Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Spring, Year:2024), B.Sc. in CSE (Day)

Lab Report # 05

Course Title: Compiler Lab

Course Code: CSE 306 Section: 222 D2

Lab Experiment Name: To check whether a mathematical statement solvable or not

<u>Student Details</u>

Name	ID
Md. Moshiur Rahman	221902324

Submission Date : 05/05/2024

Course Teacher's Name : Tasnim Tayiba Zannat

<u>Lab Report Status</u>

Marks: ………………………………… **Signature:.....................**

Comments:... **Date:..............................**

1. EXPERIMENT NAME:

Write a program to recognize all kind of operators using number of array for each type of operators..

2. OBJECTIVES

- ★ recognizes all types of operators used in C programming language.
- ★ To use arrays to store different types of operators for easy maintenance and extensibility.
- ★ provide a user-friendly interface for entering an operator and displaying the corresponding type and description.

3. INTRODUCTION

Operators play a crucial role in performing various operations on data. This program is designed to recognize and classify different types of operators used in C, including arithmetic, relational, logical, bitwise, assignment, and increment/decrement operators.

4. ALGORITHM

1. Define arrays for different types of operators.
2. Initialize a flag variable isOperator to 0.
3. Get the input operator from the user.
4. Loop through each array of operators:

a. Check if the input matches the current operator in the array.

b. If a match is found, print the corresponding operator type and description, set isOperator to 1, and break the loop.

If isOperator is still 0 after checking all arrays, print "Not an operator".

Exit the program.

5. PROCEDURE

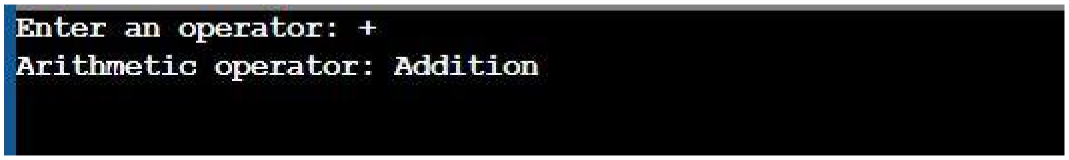
Implementation:

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

int main() {
        char input[50];
        int isOperator = 0;

        char arith_op[] = {'+', '-', '*', '/', '%'};
        char rel_op[] = {'<', '>', '!', '=', '='};
        char log_op[] = {'&', '&', '|', '|'};
        char bit_op[] = {'&', '|', '^', '~', '<', '<', '>', '>'};
        char assign_op[] = {'='};
        char inc_dec_op[] = {'+', '+', '-', '-'};

        printf("Enter an operator: ");
        fgets(input, sizeof(input), stdin);

        // Check for arithmetic operators
        for (int i = 0; i < sizeof(arith_op); i++) {
        if (input[0] == arith_op[i]) {
        printf("Arithmetic operator: ");
        switch (input[0]) {
        case '+':
                printf("Addition\n");
                break;
        case '-':
                printf("Subtraction\n");
                break;
        case '*':
                printf("Multiplication\n");
                break;
        case '/':
                printf("Division\n");
                break;
        case '%':
                printf("Modulus\n");
                break;
        }
        isOperator = 1;
        break;
        }
        }

        if (!isOperator) {
        for (int i = 0; i < sizeof(rel_op) - 1; i++) {
        if (input[0] == rel_op[i] && input[1] == rel_op[i + 1]) {
        printf("Relational operator: ");
        switch (input[0]) {
                case '<':
                printf("%s\n", input[1] == '=' ? "Less than or equal to" : "Less than");
```

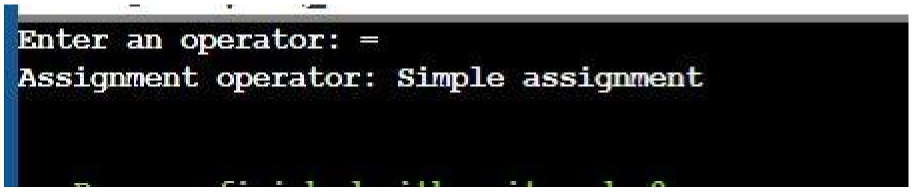
```
            break;
            case '>':
            printf("%s\n", input[1] == '=' ? "Greater than or equal to" : "Greater
than");
            break;
            case '!':
            printf("Not equal to\n");
            break;
            case '=':
            printf("Equal to\n");
            break;
    }
    isOperator = 1;
    break;
    }
    }
    }

    if (!isOperator) {
    for (int i = 0; i < sizeof(log_op); i += 2) {
    if (input[0] == log_op[i] && input[1] == log_op[i + 1]) {
    printf("Logical operator: %s\n", input[0] == '&' ? "Logical AND" : "Logical OR");
    isOperator = 1;
    break;
    }
    }
    }

    if (!isOperator) {
    for (int i = 0; i < sizeof(bit_op); i++) {
    if (input[0] == bit_op[i]) {
    printf("Bitwise operator: ");
    switch (input[0]) {
            case '&':
            printf("Bitwise AND\n");
            break;
            case '|':
            printf("Bitwise OR\n");
            break;
            case '^':
            printf("Bitwise XOR\n");
```

```
                break;
                case '~':
                printf("Bitwise Complement\n");
                break;
                case '<':
                printf("%s\n", input[1] == '<' ? "Left shift" : "Less than");
                break;
                case '>':
                printf("%s\n", input[1] == '>' ? "Right shift" : "Greater than");
                break;
        }
        isOperator = 1;
        break;
        }
        }
        }

        if (!isOperator) {
        for (int i = 0; i < sizeof(assign_op); i++) {
        if (input[0] == assign_op[i]) {
        printf("Assignment operator: %s\n", input[1] == '=' ? "Compound assignment" :
"Simple assignment");
        isOperator = 1;
        break;
        }
        }
        }

        if (!isOperator) {
        for (int i = 0; i < sizeof(inc_dec_op); i += 2) {
        if (input[0] == inc_dec_op[i] && input[1] == inc_dec_op[i + 1]) {
        printf("Increment/Decrement operator: %s\n", input[0] == '+' ? "Pre/Post
Increment" : "Pre/Post Decrement");
        isOperator = 1;
        break;
        }
        }
        }

        if (!isOperator) {
        printf("Not an operator\n");
        }

        return 0;
}
```

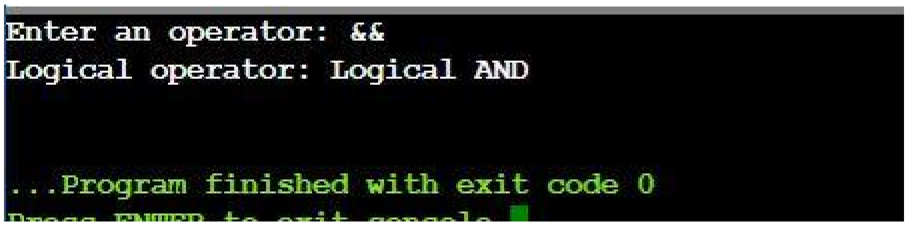
Output:



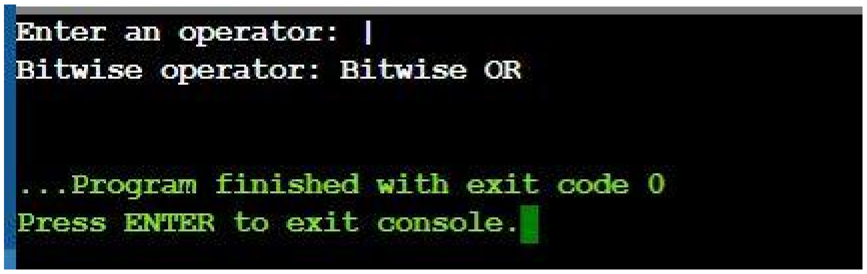
```
Enter an operator: +
Arithmetic operator: Addition
```



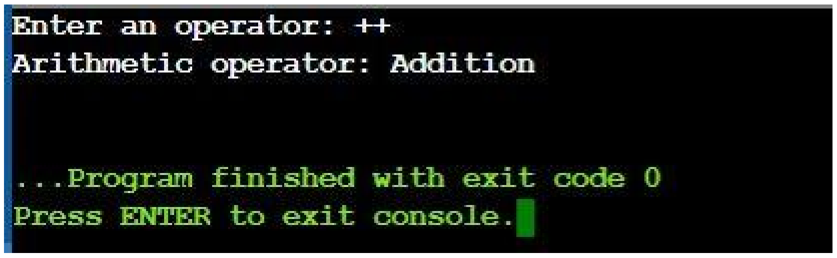
```
Enter an operator: =
Assignment operator: Simple assignment
```



```
Enter an operator: &&
Logical operator: Logical AND


...Program finished with exit code 0
```



```
Enter an operator: |
Bitwise operator: Bitwise OR


...Program finished with exit code 0
Press ENTER to exit console.
```



```
Enter an operator: ++
Arithmetic operator: Addition


...Program finished with exit code 0
Press ENTER to exit console.
```

6. ANALYSIS AND DISCUSSION

a. The program uses arrays to store different types of operators, making it easy to add or modify operators in the future.

b. The program checks for single-character operators as well as multi-character operators by comparing the first and second characters of the input.
c. Switch statements are used to print the appropriate operator description based on the input.
d. The program handles all types of operators used in C programming language, including arithmetic, relational, logical, bitwise, assignment, and increment/decrement operators.
e. The program assumes that the user enters a valid operator. Additional input validation can be added to handle invalid inputs.

7. SUMMARY

The provided C program recognizes all types of operators used in C programming language by utilizing arrays to store different types of operators. It prompts the user to enter an operator and then checks the input against each array of operators using loops and string comparisons. If a match is found, it prints the corresponding operator type and description. If no match is found, it prints "Not an operator". The program demonstrates the use of arrays, loops, string comparisons, and switch statements to achieve the desired functionality.