Pass 1

```
import java.io.*;
import java.util.*;

class MOTEntry {
   String mnemonic, type;
   int opcode;
   MOTEntry(String m, String t, int o) {
      mnemonic = m;
      type = t;
      opcode = o;
   }
}

class Symbol {
   String name;
   int address;
   Symbol(String n, int a) {
      name = n;
      address = a;
   }
}

class Literal {
   String literal;
   int address;
   Literal(String l, int a) {
      literal = l;
      address = a;
   }
}
```

```
public class PassOneAssembler {

    static Map<String, MOTEntry> MOT = new HashMap<>();
    static Map<String, Integer> REGTAB = new HashMap<>();
    static Set<String> POT = new HashSet<>();

    static List<Symbol> SYMTAB = new ArrayList<>();
    static List<Literal> LITTAB = new ArrayList<>();
    static int LC = 0;

    public static void main(String[] args) throws Exception {

        System.out.println("Current working directory: " + System.getProperty("user.dir"));
        initializeTables();

        // Ensure input file exists
        File asmFile = new File("input.asm");
        if (!asmFile.exists()) {
            System.out.println("⚠️ input.asm not found! Creating a sample program...");
            PrintWriter pw = new PrintWriter(new FileWriter(asmFile));
            pw.println("START 200");
            pw.println("MOVER AREG, ='5'");
            pw.println("ADD BREG, A");
            pw.println("MOVEM AREG, B");
            pw.println("A DS");
            pw.println("B DS");
            pw.println("END");
            pw.close();
            System.out.println("☑ Default input.asm created. Please run again.");
            return;
```

```
    }

    System.out.println("\n===== PASS - I =====");
    passOne("input.asm");
    displaySymbolAndLiteralTables();
    System.out.println("\n☑ Pass-I completed successfully! Intermediate code generated.");
}

// ---------------------- PASS - I -------------------------
static void passOne(String filename) throws Exception {
    BufferedReader br = new BufferedReader(new FileReader(filename));
    PrintWriter ic = new PrintWriter(new FileWriter("intermediate.txt"));
    String line;

    while ((line = br.readLine()) != null) {
        line = line.trim();
        if (line.equals("")) continue;
        String[] parts = line.split("[ ,]+");

        // START directive
        if (parts[0].equals("START")) {
            LC = Integer.parseInt(parts[1]);
            ic.println("(AD,01) (C," + LC + ")");
            continue;
        }

        // END directive
        if (parts[0].equals("END")) {
            ic.println("(AD,02)");
            break;
        }
```

```
// DS (declare storage)
if (parts.length >= 2 && parts[1].equals("DS")) {
   SYMTAB.add(new Symbol(parts[0], LC));
   ic.println("(DL,01) (C,1)");
   LC++;
   continue;
}

// Instruction with operands
MOTEntry m = MOT.get(parts[0]);
if (m != null) {
   ic.print("(" + m.type + "," + String.format("%02d", m.opcode) + ") ");

   // Register
   if (REGTAB.containsKey(parts[1]))
      ic.print("(R," + REGTAB.get(parts[1]) + ") ");
   else
      ic.print("(R,0) ");

   // Operand (symbol or literal)
   String operand = parts[2];
   if (operand.startsWith("='")) {  // Literal
      boolean exists = false;
      for (Literal l : LITTAB)
         if (l.literal.equals(operand)) exists = true;
      if (!exists)
         LITTAB.add(new Literal(operand, -1));
      ic.println("(L," + (LITTAB.size()) + ")");
   } else { // Symbol
      int idx = getSymbolIndex(operand);
```

```
                ic.println("(S," + idx + ")");
            }
            LC++;
        }
    }

    // Assign addresses to literals
    for (Literal l : LITTAB)
        if (l.address == -1)
            l.address = LC++;

    br.close();
    ic.close();
    System.out.println("☑ Intermediate code written to intermediate.txt");
}

// ---------- UTILITIES ----------
static void initializeTables() {
    MOT.put("MOVER", new MOTEntry("MOVER", "IS", 1));
    MOT.put("MOVEM", new MOTEntry("MOVEM", "IS", 2));
    MOT.put("ADD", new MOTEntry("ADD", "IS", 3));

    REGTAB.put("AREG", 1);
    REGTAB.put("BREG", 2);

    POT.addAll(Arrays.asList("START", "END", "DS"));
    System.out.println("Tables Initialized ☑");
}

static int getSymbolIndex(String name) {
    for (int i = 0; i < SYMTAB.size(); i++)
```

```
            if (SYMTAB.get(i).name.equals(name)) return (i + 1);
        SYMTAB.add(new Symbol(name, -1));
        return SYMTAB.size();
    }

    static void displaySymbolAndLiteralTables() {
        System.out.println("\n--- SYMBOL TABLE ---");
        System.out.println("Name\tAddress");
        for (Symbol s : SYMTAB)
            System.out.println(s.name + "\t" + s.address);

        System.out.println("\n--- LITERAL TABLE ---");
        System.out.println("Literal\tAddress");
        for (Literal l : LITTAB)
            System.out.println(l.literal + "\t" + l.address);
    }
}
```