Shri Ramdeobaba College of Engineering and Management,
Nagpur Department of CSE Cyber Security

Session: 2023-2024
Compiler Design Lab CCP308

PRACTICAL No. 8

Name: Saloni Vishwakarma(C1-13)
Topic: Code Generation
Platform: Windows or Linux

Aim:

a. Write a program to generate the code using **simple code generation algorithm.** Assume two registers R0 and R1

Solve this to get TAC x = (a * (c – (b + d * e)))

Input:

Three Address Code got from above equation

Sample Input:

1. T1 = a + b
2. T2 = c + d
3. T3 = e – T2
4. x = T1 - T3

Sample Output:

MOVE a, Ro **ADD b, Ro**
MOVE c, R1 **ADD d, R1**
MOVE R0, t1 **MOVE e, R0** **SUB R1, R0**
MOVE t1, R1

```
SUB R0, R1
```

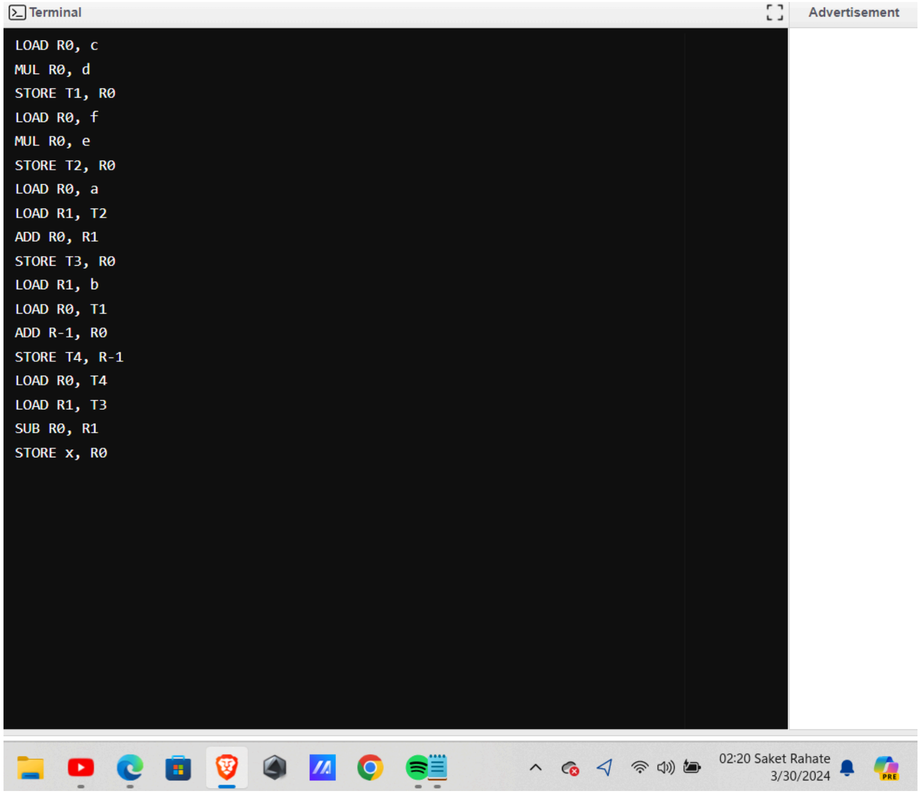
CODE :

```
public class TACGenerator {
    public static void main(String[] args) {
        String[] tac = {
            "T1 = c * d",
            "T2 = f * e",
            "T3 = a + T2",
            "T4 = b + T1",
            "x = T4 - T3"
        };
        generateAssembly(tac);
    }

    private static void generateAssembly(String[] tac) {
        int tempRegister = 0; // Keeps track of available temporary registers
        for (String instruction : tac) {
```

```
String[] parts = instruction.split(" = ");
String destination = parts[0];
String expression = parts[1];
String[] operands = expression.split(" ");

switch (operands[1]) {
    case "*":
        // Handle multiplication
        System.out.println("LOAD R" + tempRegister + ", " + operands[0]);
        System.out.println("MUL R" + tempRegister + ", " + operands[2]);
        System.out.println("STORE " + destination + ", R" + tempRegister);
        break;
    case "+":
        // Load operands into registers (assuming operands are variables)
        System.out.println("LOAD R" + tempRegister + ", " + operands[0]);
        tempRegister = (tempRegister + 1) % 2; // Alternate between R0 and R1
        System.out.println("LOAD R" + tempRegister + ", " + operands[2]);
```

```
                // Add operands and store in destination register
                System.out.println("ADD R" + (tempRegister - 1) + ", R" +
tempRegister);
                System.out.println("STORE " + destination + ", R" + (tempRegister -
1));
                break;
            case "-":
                // Similar logic as addition, but using SUB instruction
                System.out.println("LOAD R" + tempRegister + ", " + operands[0]);
                tempRegister = (tempRegister + 1) % 2;
                System.out.println("LOAD R" + tempRegister + ", " + operands[2]);
                System.out.println("SUB R" + (tempRegister - 1) + ", R" +
tempRegister);
                System.out.println("STORE " + destination + ", R" + (tempRegister -
1));
                break;
            default:
```

```
// Handle other operators (if needed)

System.out.println("Unsupported operator: " + operands[1]);

break;  }   }  }  }
```



```
LOAD R0, c
MUL R0, d
STORE T1, R0
LOAD R0, f
MUL R0, e
STORE T2, R0
LOAD R0, a
LOAD R1, T2
ADD R0, R1
STORE T3, R0
LOAD R1, b
LOAD R0, T1
ADD R-1, R0
STORE T4, R-1
LOAD R0, T4
LOAD R1, T3
SUB R0, R1
STORE x, R0
```

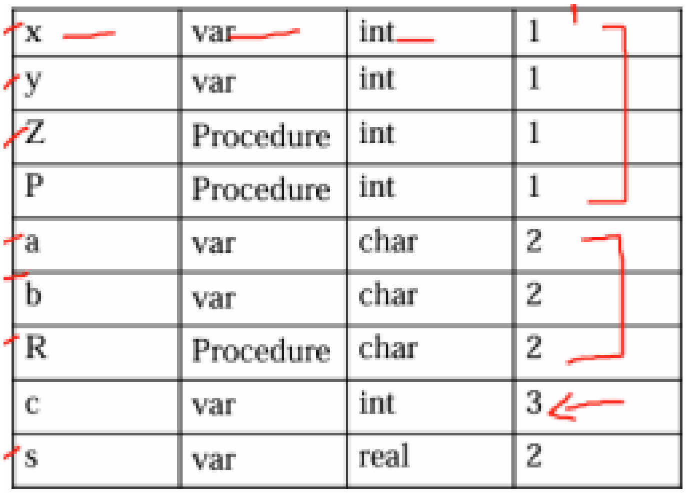
b. Write and algorithm and a program to identify the scope of Information in Symbol Table

Sample Input:

```
main()
{
int x,y;
int z()
          {
          char a,b;
          char r()
                    {
                    int c;
                    }
          }
int p()
          {
          Real s;
          }
}
```

Sample Output:

Identifier Name	Identifier Semantic	Identifier Type	Scope Depth



x	var	int	1
y	var	int	1
Z	Procedure	int	1
P	Procedure	int	1
a	var	char	2
b	var	char	2
R	Procedure	char	2
c	var	int	3
s	var	real	2

// Algorithm:
Initialize a global symbol table.

1. For each function definition, create a new scope in the symbol table.
2. For each variable definition in the current scope, add the variable to the current scope in the symbol table.
3. If a variable is defined in an inner scope, search for the variable in the current scope and, if not found, search in the outer scopes.
4. When leaving a scope, remove the scope from the symbol table.