**🧪 Experiment: Pass 1 of Two Pass Macroprocessor**

**🎯 Aim:**

To design and implement **Pass 1** of a **Two Pass Macroprocessor** using appropriate data structures in **Java** on Linux.

**📌 Objective:**

To construct the **Macro Name Table (MNT)**, **Macro Definition Table (MDT)**, and **Argument List Array (ALA)** using a sample macro definition and call.

**📖 Theory (Summary):**

**1. 📜 What is a Macroprocessor?**

A **macroprocessor** expands **macros** (reusable instruction blocks) defined in assembly programs before assembly starts.

**2. 📦 Data Structures in 2 Pass Macroprocessor:**

| **Structure** | **Description** |
| --- | --- |
| **MNT** | Macro Name Table – stores macro names and pointers to MDT |
| **MDT** | Macro Definition Table – stores macro body instructions |
| **ALA** | Argument List Array – maps formal parameters to actual arguments |

**3. 🔄 Flow of Pass 1:**

1. Detect MACRO and start recording macro.
2. Store macro name and definition in **MNT** and **MDT**.
3. Replace dummy parameters with ALA indices.
4. On MEND, stop recording and finalize entries.
5. Remove macro definitions from the program (output only macro call and rest of the code).

**4. 🧮 Algorithm for Pass 1:**

1. Initialize MNT, MDT, and ALA.
2. Read each line of input:
   * If MACRO is found, process macro:
     + Add macro name to MNT.
     + Store definition in MDT.
     + Update ALA with dummy arguments.
   * If MEND, mark end of macro in MDT.
3. Output:
   * Modified source program (without macro definition)
   * MNT, MDT, and ALA

**📥 Input:**

An Assembly Language Program containing:

* One macro definition
* One macro call
* Few normal assembly instructions

**📤 Output:**

1. **Modified Program** (without macro definition)
2. **MNT (Macro Name Table)**
3. Index | Macro Name | MDT Index
4. **MDT (Macro Definition Table)**
5. Index | Instruction
6. **ALA (Argument List Array)**
7. Index | Dummy Argument

**✅ Conclusion:**

Pass 1 of the macroprocessor identifies macros, creates essential tables (MNT, MDT, ALA), and removes macro definitions for Pass 2 to process macro calls.

**💻 Platform:**

Linux with Java

**🎤 Viva Questions & One-Line Answers**

**🧠 Macro Basics**

1. **What is a macroprocessor?**  
   → A tool that expands macro instructions into actual assembly code before assembly.
2. **What is a macro?**  
   → A named sequence of instructions that can be reused with different arguments.

**⚙️ Macro Components**

1. **What are macro instructions?**  
   → MACRO defines a macro; MEND ends the macro body.
2. **What are dummy arguments in macros?**  
   → Placeholder parameters used inside macros like &ARG1.
3. **What is the role of MNT?**  
   → Stores macro names and index pointers to their definitions in MDT.
4. **What is MDT used for?**  
   → Stores the complete body of the macro instructions.
5. **What does ALA stand for and do?**  
   → Argument List Array – maps dummy arguments to actual parameters.

**🔄 Pass 1 Execution**

1. **What is the job of Pass 1 of macroprocessor?**  
   → It stores macro definitions in MNT, MDT, ALA and removes them from code.
2. **What happens when MACRO is encountered in Pass 1?**  
   → Macro processing starts and instructions are stored in MDT.
3. **What does MEND signify in macro processing?**  
   → It marks the end of the macro definition.

**🪛 Miscellaneous**

1. **Can macro calls be nested?**  
   → Yes, but nested macro definitions are complex and often avoided.
2. **Difference between macro and subroutine?**  
   → Macros are expanded inline; subroutines are called at runtime.
3. **Why is Pass 1 important in macro processing?**  
   → It prepares the groundwork for macro expansion in Pass 2.
4. **What are positional and keyword arguments in macros?**  
   → Positional: based on order; Keyword: based on parameter names.
5. **How are ALA entries accessed in MDT?**  
   → Using index notation like (1), (2) to refer to arguments.

import java.io.\*;

import java.util.\*;

public class macroProcessor1 {

    static class MacroDefinition {

        String name;

        int mdtIndex;

        List<String> definition;

        MacroDefinition(String name, int mdtIndex) {

            this.name = name;

            this.mdtIndex = mdtIndex;

            this.definition = new ArrayList<>();

        }

    }

    private static final Map<String, MacroDefinition> MNT = new LinkedHashMap<>();

    private static final List<String> MDT = new ArrayList<>();

    private static int MDTC = 0;

    public static void pass1(List<String> sourceCode) {

        boolean isMacroDefinition = false;

        MacroDefinition currentMacro = null;

        for (String line : sourceCode) {

            String[] tokens = line.trim().split(" ");

            if (tokens[0].equalsIgnoreCase("MACRO")) {

                isMacroDefinition = true;

            } else if (isMacroDefinition && !tokens[0].equalsIgnoreCase("MEND")) {

                if (currentMacro == null) {

                    currentMacro = new MacroDefinition(tokens[0], MDTC + 1);

                    MNT.put(tokens[0], currentMacro);

                }

                currentMacro.definition.add(line);

            } else if (tokens[0].equalsIgnoreCase("MEND")) {

                if (currentMacro != null) {

                    MDT.addAll(currentMacro.definition);

                    MDT.add("MEND");

                    MDTC += currentMacro.definition.size() + 1;

                    currentMacro = null;

                    isMacroDefinition = false;

                }

            }

        }

    }

    public static List<String> readFromFile(String filename) throws IOException {

        List<String> lines = new ArrayList<>();

        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {

            String line;

            while ((line = br.readLine()) != null) {

                lines.add(line);

            }

        }

        return lines;

    }

    public static void displayTables() {

        System.out.println("\nMacro Name Table (MNT) [Total: " + MNT.size() + "]:");

        System.out.println("Index\tMacro Name\tMDT Index");

        int index = 1;

        for (Map.Entry<String, MacroDefinition> entry : MNT.entrySet()) {

            System.out.println(index + "\t" + entry.getKey() + "\t" + entry.getValue().mdtIndex);

            index++;

        }

        System.out.println("\nMacro Definition Table (MDT) [Total: " + MDTC + "]:");

        for (int i = 0; i < MDT.size(); i++) {

            System.out.println((i + 1) + "\t" + MDT.get(i));

        }

    }

    public static void main(String[] args) {

        try {

            List<String> sourceCode = readFromFile("input.txt");

            pass1(sourceCode);

            displayTables();

        } catch (IOException e) {

            System.err.println("Error reading input file: " + e.getMessage());

        }

    }

}