**2 Pass Macro-Processor**

**Code :**

import java.util.\*;

import java.io.\*;

class MntTuple

{ String name;

int index;

MntTuple(String s, int i) {

name = s;

index = i; }

public String toString() {

return("[" + name + ", " + index + "]"); } }

class MacroProcessor {

static List<MntTuple> mnt;

static List<String> mdt;

static int mntc;

static int mdtc;

static int mdtp;

static BufferedReader input;

static List<List <String>> ala;

static Map<String, Integer> ala\_macro\_binding;

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

initializeTables();

System.out.println("===== PASS 1 =====\n");

pass1();

System.out.println("\n===== PASS 2 =====\n");

pass2(); }

static void pass1() throws Exception {

String s = new String();

input = new BufferedReader(new InputStreamReader(new FileInputStream("input.txt")));

PrintWriter output = new PrintWriter(new FileOutputStream("output\_pass1.txt"), true);

while((s = input.readLine()) != null) {

if(s.equalsIgnoreCase("MACRO")) {

processMacroDefinition();

} else {

output.println(s); } }

System.out.println("ALA:");

showAla(1);

System.out.println("\nMNT:");

showMnt();

System.out.println("\nMDT:");

showMdt(); }

static void processMacroDefinition() throws Exception {

String s = input.readLine();

String macro\_name = s.substring(0, s.indexOf(" "));

mnt.add(new MntTuple(macro\_name, mdtc));

mntc++;

pass1Ala(s);

StringTokenizer st = new StringTokenizer(s, " ,", false);

String x = st.nextToken();

for(int i=x.length() ; i<12 ; i++) {

x += " "; }

String token = new String();

int index;

token = st.nextToken();

x += token;

while(st.hasMoreTokens()) {

token = st.nextToken();

x += "," + token; }

mdt.add(x);

mdtc++;

addIntoMdt(ala.size()-1); }

static void pass1Ala(String s) {

StringTokenizer st = new StringTokenizer(s, " ,", false);

String macro\_name = st.nextToken();

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

int index;

while(st.hasMoreTokens()) {

String x = st.nextToken();

if((index = x.indexOf("=")) != -1) {

x = x.substring(0, index); }

l.add(x); }

ala.add(l);

ala\_macro\_binding.put(macro\_name, ala\_macro\_binding.size()); }

static void addIntoMdt(int ala\_number) throws Exception {

String temp = new String();

String s = new String();

List l = ala.get(ala\_number);

boolean isFirst;

while(!s.equalsIgnoreCase("MEND")) {

isFirst = true;

s = input.readLine();

String line = new String();

StringTokenizer st = new StringTokenizer(s, " ,", false);

temp = st.nextToken();

for(int i=temp.length() ; i<12 ; i++) {

temp += " "; }

line += temp;

while(st.hasMoreTokens()) {

temp = st.nextToken();

if(temp.startsWith("&")) {

int x = l.indexOf(temp);

temp = ",#" + x;

isFirst = false;

} else if(!isFirst) {

temp = "," + temp; }

line += temp; }

mdt.add(line);

mdtc++; } }

static void showAla(int pass) throws Exception {

PrintWriter out = new PrintWriter(new FileOutputStream("out\_ala\_pass" + pass + ".txt"), true);

for(List l : ala) {

System.out.println(l);

out.println(l); } }

static void showMnt() throws Exception {

PrintWriter out = new PrintWriter(new FileOutputStream("out\_mnt.txt"), true);

for(MntTuple l : mnt) {

System.out.println(l);

out.println(l); } }

static void showMdt() throws Exception {

PrintWriter out = new PrintWriter(new FileOutputStream("out\_mdt.txt"), true);

for(String l : mdt) {

System.out.println(l);

out.println(l); } }

static void pass2() throws Exception {

input = new BufferedReader(new InputStreamReader(new FileInputStream("output\_pass1.txt")));

PrintWriter output = new PrintWriter(new FileOutputStream("output\_pass2.txt"), true);

String token = new String();

String s;

while((s = input.readLine()) != null) {

StringTokenizer st = new StringTokenizer(s, " ", false);

while(st.hasMoreTokens()) {

token = st.nextToken();

if(st.countTokens() > 2) {

token = st.nextToken(); }

MntTuple x = null;

for(MntTuple m : mnt) {

if(m.name.equalsIgnoreCase(token)) {

x = m;

break; } }

if(x != null) {

mdtp = x.index;

List<String> l = pass2Ala(s);

mdtp++;

String temp = new String();

while(!(temp = mdt.get(mdtp)).trim().equalsIgnoreCase("MEND")) {

String line = new String();

StringTokenizer st2 = new StringTokenizer(temp, " ,",false);

for(int i=0 ; i<12 ; i++) {

line += " "; }

String opcode = st2.nextToken();

line += opcode;

for(int i=opcode.length() ; i<24 ; i++) {

line += " "; }

line += st2.nextToken();

while(st2.hasMoreTokens()) {

String token2 = st2.nextToken();

int index;

if((index = token2.indexOf("#")) != -1) {

line += "," + l.get(Integer.parseInt(token2.substring(index+1,index+2))); } }

mdtp++;

output.println(line);

System.out.println(line); }

break;

} else { output.println(s);

System.out.println(s);

break; } } }

System.out.println("\nALA:");

showAla(2); }

static List<String> pass2Ala(String s) {

StringTokenizer st = new StringTokenizer(s, " ", false);

int num\_tokens = st.countTokens();

String macro\_name = st.nextToken();

int ala\_no = ala\_macro\_binding.get(macro\_name);

List<String> l = ala.get(ala\_no);

int ctr = 0;

StringTokenizer st2 = null;

try { st2 = new StringTokenizer(st.nextToken(), ",", false);

while(st2.hasMoreTokens()) {

l.set(ctr, st2.nextToken());

ctr++; }

} catch(Exception e) {}

if(ctr < num\_tokens) {

String s2 = mdt.get(mdtp);

StringTokenizer st3 = new StringTokenizer(s2, " ,", false);

String token = new String();

int index = 0;

while(st3.hasMoreTokens()) {

token = st3.nextToken();

if((index = token.indexOf("=")) != -1) {

try { l.set(ctr++, token.substring(index+1, token.length()));

} catch(Exception e) {} } } }

ala.set(ala\_no, l);

return l; }

static void initializeTables() {

mnt = new LinkedList<>();

mdt = new ArrayList<>();

ala = new LinkedList<>();

mntc = 0;

mdtc = 0;

ala\_macro\_binding = new HashMap<>(); } }

**Text File :**

MACRO

INCR1 &FIRST,&SECOND=DATA9

A 1,&FIRST

L 2,&SECOND

MEND

MACRO

INCR2 &ARG1,&ARG2=DATA5

L 3,&ARG1

ST 4,&ARG2

MEND

PRG2 START

USING \*,BASE

INCR1 DATA1

INCR2 DATA3,DATA4

FOUR DC F'4'

FIVE DC F'5'

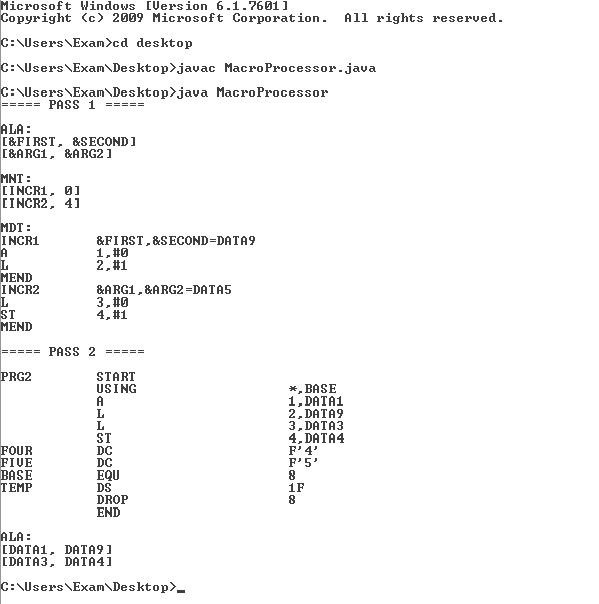
BASE EQU 8

TEMP DS 1F

DROP 8

END

**Output :**

****