NAME: Swanand M Garge
ROLL NO:39
SRN:202201589
DIV:D(D2)
SUBJECT: SYSTEM PROGRAMMING

ASSIGNMENT- 3

Implement Pass-II of a two pass assembler designed in assignment 1. Use the output of Assignment-1 (intermediate code file, MOT and symbol table) as input for this assignment.

Output will be machine code for the IC. Submit a single .pdf / .doc file containing IC, Symbol table and machine code with LC and your source code in that sequence.

Input IC:

```
Intermediate Code with LC:
LC
200   (AD, 10)   (C, 200)
200   (IS, 24)   (R, 1)   (R, 2)
202   (IS, 20)   (R, 1)   (R, 3)
204   (IS, 21)   (R, 2)   (R, 4)
207   (IS, 22)   (R, 4)   (C, 3)
209   (IS, 27)   (R, 1)   (S, 0)
211   (IS, 24)   (R, 1)   (C, 10)
213   (DL, 1)   (C, 5)
218   (DL, 2)   (C, 2)
219   (IS, 24)   (R, 1)   (S, 3)
221   (AD, 11)
```

Input Symbol Table:

```
Symbol Table:
0    C    209
1    y    213
2    LOOP:    218
3    z    219
```

Output Machine Code:

```
Machine Code:
LC    Machine Code
200   24 1 2
202   20 1 3
204   21 2 4
207   22 4 3
209   27 1 209
211   24 1 10
218   2
219   24 1 219
```

Complete Output (machine_code.txt):

SP > machine_code.txt

```
Symbol Table:
0    C    209
1    y    213
2    LOOP:    218
3    z    219

Intermediate Code with LC:
LC
200  (AD, 10)   (C, 200)
200  (IS, 24)   (R, 1)   (R, 2)
202  (IS, 20)   (R, 1)   (R, 3)
204  (IS, 21)   (R, 2)   (R, 4)
207  (IS, 22)   (R, 4)   (C, 3)
209  (IS, 27)   (R, 1)   (S, 0)
211  (IS, 24)   (R, 1)   (C, 10)
213  (DL,  1)   (C, 5)
218  (DL,  2)   (C, 2)
219  (IS, 24)   (R, 1)   (S, 3)
221  (AD, 11)

Machine Code:
LC    Machine Code
200   24 1 2
202   20 1 3
204   21 2 4
207   22 4 3
209   27 1 209
211   24 1 10
218   2
219   24 1 219

```

Source Code:

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

class SP_3 {
    static class MOTEntry {
        String  name;
        String type;
        int opcode;
        int size;

        MOTEntry(String name, String type, int opcode, int size) {
            this.name = name;
            this.type = type;
            this.opcode = opcode;
            this.size = size;
        }
    }

    static class Register {
        String name;
        int code;

        Register(String name, int code) {
            this.name = name;
            this.code = code;
        }
    }

    static class Symbol {
        String name;
        int address;

        Symbol(String name, int address) {
            this.name = name;
```

```
            this.address = address;
        }
    }

    static class IntermediateCode {
        String operation;
        int code;
        int reg1;
        int reg2;
        String operandType;
        int operandValue;
        int lc;

        IntermediateCode(String operation, int code, int reg1, int reg2,
String operandType, int operandValue, int lc) {
            this.operation = operation;
            this.code  =  code;
            this.reg1  =  reg1;
            this.reg2 = reg2;
            this.operandType = operandType;
            this.operandValue = operandValue;
            this.lc = lc;
        }
    }

    static class MachineCode {
        int lc;
        String binaryCode;

        MachineCode(int lc, String binaryCode) {
            this.lc = lc;
            this.binaryCode = binaryCode;
        }
    }

    static List<Symbol> symbolTable = new ArrayList<>();
    static List<IntermediateCode> intermediateCode = new ArrayList<>();

    static List<MachineCode> generateMachineCode() {
        List<MachineCode> machineCodeList = new ArrayList<>();

        for (IntermediateCode ic : intermediateCode) {
            if (ic.operation.equals("IS")) {
                StringBuilder machineInstruction = new StringBuilder();
                machineInstruction.append(String.format("%02d", ic.code));

                // Add register or constant values correctly
                if (ic.reg1 != -1) {
```

```
                    machineInstruction.append(" ").append(ic.reg1);
                } else {
                    machineInstruction.append(" 00");
                }

                if (ic.reg2 != -1) {
                    machineInstruction.append(" ").append(ic.reg2);
                } else if (ic.operandType != null) {
                    switch (ic.operandType) {
                        case "S":
                            // For symbols, use their address in the machine
code
                            int symbolAddress =
symbolTable.get(ic.operandValue).address;
                            machineInstruction.append("
").append(symbolAddress);
                            break;
                        case "C":
                            // For constants, use the constant value directly
                            machineInstruction.append("
").append(ic.operandValue);
                            break;
                    }
                } else {

                    machineInstruction.append(" 000");
                }

                machineCodeList.add(new MachineCode(ic.lc,
machineInstruction.toString().trim()));
            } else if (ic.operation.equals("DL") && ic.code == 2) { // DC
statement
                machineCodeList.add(new MachineCode(ic.lc,
String.valueOf(ic.operandValue)));
            }
            // Skip other AD and DL instructions for machine code generation
        }

        return machineCodeList;
    }



    static void writeOutputToFile(String filename, List<MachineCode>
machineCodeList) throws IOException {
        BufferedWriter bw = new BufferedWriter(new FileWriter(filename));

        // Write Symbol Table
        bw.write("Symbol Table:\n");
```

```
        for (int i = 0; i < symbolTable.size(); i++) {
            bw.write(i + "     " + symbolTable.get(i).name + "     " +
symbolTable.get(i).address + "\n");
        }

        // Write Intermediate Code
        bw.write("\nIntermediate Code with LC:\n");
        bw.write("LC\n");
        for (IntermediateCode ic : intermediateCode) {
            bw.write(String.format("%-5d", ic.lc));
            bw.write(String.format("(%-2s, %2d)", ic.operation, ic.code));
            if (ic.reg1 != -1) {
                bw.write(String.format("   (R, %d)", ic.reg1));
            }
            if (ic.reg2 != -1) {
                bw.write(String.format("   (R, %d)", ic.reg2));
            } else if (ic.operandType != null) {
                bw.write(String.format("   (%s, %d)", ic.operandType,
ic.operandValue));
            }
            bw.write("\n");
        }

        // Write Machine Code
        bw.write("\nMachine Code:\n");
        bw.write("LC    Machine Code\n");
        for (MachineCode mc : machineCodeList) {
            bw.write(String.format("%-6d%s\n", mc.lc, mc.binaryCode));
        }

        bw.close();
    }




    static void readIntermediateCodeFromFile(String filename) throws
IOException {
        BufferedReader br = new BufferedReader(new FileReader(filename));
        String line;
        boolean readingSymbolTable = false;
        boolean readingIntermediateCode = false;

        while ((line = br.readLine()) != null) {
            line = line.trim();
            if (line.equals("Symbol Table:")) {
                readingSymbolTable = true;
                continue;
```

```
            } else if (line.equals("Intermediate Code with LC:")) {
                readingSymbolTable  =  false;
                readingIntermediateCode = true;
                br.readLine(); // Skip the "LC" line
                continue;
            }

            if (readingSymbolTable) {
                String[] parts = line.split("\\s+");
                if (parts.length >= 3) {
                    symbolTable.add(new Symbol(parts[1],
Integer.parseInt(parts[2])));
                }
            } else if (readingIntermediateCode) {
                try {
                    String[] parts = line.split("\\s+", 2);
                    if (parts.length >= 2) {
                        int lc = Integer.parseInt(parts[0]);
                        String[] codeParts = parts[1].split("\\)\\s*\\(");

                        for (int i = 0; i < codeParts.length; i++) {
                            codeParts[i] = codeParts[i].replaceAll("[()]",
"").trim();

                        }

                        String operation = codeParts[0].split(",")[0].trim();
                        int code =
Integer.parseInt(codeParts[0].split(",")[1].trim());
                        int reg1 = -1;
                        int reg2 = -1;
                        String operandType = null;
                        int operandValue = 0;

                        if (codeParts.length > 1) {
                            String[] regParts = codeParts[1].split(",");
                            if (regParts[0].trim().equals("R")) {
                                reg1 = Integer.parseInt(regParts[1].trim());
                            } else if (regParts[0].trim().equals("C")) {
                                operandType = regParts[0].trim();
                                operandValue =
Integer.parseInt(regParts[1].trim());
                            }
                        }
                        if (codeParts.length > 2) {
                            String[] operandParts = codeParts[2].split(",");
                            if (operandParts[0].trim().equals("R")) {
                                reg2 =
Integer.parseInt(operandParts[1].trim());
```

```
                            } else {
                                operandType = operandParts[0].trim();
                                operandValue =
Integer.parseInt(operandParts[1].trim());
                            }
                        }

                        intermediateCode.add(new IntermediateCode(operation,
code, reg1, reg2, operandType, operandValue, lc));
                    }
                } catch (NumberFormatException |
ArrayIndexOutOfBoundsException e) {
                    System.out.println("Error parsing line: " + line);
                    e.printStackTrace();
                }
            }
        }
        br.close();
    }


    public static void main(String[] args) {
        String inputFilename = "out.txt";
        String outputFilename = "machine_code.txt";
        try {
            readIntermediateCodeFromFile(inputFilename);
            List<MachineCode> machineCodeList = generateMachineCode();
            writeOutputToFile(outputFilename, machineCodeList);
            System.out.println("Pass-II completed successfully. Output written
to " + outputFilename);
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```