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

**🎯 Aim:**

To design and implement **Pass II** of a **Two Pass Macroprocessor** using suitable data structures in Java.

**📌 Objective:**

To expand macro calls in the source program using the **MDT, MNT**, and **ALA** generated in Pass I.

**📖 Theory (Summary):**

**🔄 1. Algorithm for Pass II:**

1. Read the input program line by line.
2. If a macro call is found:
   * Search the **MNT** for the macro name.
   * Get the pointer to **MDT** and retrieve macro body.
   * Create a fresh **ALA** and assign actual arguments to dummy parameters.
   * Expand macro line by line using ALA values.
3. If it's a normal instruction, copy it to output.
4. Continue till end of program.

**📦 2. Data Structures Used:**

| **Structure** | **Description** |
| --- | --- |
| **MNT** | Stores macro names and MDT starting index |
| **MDT** | Stores macro body instructions |
| **ALA** | Maps actual arguments to dummy parameters during macro expansion |
| **Expanded Program** | Stores the final assembly language program after macro expansion |

**📊 3. Flowchart for Pass II:**

**Start → Read Source Line → Is Macro Call? → Yes → Lookup MNT → Fetch MDT → Setup ALA → Replace Params → Output Expanded Code → Else → Output Normal Code → Repeat → End**

**📥 Input:**

* Assembly language code **without macro definition**
* **MNT** – Macro Name Table
* **MDT** – Macro Definition Table
* **ALA** – Argument List Array  
  (Generated from **Pass I**)

**📤 Output:**

* Fully **expanded assembly program**, with macro calls replaced by actual instructions.

**✅ Conclusion:**

In this experiment, we studied how **Pass II** of a macroprocessor expands macro calls using previously created tables, completing the macro expansion process.

**💻 Platform:**

Java on Linux/Windows

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

**🧠 Macro Processing Basics**

1. **What is the purpose of Pass II of a macroprocessor?**  
   → To expand macro calls using MDT, MNT, and ALA.
2. **What are the inputs to Pass II?**  
   → Source code with macro calls, MNT, MDT, and ALA.
3. **What is the output of Pass II?**  
   → An assembly language program with all macros fully expanded.
4. **Why is a fresh ALA created for each macro call?**  
   → To map actual arguments to dummy arguments uniquely for each call.

**🔁 Macro Expansion**

1. **What happens when a macro call is found in Pass II?**  
   → It is replaced with corresponding instructions from MDT using ALA.
2. **How are actual arguments passed to the macro?**  
   → They are assigned to dummy parameters via ALA during macro expansion.
3. **How does Pass II handle normal assembly instructions?**  
   → It directly copies them to the output.

**⚙️ Advanced Concepts**

1. **What is a single pass macroprocessor?**  
   → A processor that defines and expands macros in one pass.
2. **Why are two passes preferred in macroprocessing?**  
   → For clear separation of definition and expansion, making it easier to manage.
3. **What is a nested macro?**  
   → A macro that contains a call to another macro within its body.
4. **Can macros call other macros?**  
   → Yes, but this is known as **nested macros** and must be handled carefully.

**🛠️ Examples**

1. **Give an example of a nested macro.**  
   → A macro M1 calling another macro M2 inside its definition.

MACRO

M1 &A

M2 &A

MEND

MACRO

M2 &X

ADD &X, =‘1’

MEND

import java.io.\*;

import java.util.\*;

public class MacroProcessorPass2 {

    private static final Map<String, Integer> MNT = new LinkedHashMap<>(); // Macro Name Table

    private static final List<String> MDT = new ArrayList<>(); // Macro Definition Table

    private static final List<String> ALA = new ArrayList<>(); // Argument List Array

    private static final Map<String, String> ALP = new LinkedHashMap<>(); // Actual Argument List

    public static void loadTables(String mntFile, String mdtFile, String alaFile, String alpFile) throws IOException {

        // Load MNT

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

            String line;

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

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

                if (tokens.length == 2) {

                    MNT.put(tokens[0], Integer.parseInt(tokens[1]));

                }

            }

        }

        // Load MDT

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

            String line;

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

                MDT.add(line);

            }

        }

        // Load ALA

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

            String line;

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

                ALA.add(line.trim());

            }

        }

        // Load ALP

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

            String line;

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

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

                if (tokens.length == 2) {

                    ALP.put(tokens[0], tokens[1]);

                }

            }

        }

    }

    public static List<String> expandMacroCalls() {

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

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

            String macroName = entry.getKey();

            String actualArg = entry.getValue();

            if (MNT.containsKey(macroName)) {

                int index = MNT.get(macroName) - 1; // MDT index (0-based)

                System.out.println("\nExpanding Macro: " + macroName + " (MDT Index: " + (index + 1) + ")");

                if (index >= MDT.size()) {

                    System.out.println("Error: MDT Index out of bounds! Index=" + index + ", MDT Size=" + MDT.size());

                    continue;

                }

                while (index < MDT.size()) {

                    String expandedLine = MDT.get(index);

                    // Stop expansion when MEND is reached

                    if (expandedLine.equalsIgnoreCase("MEND")) {

                        break;

                    }

                    // Replace formal argument with actual argument

                    for (String formalArg : ALA) {

                        expandedLine = expandedLine.replace(formalArg, actualArg);

                    }

                    expandedCode.add(expandedLine);

                    System.out.println("Generated: " + expandedLine);

                    index++;

                }

            } else {

                System.out.println("Error: Macro " + macroName + " not found in MNT.");

            }

        }

        return expandedCode;

    }

    public static void writeToFile(String filename, List<String> lines) throws IOException {

        try (BufferedWriter bw = new BufferedWriter(new FileWriter(filename))) {

            for (String line : lines) {

                bw.write(line);

                bw.newLine();

            }

        }

    }

    public static void main(String[] args) {

        try {

            // Load macro tables from files

            loadTables("mnt.txt", "mdt.txt", "ala.txt", "alp.txt");

            // Perform Pass 2 (macro expansion)

            List<String> expandedCode = expandMacroCalls();

            // Display expanded code

            System.out.println("\nExpanded Macro Code:");

            for (String line : expandedCode) {

                System.out.println(line);

            }

            // Write expanded code to output file

            writeToFile("expanded\_code.txt", expandedCode);

            System.out.println("\nExpanded code saved in 'expanded\_code.txt'");

        } catch (IOException e) {

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

        }

    }

}