter an expression: a+b*t
Error in parsing at position 0!
-----
Process exited after 6.726 seconds with return value 1
Press any key to continue . . .
```

**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.**

### Aim:

To implement a Bottom-Up Parsing technique (Shift-Reduce Parsing) in C to check whether the given input string satisfies the specified grammar.

### C Code Implementation:

```
#include <stdio.h>

#include <string.h>

char stack[50];
char input[50];

int top = -1;
int ip = 0;

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

```

void pop() {
    top--;
}

void display() {
    printf("\nStack: %s\t Input: %s", stack, input + ip);
}

int reduce() {
    if (top >= 2) {
        if (stack[top] == 'E' && stack[top - 1] == '+' && stack[top - 2] == 'E') {
            printf("\nReduce by E -> E+E");
            top -= 2;
            return 1;
        }

        if (stack[top] == 'E' && stack[top - 1] == '*' && stack[top - 2] == 'E') {
            printf("\nReduce by E -> E*E");
            top -= 2;
            return 1;
        }
    }

    if (top >= 2) {
        if (stack[top] == ')' && stack[top - 1] == 'E' && stack[top - 2] == '(') {
            printf("\nReduce by E -> (E)");
            top -= 2;
            return 1;
        }
    }

    if (top >= 0) {

```



```

        if (stack[top] == 'a') {
            printf("\nReduce by E -> a");
            stack[top] = 'E';
            return 1;
        }
    }
    return 0;
}

int main() {
    printf("Enter the input string ending with $: ");
    scanf("%s", input);
    push('$');
    printf("\nBottom-Up Parsing (Shift-Reduce) Simulation:\n");
    display();
    while (1) {
        if (input[ip] != '\0') {
            push(input[ip++]);
            printf("\nShift '%c'", stack[top]);
            display();
        }
        while (reduce()) {
            display();
        }
        if (input[ip] == '\0' && top == 1 && stack[top] == 'E' && stack[0] == '$')
    {
        printf("\n\nThe input string is successfully parsed!\n");
        break;
    }
}

```

```

        if (input[ip] == '\0' && top != 1) {

            printf("\n\nError: The input string cannot be parsed by the given
grammar.\n");

            break;

        }

    }

    return 0;

}

```

```

LAB 11.cpp
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  char stack[100];
6  char input[100];
7  int top = -1;
8
9  void push(char c) {
10     stack[++top] = c;
11 }
12
13 void pop() {
14     top--;
15 }
16
17 void display() {
18     printf("\nStack: %.*s\tInput: %s", top + 1, stack, input);
19 }
20
21 int reduce() {
22     if (top >= 2 && stack[top - 2] == 'E' && (stack[top - 1] == '+' && stack[top] == 'E')) {
23         pop(); pop(); pop();
24         push('E');
25         return 1;
26     }
27     if (top >= 2 && stack[top - 2] == '(' && stack[top - 1] == '+' && stack[top] == 'E') {
28         pop(); pop(); pop();
29         push('E');
30         return 1;
31     }
32     return 0;
33 }

```

```

E:\COMPILER DESIGN\LAB 11.exe
Enter the input string (without spaces, id as 'i'): E -> E + E
Stack: E      Input: E
Parsing successful!

-----
Process exited after 3.301 seconds with return value 0
Press any key to continue . . .

```

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

##### Aim:

To implement the Shift-Reduce Parsing technique in C to parse a given input string based on a specified grammar.

##### Code:

```

#include <stdio.h>

#include <string.h>

```

```

char stack[50];
char input[50];
int top = -1, ip = 0;
void push(char c) { stack[++top] = c; }
void pop() { top--; }
int reduce() {
    if (top >= 2) {
        if (stack[top] == 'E' && stack[top - 1] == '+' && stack[top - 2] == 'E') {
            top -= 2;
            stack[top] = 'E';
            return 1;
        }
        if (stack[top] == 'E' && stack[top - 1] == '*' && stack[top - 2] == 'E') {
            top -= 2;
            stack[top] = 'E';
            return 1;
        }
    }
    if (top >= 2 && stack[top] == ')' && stack[top - 1] == 'E' && stack[top - 2] == '(') {
        top -= 2;
        stack[top] = 'E';
        return 1;
    }
    if (top >= 0 && stack[top] == 'a') {
        stack[top] = 'E';
        return 1;
    }
}


```

```

    return 0;
}
int main() {
    printf("Enter input string (end with $): ");
    scanf("%s", input);
    push('$');
    while (1) {
        if (input[ip] != '\0') {
            push(input[ip++]);
        }
        while (reduce());
        if (input[ip] == '\0' && top == 1 && stack[0] == '$' && stack[1] == 'E') {
            printf("String is successfully parsed!\n");
            break;
        }
        if (input[ip] == '\0' && top != 1) {
            printf("Error: Parsing failed.\n");
            break;
        }
    }
    return 0;
}

```

## Output:



The screenshot shows a C program in a text editor on the left and its execution output in a terminal window on the right. The program, LAB 11.cpp, implements a simple parser with a stack. It includes headers for stdio, stdlib, and string. It defines a stack array of 100 characters, an input array of 100 characters, and a top index initialized to -1. Functions include push, pop, display, and reduce. The reduce function checks for the expression E -> E + E. The terminal output shows the program being run, the input string 'E' being entered, and the stack and input being displayed as 'E'. The parsing is successful, and the program exits after 2.246 seconds.

```
LAB 11.cpp
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 char stack[100];
6 char input[100];
7 int top = -1;
8
9 void push(char c) {
10     stack[++top] = c;
11 }
12
13 void pop() {
14     top--;
15 }
16
17 void display() {
18     printf("\nStack: %s\tInput: %s", top + 1, stack);
19 }
20
21 int reduce() {
22     if (top >= 2 && stack[top - 2] == 'E' && stack[top - 1] == '+' && stack[top] == 'E') {
23         pop(); pop(); pop();
24         push('E');
25         return 1;
26     }
27     if (top >= 2 && stack[top - 2] == '(' && stack[top - 1] == ')' && stack[top] == 'E') {
28         pop(); pop(); pop();
29         push('E');
30         return 1;
31     }
32     return 0;
33 }
34
35 int main() {
36     printf("Enter the input string (without spaces, id as 'i'): E -> E + E\n");
37     fgets(input, 100, stdin);
38     display();
39     while (reduce()) {
40         display();
41     }
42     printf("Parsing successful!\n");
43     return 0;
44 }
```

```
EXCOMPILER DESIGN\LAB 11.exe
Enter the input string (without spaces, id as 'i'): E -> E + E

Stack: E      Input: E
Parsing successful!

-----
Process exited after 2.246 seconds with return value 0
Press any key to continue . . .
```

## 15. Write a C Program to implement Operator Precedence Parsing.

### Aim:

To implement Operator Precedence Parsing in C to parse and evaluate a given input expression using operator precedence relations.

### Code:

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

int prec(char c) {
    if (c == '+' || c == '-') return 1;
    if (c == '*' || c == '/') return 2;
    return 0;
}

void parse(char *exp) {
    char stack[50];
    int top = -1, i = 0;
```

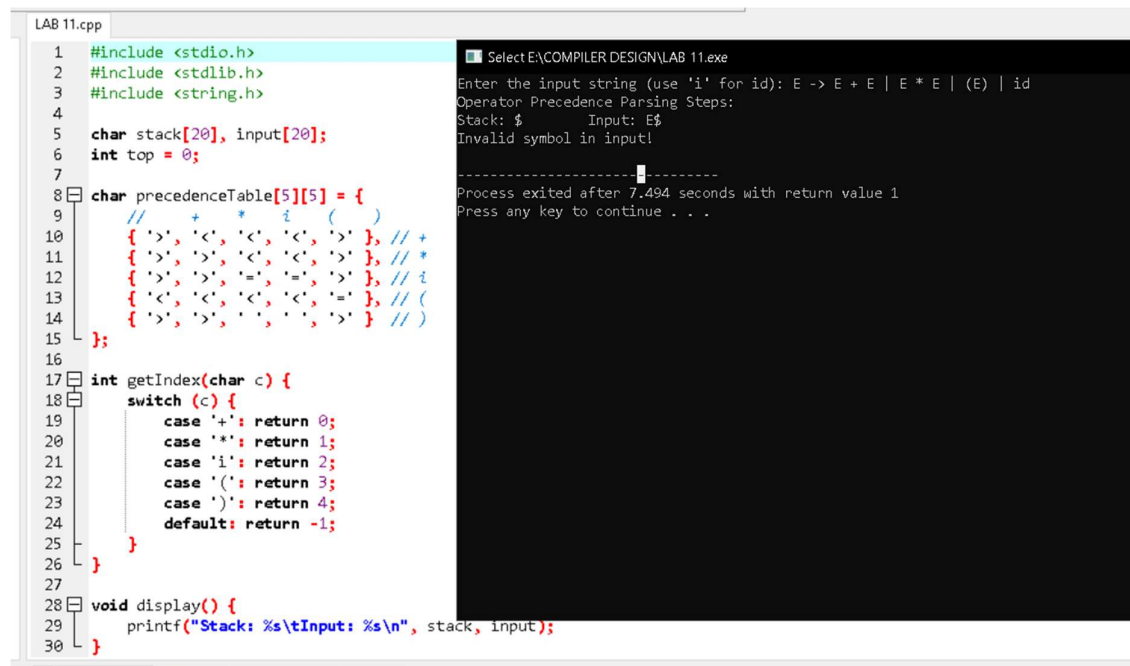
```

stack[++top] = '$'
while (exp[i] != '\0') {
    while (top >= 0 && prec(stack[top]) >= prec(exp[i])) {
        printf("Reduce by popping %c\n", stack[top--]);
    }
    printf("Shift %c\n", exp[i]);
    stack[++top] = exp[i++];
}
while (top > 0) {
    printf("Reduce by popping %c\n", stack[top--]);
}
printf("Parsing Successful!\n");
}

int main() {
    char exp[50];
    printf("Enter expression: ");
    scanf("%s", exp);
    parse(exp);
    return 0;
}

```

## Output:



The screenshot shows a C++ IDE with a file named 'LAB 11.cpp'. The code implements a precedence parser. It includes `<stdio.h>`, `<stdlib.h>`, and `<string.h>`. It defines a stack and input array of size 20, and a precedence table for operators '+', '\*', '(', and ')'. The `getIndex` function maps operators to indices 0-4. The `display` function prints the stack and input. The main function prompts the user for an input string and calls `display`. The output window shows the execution process, including the input string 'E', the stack '\$', and an error message 'Invalid symbol in input!'.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 char stack[20], input[20];
6 int top = 0;
7
8 char precedenceTable[5][5] = {
9     // + * ( )
10    { '>', '<', '<', '<', '>' }, // +
11    { '>', '>', '<', '<', '>' }, // *
12    { '>', '>', '=', '=', '>' }, // (
13    { '<', '<', '<', '<', '=' }, // )
14    { '>', '>', '>', '>', '>' } //
15};
16
17 int getIndex(char c) {
18     switch (c) {
19         case '+': return 0;
20         case '*': return 1;
21         case '(': return 2;
22         case ')': return 3;
23         case '=': return 4;
24         default: return -1;
25     }
26 }
27
28 void display() {
29     printf("Stack: %s\tInput: %s\n", stack, input);
30 }
```

Process exited after 7.494 seconds with return value 1  
Press any key to continue . . .

**16. Write a C Program to Generate the Three-Address Code representation for the given input statement.**

### Aim:

To implement a C program that generates three-address code for arithmetic expressions.

### Code:

```
#include <stdio.h>
#include <string.h>
int main() {
    char expr[50], temp[5] = "t";
    int i = 0, j = 1, k = 0;
    printf("Enter an arithmetic expression: ");
    scanf("%s", expr);
    printf("\nThree-Address Code:\n");
    while (expr[i] != '\0') {
        if (expr[i] == '+' || expr[i] == '-' || expr[i] == '*' || expr[i] == '/') {
            printf("t%d = %c %c %c\n", k++, expr[i - 1], expr[i], expr[i + 1]);
```

```

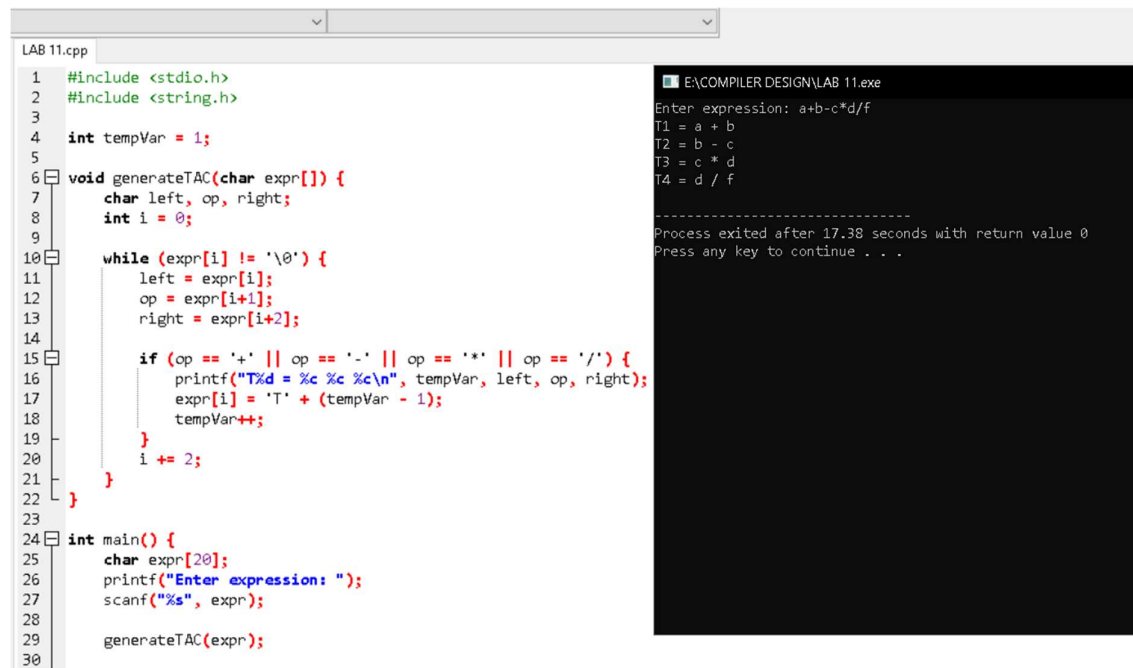
        i++;
    }

    i++;
}

return 0;
}

```

## Output:



The screenshot shows a C++ IDE with two panes. The left pane displays the source code for 'LAB 11.cpp', and the right pane shows the program's output in a terminal window.

**Source Code (LAB 11.cpp):**

```

1  #include <stdio.h>
2  #include <string.h>
3
4  int tempVar = 1;
5
6  void generateTAC(char expr[]) {
7      char left, op, right;
8      int i = 0;
9
10     while (expr[i] != '\0') {
11         left = expr[i];
12         op = expr[i+1];
13         right = expr[i+2];
14
15         if (op == '+' || op == '-' || op == '*' || op == '/') {
16             printf("T%d = %c %c %c\n", tempVar, left, op, right);
17             expr[i] = 'T' + (tempVar - 1);
18             tempVar++;
19         }
20         i += 2;
21     }
22 }
23
24 int main() {
25     char expr[20];
26     printf("Enter expression: ");
27     scanf("%s", expr);
28
29     generateTAC(expr);
30 }

```

**Output (Terminal Window):**

```

E:\COMPILER DESIGN\LAB 11.exe
Enter expression: a+b-c*d/f
T1 = a + b
T2 = b - c
T3 = c * d
T4 = d / f
-----
Process exited after 17.38 seconds with return value 0
Press any key to continue . . .

```

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

### Aim:

To implement a lexical analyzer in C that scans and counts characters, words, and lines from a given file.

### Code:

```

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

```



```

int main() {
    FILE *file;

    char filename[50], ch;

    int characters = 0, words = 0, lines = 0;

    printf("Enter filename: ");
    scanf("%s", filename);

    file = fopen(filename, "r");

    if (file == NULL) {
        printf("File not found!\n");
        return 1;
    }

    while ((ch = fgetc(file)) != EOF) {
        characters++;

        if (ch == ' ' || ch == '\t' || ch == '\n') words++;

        if (ch == '\n') lines++;
    }

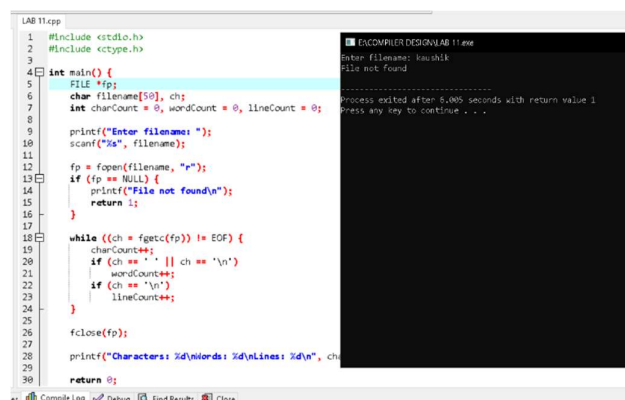
    fclose(file);

    printf("Characters: %d\nWords: %d\nLines: %d\n", characters, words, lines);

    return 0;
}

```

## Output:



The screenshot displays a C++ IDE with the source code on the left and the execution output on the right. The source code is a C++ program that counts characters, words, and lines in a file. The output window shows the program's execution, including the prompt 'Enter filename: kaushik', the error message 'File not found', and the final status 'Process exited after 5.005 seconds with return value 1'.

```

LAB 11.cpp
1 #include <stdio.h>
2 #include <ctype.h>
3
4 int main() {
5     FILE *fp;
6     char filename[50], ch;
7     int charCount = 0, wordCount = 0, lineCount = 0;
8
9     printf("Enter filename: ");
10    scanf("%s", filename);
11
12    fp = fopen(filename, "r");
13    if (fp == NULL) {
14        printf("File not found\n");
15        return 1;
16    }
17
18    while ((ch = fgetc(fp)) != EOF) {
19        charCount++;
20        if (ch == ' ' || ch == '\t' || ch == '\n')
21            wordCount++;
22        if (ch == '\n')
23            lineCount++;
24    }
25
26    fclose(fp);
27
28    printf("Characters: %d\nWords: %d\nLines: %d\n", charCount, wordCount, lineCount);
29
30    return 0;
31 }

```

Output:

```

Enter filename: kaushik
File not found
Process exited after 5.005 seconds with return value 1
Press any key to continue . . .

```

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

### Aim:

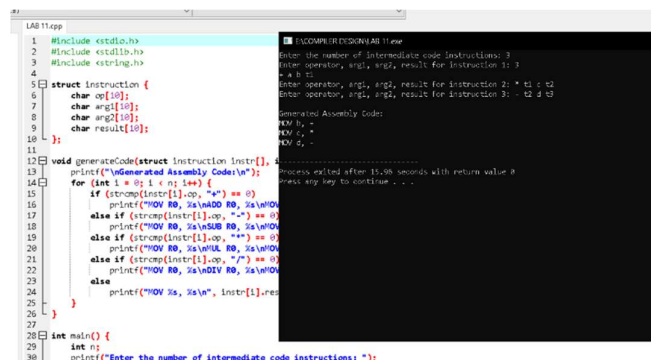
To implement the back end of a compiler in C that generates machine code for arithmetic expressions.

### Code:

```
#include <stdio.h>

int main() {
    char expr[50];
    printf("Enter arithmetic expression: ");
    scanf("%s", expr);
    printf("\nGenerated Assembly Code:\n");
    for (int i = 0; expr[i] != '\0'; i++) {
        if (expr[i] == '+') printf("ADD\n");
        else if (expr[i] == '-') printf("SUB\n");
        else if (expr[i] == '*') printf("MUL\n");
        else if (expr[i] == '/') printf("DIV\n");
        else printf("PUSH %c\n", expr[i]);
    }
    printf("POP RESULT\n");
    return 0;
}
```

### Output:



```
LAB11.cpp
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 struct Instruction {
6     char op[10];
7     char arg1[10];
8     char arg2[10];
9     char result[10];
10 };
11
12 void generateCode(struct Instruction Instr[], int n) {
13     printf("\nGenerated Assembly Code:\n");
14     for (int i = 0; i < n; i++) {
15         if (strcmp(Instr[i].op, "") == 0) {
16             printf("MOV R0, %s\n", Instr[i].result);
17         } else if (strcmp(Instr[i].op, "+") == 0) {
18             printf("MOV R0, %s\n", Instr[i].result);
19         } else if (strcmp(Instr[i].op, "-") == 0) {
20             printf("MOV R0, %s\n", Instr[i].result);
21         } else if (strcmp(Instr[i].op, "*") == 0) {
22             printf("MOV R0, %s\n", Instr[i].result);
23         } else if (strcmp(Instr[i].op, "/") == 0) {
24             printf("MOV R0, %s\n", Instr[i].result);
25         } else {
26             printf("MOV %s, %s\n", Instr[i].result, Instr[i].op);
27         }
28     }
29 }
30
31 int main() {
32     int n;
33     printf("Enter the number of intermediate code instructions: ");
34     scanf("%d", &n);
35     struct Instruction Instr[n];
36     printf("Enter operator, arg1, arg2, result for Instruction 1: ");
37     scanf("%s %s %s %s", Instr[0].op, Instr[0].arg1, Instr[0].arg2, Instr[0].result);
38     printf("Enter operator, arg1, arg2, result for Instruction 2: ");
39     scanf("%s %s %s %s", Instr[1].op, Instr[1].arg1, Instr[1].arg2, Instr[1].result);
40     printf("Enter operator, arg1, arg2, result for Instruction 3: ");
41     scanf("%s %s %s %s", Instr[2].op, Instr[2].arg1, Instr[2].arg2, Instr[2].result);
42     generateCode(Instr, n);
43     return 0;
44 }
```

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

**Aim:**

To implement a C program that computes the LEADING sets for operators in a given grammar.

**Code:**

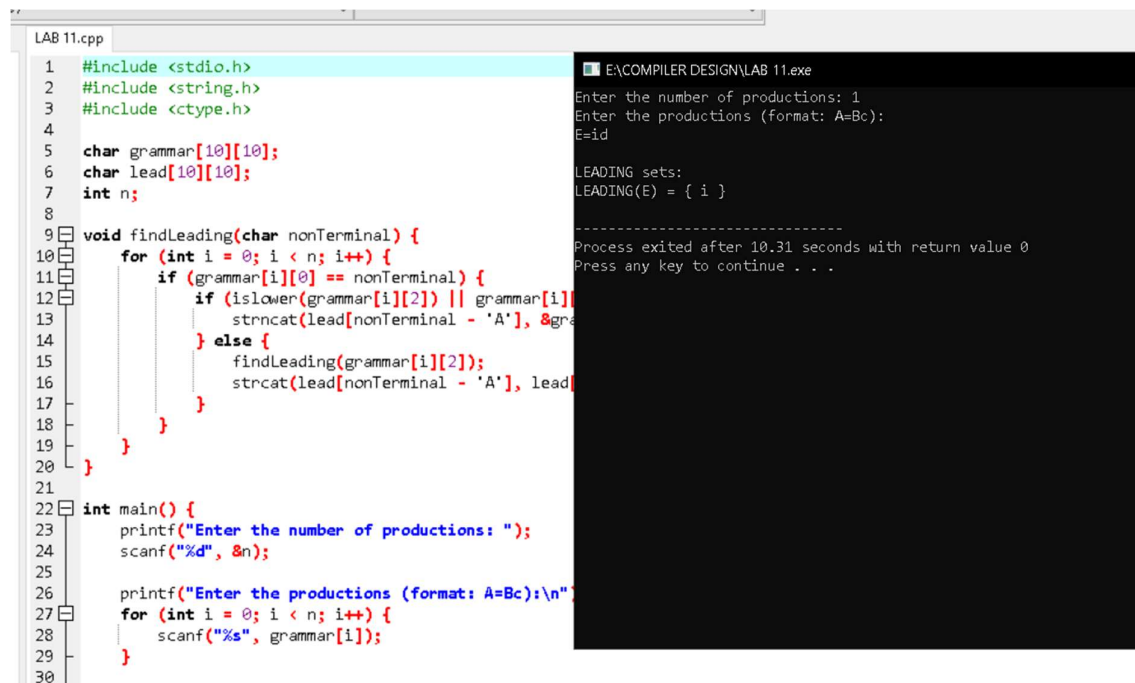
```
#include <stdio.h>

#include <string.h>

void findLeading(char *grammar, char terminal) {
    printf("Leading(%c) = {", terminal);
    for (int i = 0; i < strlen(grammar); i++) {
        if (grammar[i] == terminal && grammar[i + 1] != '\0') {
            printf(" %c ", grammar[i + 1]);
        }
    }
    printf("}\n");
}

int main() {
    char grammar[50], terminal;
    printf("Enter grammar (like E->E+T|T): ");
    scanf("%s", grammar);
    printf("Enter terminal symbol: ");
    scanf(" %c", &terminal);
    findLeading(grammar, terminal);
    return 0;
}
```

## Output:



```
LAB 11.cpp
1 #include <stdio.h>
2 #include <string.h>
3 #include <ctype.h>
4
5 char grammar[10][10];
6 char lead[10][10];
7 int n;
8
9 void findLeading(char nonTerminal) {
10     for (int i = 0; i < n; i++) {
11         if (grammar[i][0] == nonTerminal) {
12             if (islower(grammar[i][2]) || grammar[i][2] == '|') {
13                 strcat(lead[nonTerminal - 'A'], &grammar[i][2]);
14             } else {
15                 findLeading(grammar[i][2]);
16                 strcat(lead[nonTerminal - 'A'], lead[grammar[i][2] - 'A']);
17             }
18         }
19     }
20 }
21
22 int main() {
23     printf("Enter the number of productions: ");
24     scanf("%d", &n);
25
26     printf("Enter the productions (format: A=Bc):\n");
27     for (int i = 0; i < n; i++) {
28         scanf("%s", grammar[i]);
29     }
30 }
```

```
E:\COMPILER DESIGN\LAB 11.exe
Enter the number of productions: 1
Enter the productions (format: A=Bc):
E=id

LEADING sets:
LEADING(E) = { i }

-----
Process exited after 10.31 seconds with return value 0
Press any key to continue . . .
```

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

### Aim:

To implement a C program that computes the TRAILING sets for operators in a given grammar.

### Code:

```
#include <stdio.h>

#include <string.h>

void findTrailing(char *grammar, char terminal) {
    printf("Trailing(%c) = {", terminal);
    for (int i = 0; i < strlen(grammar); i++) {
        if (grammar[i] == terminal && i > 0) {
            printf(" %c ", grammar[i - 1]);
        }
    }
    printf("}\n");
}
```

```

}

int main() {

    char grammar[50], terminal;

    printf("Enter grammar (like E->E+T|T): ");

    scanf("%s", grammar);

    printf("Enter terminal symbol: ");

    scanf(" %c", &terminal);

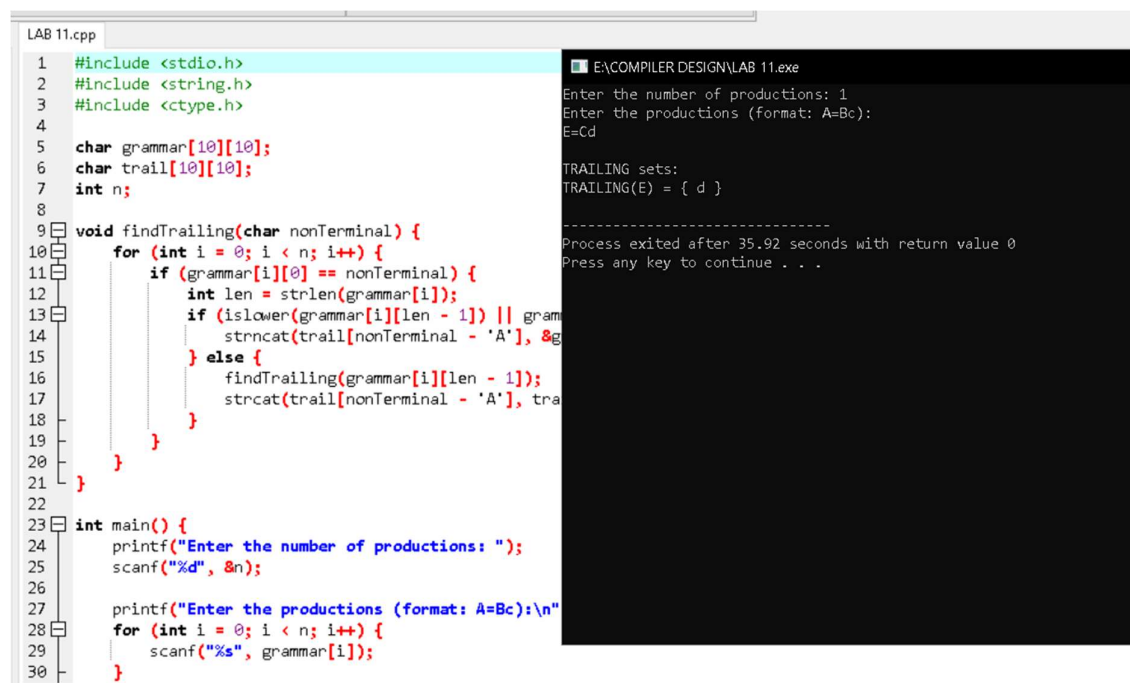
    findTrailing(grammar, terminal);

    return 0;

}

```

## Output:



The screenshot displays a C++ IDE with two panels. The left panel shows the source code for 'LAB 11.cpp', and the right panel shows the output of the program 'E:\COMPILER DESIGN\LAB 11.exe'.

**Source Code (LAB 11.cpp):**

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <ctype.h>
4
5  char grammar[10][10];
6  char trail[10][10];
7  int n;
8
9  void findTrailing(char nonTerminal) {
10     for (int i = 0; i < n; i++) {
11         if (grammar[i][0] == nonTerminal) {
12             int len = strlen(grammar[i]);
13             if (islower(grammar[i][len - 1]) || gram
14                 strcat(trail[nonTerminal - 'A'], &g
15             } else {
16                 findTrailing(grammar[i][len - 1]);
17                 strcat(trail[nonTerminal - 'A'], tra
18             }
19         }
20     }
21 }
22
23 int main() {
24     printf("Enter the number of productions: ");
25     scanf("%d", &n);
26
27     printf("Enter the productions (format: A=Bc):\n");
28     for (int i = 0; i < n; i++) {
29         scanf("%s", grammar[i]);
30     }

```

**Output (E:\COMPILER DESIGN\LAB 11.exe):**

```

Enter the number of productions: 1
Enter the productions (format: A=Bc):
E=Cd

TRAILING sets:
TRAILING(E) = { d }

-----
Process exited after 35.92 seconds with return value 0
Press any key to continue . . .

```