192311044

Girija B

19.02.2025

CSA1428 - Compiler Design

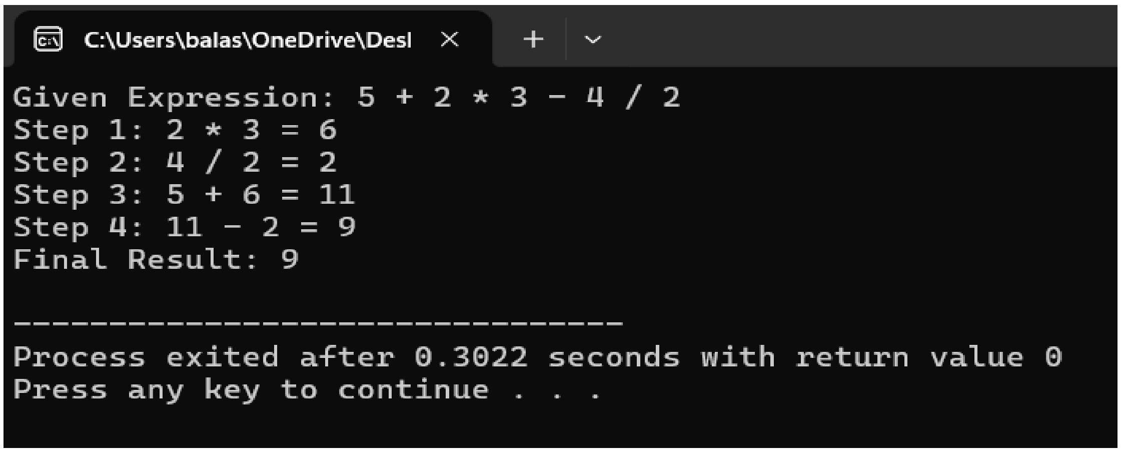
LAB ACTIVITY-2

1. 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>
int main() {
    int a = 5, b = 2, c = 3, d = 4;
    int result;
    printf("Given Expression: %d + %d * %d - %d / %d\n", a, b, c, d, b);
    int step1 = b * c;
    printf("Step 1: %d * %d = %d\n", b, c, step1);


    int step2 = d / b;
    printf("Step 2: %d / %d = %d\n", d, b, step2);


    int step3 = a + step1;
    printf("Step 3: %d + %d = %d\n", a, step1, step3);
    result = step3 - step2;
    printf("Step 4: %d - %d = %d\n", step3, step2, result);
    printf("Final Result: %d\n", result);
    return 0;
}
```

Output:



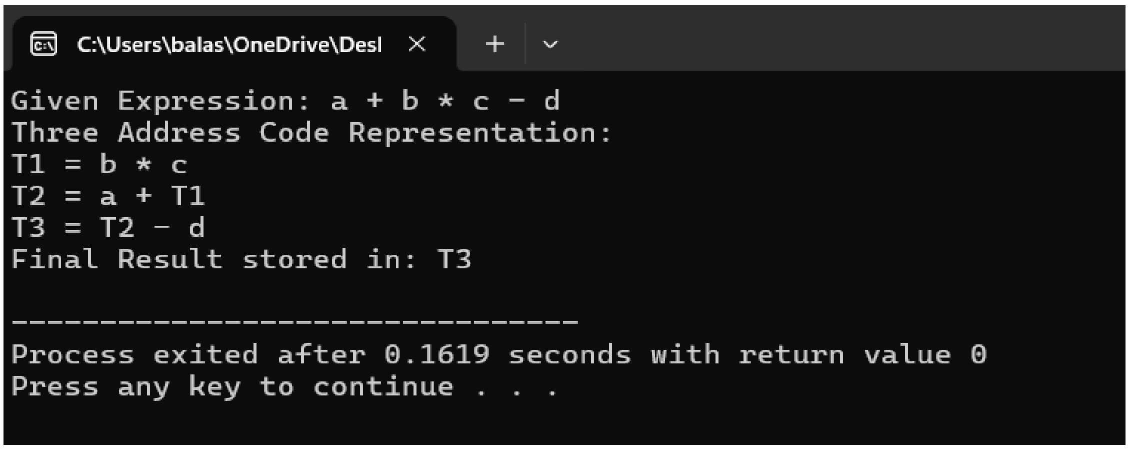
```
Given Expression: 5 + 2 * 3 - 4 / 2
Step 1: 2 * 3 = 6
Step 2: 4 / 2 = 2
Step 3: 5 + 6 = 11
Step 4: 11 - 2 = 9
Final Result: 9

--------------------------------
Process exited after 0.3022 seconds with return value 0
Press any key to continue . . .
```

2. 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 tempVar = 1;

void generateTAC(char op, char arg1[], char arg2[], char result[]) {

   printf("%s = %s %c %s\n", result, arg1, op, arg2);

}

int main() {

   char expr[] = "a + b * c - d";

   char t1[] = "T1", t2[] = "T2", t3[] = "T3";

   printf("Given Expression: %s\n", expr);

   printf("Three Address Code Representation:\n");

   generateTAC('*', "b", "c", t1);  // T1 = b * c

   generateTAC('+', "a", t1, t2);   // T2 = a + T1

   generateTAC('-', t2, "d", t3);   // T3 = T2 – d

   printf("Final Result stored in: %s\n", t3);

   return 0;

}
```

Output:



```
Given Expression: a + b * c - d
Three Address Code Representation:
T1 = b * c
T2 = a + T1
T3 = T2 - d
Final Result stored in: T3

--------------------------------
Process exited after 0.1619 seconds with return value 0
Press any key to continue . . .
```

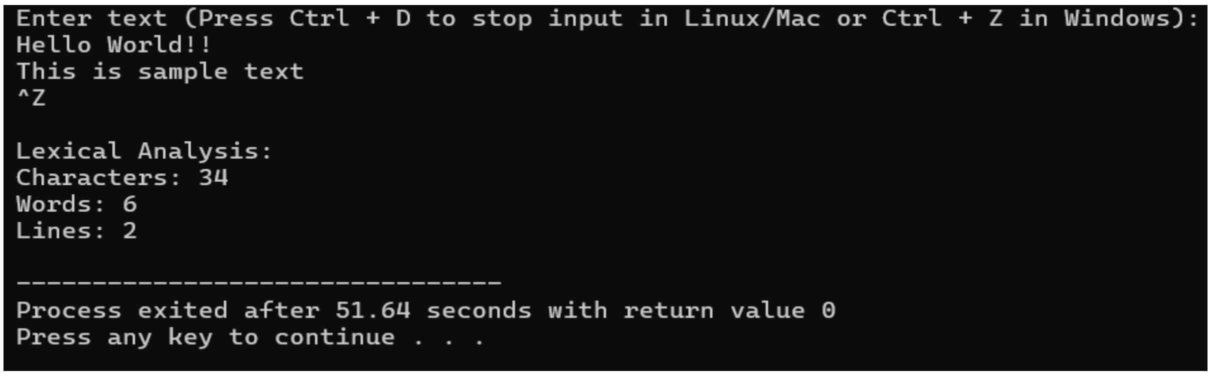
3. Write a C program for implementing a Lexical Analyzer to Count the number of characters, words, and lines .

Code:

```c
#include <stdio.h>
int main() {
   char ch;
   int characters = 0, words = 0, lines = 0;
   char lastChar = ' ';
   printf("Enter text (Press Ctrl + D to stop input in Linux/Mac or Ctrl + Z in Windows):\n");
   while ((ch = getchar()) != EOF) {
      characters++;

      if (ch == '\n')
         lines++;
      if ((ch == ' ' || ch == '\n' || ch == '\t') && (lastChar != ' ' && lastChar != '\n' && lastChar != '\t')) {
         words++;
      }
      lastChar = ch;
   }
   if (lastChar != ' ' && lastChar != '\n' && lastChar != '\t')
      words++;
```

```
    printf("\nLexical Analysis:\n");
    printf("Characters: %d\n", characters);
    printf("Words: %d\n", words);
    printf("Lines: %d\n", lines);
    return 0;
}
```

Output:



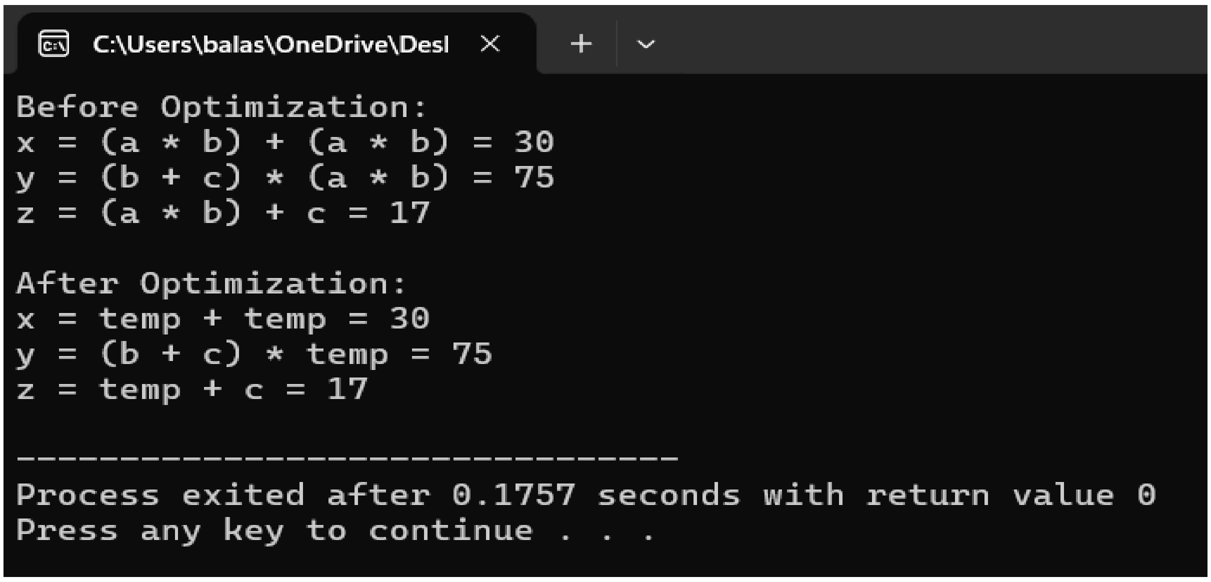
```
Enter text (Press Ctrl + D to stop input in Linux/Mac or Ctrl + Z in Windows):
Hello World!!
This is sample text
^Z

Lexical Analysis:
Characters: 34
Words: 6
Lines: 2

--------------------------------
Process exited after 51.64 seconds with return value 0
Press any key to continue . . .
```

4. Write a C Program for code optimization to eliminate common subexpression.

Code:

```
#include <stdio.h>
int main() {
    int a = 5, b = 3, c = 2, x, y, z;
    x = (a * b) + (a * b);
    y = (b + c) * (a * b);
    z = (a * b) + c;
    printf("Before Optimization:\n");
    printf("x = (a * b) + (a * b) = %d\n", x);
    printf("y = (b + c) * (a * b) = %d\n", y);
    printf("z = (a * b) + c = %d\n", z);
    int temp = a * b;
    x = temp + temp;
    y = (b + c) * temp;
    z = temp + c;
    printf("\nAfter Optimization:\n");
    printf("x = temp + temp = %d\n", x);
    printf("y = (b + c) * temp = %d\n", y);
    printf("z = temp + c = %d\n", z);
    return 0;
}
```

Output:



```
Before Optimization:
x = (a * b) + (a * b) = 30
y = (b + c) * (a * b) = 75
z = (a * b) + c = 17

After Optimization:
x = temp + temp = 30
y = (b + c) * temp = 75
z = temp + c = 17

--------------------------------
Process exited after 0.1757 seconds with return value 0
Press any key to continue . . .
```

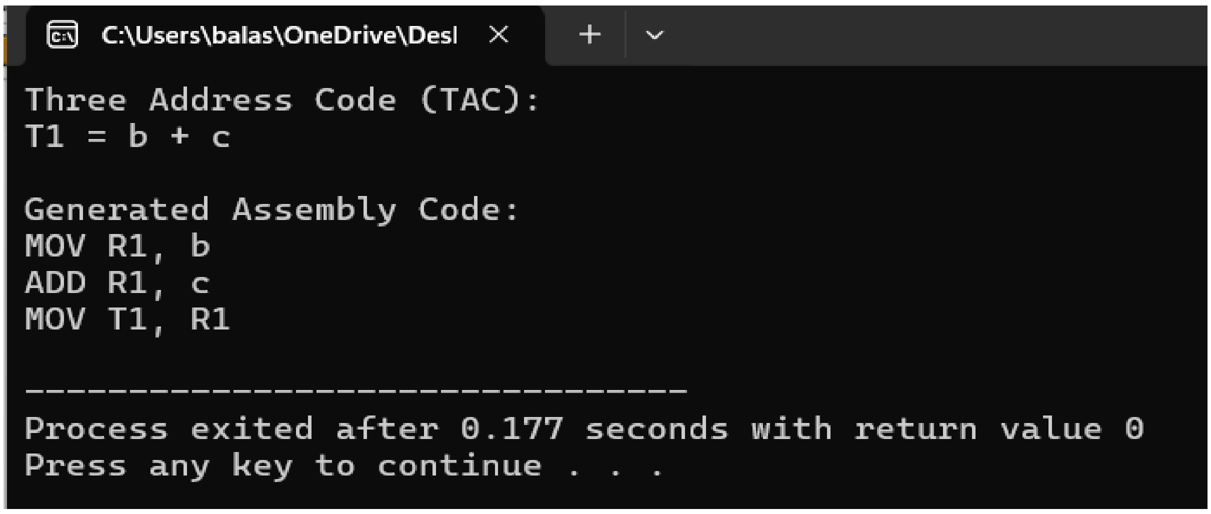
5. Write a C program to implement the back end of the compiler.

Code:

```
#include <stdio.h>
#include <string.h>
void generateAssembly(char op, char arg1[], char arg2[], char result[]) {
    switch (op) {
        case '+': printf("MOV R1, %s\nADD R1, %s\nMOV %s, R1\n", arg1, arg2, result); break;
        case '-': printf("MOV R1, %s\nSUB R1, %s\nMOV %s, R1\n", arg1, arg2, result); break;
        case '*': printf("MOV R1, %s\nMUL R1, %s\nMOV %s, R1\n", arg1, arg2, result); break;
        case '/': printf("MOV R1, %s\nDIV R1, %s\nMOV %s, R1\n", arg1, arg2, result); break;
        default: printf("Invalid Operation\n");
    }
}
```

```c
int main() {
    char result[] = "T1";
    char arg1[] = "b";
    char arg2[] = "c";
    char op = '+';

    printf("Three Address Code (TAC):\n");
    printf("%s = %s %c %s\n", result, arg1, op, arg2);
    printf("\nGenerated Assembly Code:\n");
    generateAssembly(op, arg1, arg2, result);
    return 0;
}
```

Output:



```
Three Address Code (TAC):
T1 = b + c

Generated Assembly Code:
MOV R1, b
ADD R1, c
MOV T1, R1

--------------------------------
Process exited after 0.177 seconds with return value 0
Press any key to continue . . .
```