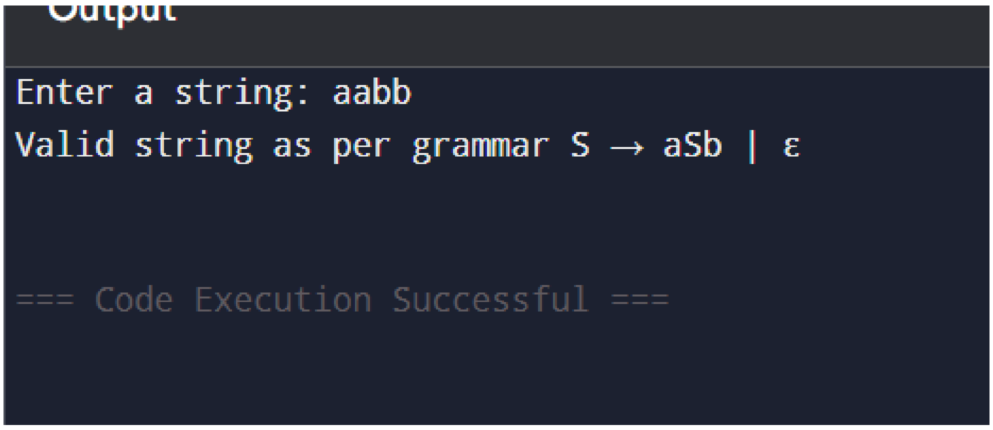
Experiment 9:

All languages have Grammar. When people frame a sentence, we usually say whether the sentence is framed as per the rules of the Grammar or Not. Similarly use the same ideology, implement to check whether the given input string is satisfying the grammar or not.

Code:

```
#include <stdio.h>
#include <string.h>
int checkGrammar(char str[]) {
   int i = 0, j = strlen(str) - 1;
   while (i <= j) {
      if (str[i] == 'a' && str[j] == 'b') {
         i++;
         j--;
      } else {
         return 0;
      }
   }
   return 1;
}
int main() {
   char str[100];
   printf("Enter a string: ");
   scanf("%s", str);
   if (checkGrammar(str))
      printf("Valid string as per grammar S → aSb | ε\n");
   else
      printf("Invalid string\n");
   return 0;
}
```

Output:



```
Enter a string: aabb
Valid string as per grammar S → aSb | ε


=== Code Execution Successful ===
```

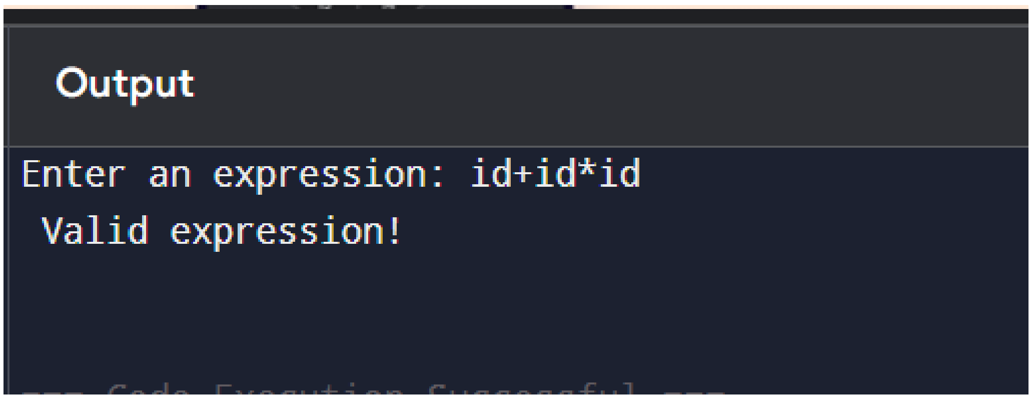
Experiment 10:

Write a C program to construct recursive descent parsing.

Code:

```
#include <stdio.h>
#include <string.h>
char input[100];
int i = 0;
void E();
void T();
void F();
void E() {
    T();
    if (input[i] == '+') {
        i++;
        E();
    }
}
void T() {
    F();
    if (input[i] == '*') {
        i++;
        T();
    }
}
void F() {
    if (input[i] == '(') {
        i++;
        E();
        if (input[i] == ')')
            i++;
    } else if (input[i] == 'i' && input[i + 1] == 'd') {
        i += 2; // accept "id"
    } else {
        printf(" Error at position %d\n", i);
    }
}
int main() {
    printf("Enter an expression: ");
    scanf("%s", input);
    E();
    if (input[i] == '\0')
        printf(" Valid expression! \n");
    else
        printf(" Invalid expression! \n");
    return 0;
}
```

Output:



```
Enter an expression: id+id*id
 Valid expression!



=== Code Execution Successful ===
```

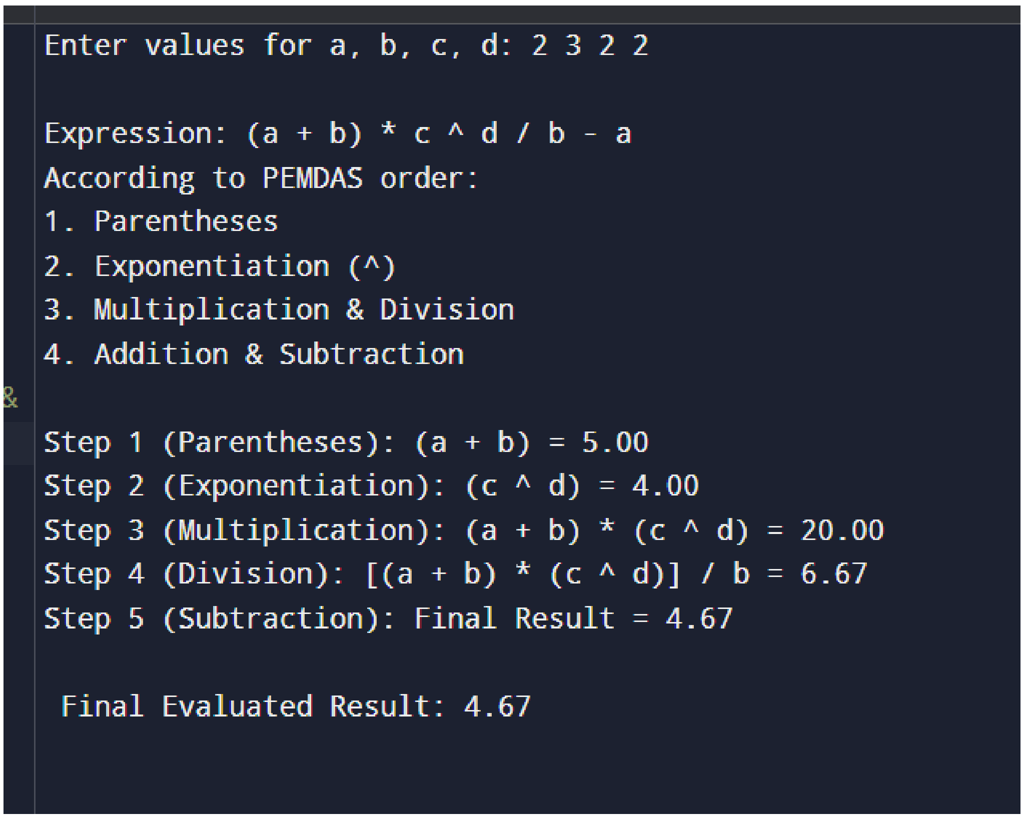
Experiment 11:

In a class of Grade 3, Mathematics Teacher asked for the Acronym PEMDAS?. All of them are thinking for a while. A smart kid of the class Kishore of the class says it is Parentheses, Exponentiation, Multiplication, Division, Addition, Subtraction. Can you write a C Program to help the students to understand about the operator precedence parsing for an expression containing more than one operator, the order of evaluation depends on the order of operations.

Code:

```
#include <stdio.h>
#include <math.h>
int main() {
    double a, b, c, d, result;
    printf("Enter values for a, b, c, d: ");
    scanf("%lf %lf %lf %lf", &a, &b, &c, &d);
    printf("\nExpression: (a + b) * c ^ d / b - a\n");
    printf("According to PEMDAS order:\n");
    printf("1. Parentheses\n2. Exponentiation (^)\n3. Multiplication & Division\n4. Addition &
Subtraction\n\n");
    double step1 = a + b;
    printf("Step 1 (Parentheses): (a + b) = %.2lf\n", step1);
    double step2 = pow(c, d);
    printf("Step 2 (Exponentiation): (c ^ d) = %.2lf\n", step2);
    double step3 = step1 * step2;
    printf("Step 3 (Multiplication): (a + b) * (c ^ d) = %.2lf\n", step3);
    double step4 = step3 / b;
    printf("Step 4 (Division): [(a + b) * (c ^ d)] / b = %.2lf\n", step4);
    result = step4 - a;
    printf("Step 5 (Subtraction): Final Result = %.2lf\n", result);
    printf("\n Final Evaluated Result: %.2lf\n", result);
    return 0;
}
```

Output:



```
Enter values for a, b, c, d: 2 3 2 2

Expression: (a + b) * c ^ d / b - a
According to PEMDAS order:
1. Parentheses
2. Exponentiation (^)
3. Multiplication & Division
4. Addition & Subtraction

Step 1 (Parentheses): (a + b) = 5.00
Step 2 (Exponentiation): (c ^ d) = 4.00
Step 3 (Multiplication): (a + b) * (c ^ d) = 20.00
Step 4 (Division): [(a + b) * (c ^ d)] / b = 6.67
Step 5 (Subtraction): Final Result = 4.67

 Final Evaluated Result: 4.67
```

Experiment 12:

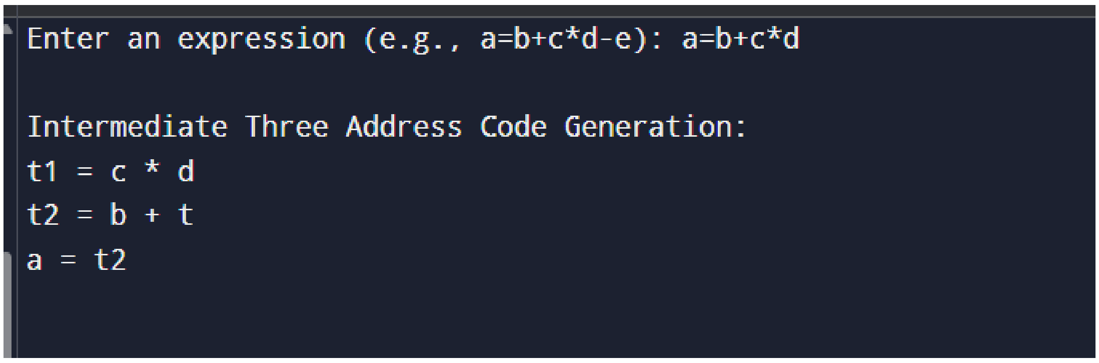
The main function of the Intermediate code generation is producing three address code statements for a given input expression. The three address codes help in determining the sequence in which operations are actioned by the compiler. The key work of Intermediate code generators is to simplify the process of Code Generator. Write a C Program to Generate the Three address code representation for the given input statement.

Code:

```
#include <stdio.h>
#include <string.h>
int main() {
   char expr[50];
   char tempVar = '1';
   printf("Enter an expression (e.g., a=b+c*d-e): ");
   scanf("%s", expr);
   int len = strlen(expr);
   char lhs = expr[0];
   printf("\nIntermediate Three Address Code Generation:\n");
   for (int i = 0; i < len; i++) {
      if (expr[i] == '*' || expr[i] == '/') {
         printf("t%c = %c %c %c\n", tempVar, expr[i-1], expr[i], expr[i+1]);
         expr[i-1] = 't';
         expr[i] = tempVar;
         tempVar++;
      }
   }
   for (int i = 0; i < len; i++) {
      if (expr[i] == '+' || expr[i] == '-') {
         printf("t%c = %c %c %c\n", tempVar, expr[i-1], expr[i], expr[i+1]);
         expr[i-1] = 't';
         expr[i] = tempVar;
         tempVar++;
      }
   }
   printf("%c = t%c\n", lhs, tempVar - 1);
```

```
    return 0;
}
```

Output:



```
Enter an expression (e.g., a=b+c*d-e): a=b+c*d

Intermediate Three Address Code Generation:
t1 = c * d
t2 = b + t
a = t2
```