# LAB PROGRAMS 9_12

M.SAI HARSHITHA

192424408

CSE AI& DS

CSAA1402

9. All languages have Grammar. When people frame a sentence we usually say whether the sentence is framed as perthe 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 && str[i] == 'a' && str[j] == 'b') {
      i++;
      j--;
   }
   if (i > j)
      return 1;
   else
      return 0;
}
int main() {
   char str[50];
   printf("Enter a string: ");
   scanf("%s", str);
   if (checkGrammar(str))
      printf("String is valid as per grammar S -> aSb | ab\n");
   else
      printf("String is NOT valid as per grammar S -> aSb | ab\n");
   return 0;
}
```

**SAMPLE OUTPUT:**

Enter a string: aabb

String is valid as per grammar S -> aSb | ab

**OUTPUT:**

```
Enter a string: AABB
String is NOT valid as per grammar S -> aSb | ab

--------------------------------
Process exited after 14.81 seconds with return value 0
Press any key to continue . . .
```

10. Write a C program to constructrecursive descent parsing.

**CODE:**

```
#include <stdio.h>
#include <string.h>
char input[20];
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;
    else
        printf("Error\n");
}
int main() {
    printf("Enter expression: ");
    scanf("%s", input);
    E();
    if (input[i] == '\0')
        printf("Parsing Successful!\n");
    else
        printf("Parsing Failed!\n");
    return 0;
}
```

SAMPLE OUTPUT:

Enter an expression: id+id*id

Parsing successful!

**OUTPUT:**

```
Enter expression: id+id*id
Parsing Successful!

--------------------------------
Process exited after 50.49 seconds with return value 0
Press any key to continue . . .
```

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

**CODE:**

```
#include <stdio.h
int main() {
   int a = 10, b = 5, c = 2;
   int result;

   printf("Expression: a + b * c\n");
   result = a + b * c;
   printf("Result = %d (Multiplication before Addition)\n\n", result);

   printf("Expression: (a + b) * c\n");
   result = (a + b) * c;
   printf("Result = %d (Parentheses first)\n\n", result);
```

```
    printf("Expression: a + b / c\n");
    result = a + b / c;
    printf("Result = %d (Division before Addition)\n\n", result);

    printf("Expression: a - b + c\n");
    result = a - b + c;
    printf("Result = %d (Left to Right for same level)\n\n", result);

    printf("\nPEMDAS means:\n");
    printf("P - Parentheses\nE - Exponentiation\nM - Multiplication\nD - Division\nA - Addition\nS - Subtraction\n");

    return 0;
}
```

**SAMPLE OUTPUT:**

Expression: a + b * c

Result = 20 (Multiplication before Addition)

Expression: (a + b) * c

Result = 30 (Parentheses first)

Expression: a + b / c

Result = 12 (Division before Addition)

Expression: a - b + c

Result = 7 (Left to Right for same level)

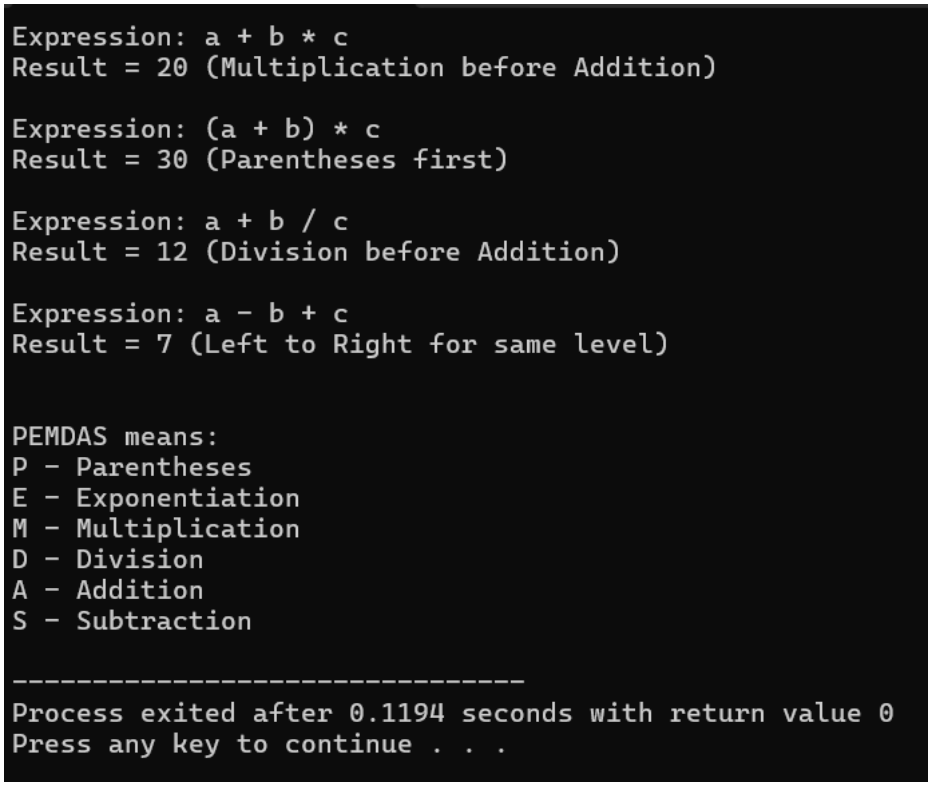
PEMDAS means:

P – Parentheses D - Division

A – Addition M - Multiplication

S – Subtraction E - Exponentiation

**OUTPUT:**




**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>

int main() {

   char a, b, c, op1, op2;

   printf("Enter expression (like a+b*c): ");

   scanf("%c%c%c%c%c", &a, &op1, &b, &op2, &c);

   if (op2 == '*' || op2 == '/')

      printf("t1 = %c %c %c\n", b, op2, c),
```

```
        printf("t2 = %c %c t1\n", a, op1);
    else
        printf("t1 = %c %c %c\n", a, op1, b),
        printf("t2 = t1 %c %c\n", op2, c);
    printf("Result in t2\n");
    return 0;
}
```

**SAMPLE CODE:**

```
Enter expression (like a+b*c): a+b*c
t1 = b * c
t2 = a + t1
Result in t2
```

**OUTPUT:**

```
Enter expression (like a+b*c): a+b*c
t1 = b * c
t2 = a + t1
Result in t2

--------------------------------
Process exited after 21.08 seconds with return value 0
Press any key to continue . . .
```