**Assignment No - 2**

**Program -**

import java.io.\*;

import java.util.\*;

class MacroProcessor {

static class MacroEntry {

int mdtIndex;

List<String> params;

Map<String, String> defaults;

MacroEntry(int mdtIndex, List<String> params, Map<String, String> defaults) {

this.mdtIndex = mdtIndex;

this.params = params;

this.defaults = defaults;

}

}

private Map<String, MacroEntry> MNT = new LinkedHashMap<>();

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

private List<String> intermediateCode = new ArrayList<>();

private MacroEntry parseMacroHeader(String line, int mdtIndex) {

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

String macroName = parts[0];

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

Map<String, String> defaults = new LinkedHashMap<>();

if (parts.length > 1) {

String paramStr = line.substring(macroName.length()).replace(" ", "");

String[] paramList = paramStr.split(",");

for (String p : paramList) {

if (p.contains("=")) {

String[] kv = p.split("=");

String paramName = kv[0];

String defaultVal = (kv.length > 1 && !kv[1].isEmpty()) ? kv[1] : null;

params.add(paramName);

defaults.put(paramName, defaultVal);

} else {

params.add(p);

}

}

}

return new MacroEntry(mdtIndex, params, defaults);

}

public void pass1(List<String> sourceLines) {

boolean inMacro = false;

String currentMacro = null;

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

Map<String, String> defaultParams = new LinkedHashMap<>();

int mdtIndex = 0;

for (String rawLine : sourceLines) {

String line = rawLine.trim();

if (line.equals("MACRO")) {

inMacro = true;

continue;

}

if (inMacro) {

if (currentMacro == null) {

MacroEntry entry = parseMacroHeader(line, mdtIndex);

currentMacro = line.split("\\s+")[0];

paramList = entry.params;

defaultParams = entry.defaults;

MNT.put(currentMacro, entry);

} else {

String processedLine = line;

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

processedLine = processedLine.replace(paramList.get(i), "(P," + i + ")");

}

MDT.add(processedLine);

mdtIndex++;

if (processedLine.equals("MEND")) {

inMacro = false;

currentMacro = null;

paramList.clear();

defaultParams.clear();

}

}

} else {

intermediateCode.add(line);

}

}

}

public Pair<List<String>, List<Map<String, Map<String, String>>>> pass2() {

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

List<Map<String, Map<String, String>>> allALAs = new ArrayList<>();

for (String line : intermediateCode) {

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

if (parts.length == 0) {

output.add(line);

continue;

}

String macroName = parts[0];

if (MNT.containsKey(macroName)) {

MacroEntry entry = MNT.get(macroName);

List<String> formalParams = entry.params;

Map<String, String> defaults = entry.defaults;

String actualParamsStr = line.substring(macroName.length()).trim();

Map<String, String> ALA = new LinkedHashMap<>();

if (!actualParamsStr.isEmpty()) {

String[] tokens = actualParamsStr.split(",");

int posIndex = 0;

for (String tok : tokens) {

tok = tok.trim();

if (tok.contains("=")) {

String[] kv = tok.split("=");

String key = kv[0].trim();

String val = kv[1].trim();

if (!key.startsWith("&")) key = "&" + key;

ALA.put(key, val);

} else {

if (posIndex < formalParams.size()) {

ALA.put(formalParams.get(posIndex), tok);

posIndex++;

}

}

}

}

for (String p : formalParams) {

if (!ALA.containsKey(p)) {

ALA.put(p, defaults.getOrDefault(p, ""));

}

}

Map<String, Map<String, String>> alaEntry = new LinkedHashMap<>();

alaEntry.put(macroName, new LinkedHashMap<>(ALA));

allALAs.add(alaEntry);

int i = entry.mdtIndex;

while (i < MDT.size()) {

String mdtLine = MDT.get(i);

if (mdtLine.equals("MEND")) break;

String expandedLine = mdtLine;

for (int idx = 0; idx < formalParams.size(); idx++) {

expandedLine = expandedLine.replace("(P," + idx + ")", ALA.get(formalParams.get(idx)));

}

output.add(expandedLine);

i++;

}

} else {

output.add(line);

}

}

return new Pair<>(output, allALAs);

}

private void writeToFile(String filename, String content) {

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

writer.write(content);

} catch (IOException e) {

e.printStackTrace();

}

}

public void generateFiles(List<String> expandedCode, List<Map<String, Map<String, String>>> allALAs) {

StringBuilder mntContent = new StringBuilder("=== MNT (Macro Name Table) ===\n");

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

mntContent.append(entry.getKey()).append(":\n");

mntContent.append(" MDT index: ").append(entry.getValue().mdtIndex).append("\n");

mntContent.append(" Params: ").append(entry.getValue().params.toString()).append("\n");

mntContent.append(" Defaults: {");

boolean first = true;

for (Map.Entry<String, String> def : entry.getValue().defaults.entrySet()) {

if (!first) mntContent.append(", ");

mntContent.append("'").append(def.getKey()).append("': ");

mntContent.append(def.getValue() == null ? "None" : "'" + def.getValue() + "'");

first = false;

}

mntContent.append("}\n");

}

writeToFile("mnt.txt", mntContent.toString());

StringBuilder mdtContent = new StringBuilder("=== MDT (Macro Definition Table) ===\n");

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

mdtContent.append(i).append(": ").append(MDT.get(i)).append("\n");

}

writeToFile("mdt.txt", mdtContent.toString());

StringBuilder icContent = new StringBuilder("=== Intermediate Code (Pass-I Output) ===\n");

for (String line : intermediateCode) {

icContent.append(line).append("\n");

}

writeToFile("intermediate\_code.txt", icContent.toString());

StringBuilder alaContent = new StringBuilder("=== ALAs for each Macro Invocation ===\n");

int idx = 1;

for (Map<String, Map<String, String>> ala : allALAs) {

alaContent.append("Invocation ").append(idx++).append(": ").append(ala).append("\n");

}

writeToFile("alas.txt", alaContent.toString());

StringBuilder expContent = new StringBuilder("=== Expanded Code (Pass-II Output) ===\n");

for (String line : expandedCode) {

expContent.append(line).append("\n");

}

writeToFile("expanded\_code.txt", expContent.toString());

}

public static void main(String[] args) {

List<String> sourceProgram = Arrays.asList(

"MACRO",

"INCR\_D &MEM\_VAL, &INCR\_VAL=, &REG=AREG",

"MOVER &REG, &MEM\_VAL",

"ADD &REG, &INCR\_VAL",

"MOVEM &REG, &MEM\_VAL",

"MEND",

"START 200",

"DS A 1",

"DC B 1",

"INCR\_D A, INCR\_VAL=B, REG=BREG",

"INCR\_D A, INCR\_VAL=B",

"PRINT A",

"END 202"

);

MacroProcessor mp = new MacroProcessor();

mp.pass1(sourceProgram);

Pair<List<String>, List<Map<String, Map<String, String>>>> result = mp.pass2();

List<String> expandedCode = result.first;

List<Map<String, Map<String, String>>> allALAs = result.second;

mp.generateFiles(expandedCode, allALAs);

}

}

class Pair<F, S> {

public final F first;

public final S second;

public Pair(F first, S second) {

this.first = first;

this.second = second;

}

}

**Output-**

=== MDT (Macro Definition Table) ===

0: MOVER (P,2), (P,0)

1: ADD (P,2), (P,1)

2: MOVEM (P,2), (P,0)

3: MEND

=== MNT (Macro Name Table) ===

INCR\_D:

MDT index: 0

Params: ['&MEM\_VAL', '&INCR\_VAL', '&REG']

Defaults: {'&INCR\_VAL': None, '&REG': 'AREG'}

=== Intermediate Code (Pass-I Output) ===

START 200

DS A 1

DC B 1

INCR\_D A, INCR\_VAL=B, REG=BREG

INCR\_D A, INCR\_VAL=B

PRINT A

END 202

=== Expanded Code (Pass-II Output) ===

START 200

DS A 1

DC B 1

MOVER BREG, A

ADD BREG, B

MOVEM BREG, A

MOVER AREG, A

ADD AREG, B

MOVEM AREG, A

PRINT A

END 202