**MACRO**

package ketan;

import java.util.\*;

import java.io.\*;

public class MACRO {

    static List<String> MDT;

    static Map<String,String> MNT;

    static int mntPtr,mdtPtr;

    static Map<String,String> ALA;

    public static void main(String[] args) {

        try {

            pass1();

        }catch(Exception ex) {

            ex.printStackTrace();

        }

    }

    static void pass1() throws Exception {

        // Initialize data structures

        MDT = new ArrayList<>();

        MNT = new LinkedHashMap<>();

        ALA = new HashMap<>();

        mntPtr = 0;

        mdtPtr = 0;

        BufferedReader input = new BufferedReader(new InputStreamReader(new FileInputStream("src/lp\_practical4/input.txt")));

        PrintWriter out\_pass1 = new PrintWriter(new FileWriter("src/lp\_practical4/output\_pass1.txt"), true);

        PrintWriter out\_mnt = new PrintWriter(new FileWriter("src/lp\_practical4/MNT.txt"), true);

        PrintWriter out\_mdt = new PrintWriter(new FileWriter("src/lp\_practical4/MDT.txt"), true);

        out\_pass1.println("Ketan");

        String s;

        boolean processingMacroDefinition = false;

        boolean processMacroName = false;

        System.out.println("Name:Ketan Devraj || Roll no: TACO22122");

        System.out.println("======Pass1 output=======");

        // Read from input file one line at a time

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

            // Separate out tokens for each line

            String s\_arr[] = tokenizeString(s, " ");

            // Ensure that the token array has at least one element

            if (s\_arr.length == 0) {

                continue; // Skip empty lines

            }

            String curToken = s\_arr[0];

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

                processingMacroDefinition = true;

                processMacroName = true;

            } else if (processingMacroDefinition) {

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

                    MDT.add(mdtPtr++, s);

                    processingMacroDefinition = false;

                    continue;

                }

                if (processMacroName) {

                    if (s\_arr.length > 1) { // Ensure there's a second token for arguments

                        MNT.put(curToken, mdtPtr + "");

                        mntPtr++;

                        processMacroName = false;

                        processArgumentList(s\_arr[1]);

                        MDT.add(mdtPtr, s);

                        mdtPtr++;

                    }

                    continue;

                }

                if (s\_arr.length > 1) { // Ensure there's a second token for arguments

                    String indexedArgList = processArguments(s\_arr[1]);

                    MDT.add(mdtPtr++, curToken + " " + indexedArgList);

                } else {

                    MDT.add(mdtPtr++, curToken);

                }

            } else {

                // If line is not part of a macro definition, print it as is

                System.out.println(s);

                out\_pass1.println(s);

            }

        }

        input.close();

        // Print MNT

        System.out.println("========MNT========");

        for (String key : MNT.keySet()) {

            String mntRow = key + " " + MNT.get(key);

            System.out.println(mntRow);

            out\_mnt.println(mntRow);

        }

        // Print MDT

        System.out.println("==========MDT=========");

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

            String mdtRow = i + " " + MDT.get(i);

            System.out.println(mdtRow);

            out\_mdt.println(mdtRow);

        }

        out\_pass1.close();

        out\_mnt.close();

        out\_mdt.close();

    }

    static void processArgumentList(String argList) {

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

        //for each macro definition,remove contents of HashMap

        //which are arguments from previous macro definition

        ALA.clear();

        int argCount = st.countTokens();

        //put all arguments for current macro definition in the HashMap

        //with arguments as key and argument index as value

        String curArg;

        for(int i=1; i<=argCount ;i++) {

            curArg = st.nextToken();

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

                curArg = curArg.substring(0,curArg.indexOf("="));

            }

            ALA.put(curArg, "#"+i);

        }

    }

    static String processArguments(String argList) {

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

        int argCount = st.countTokens();

        String curArg, argIndexed;

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

            curArg = st.nextToken();

            argIndexed = ALA.get(curArg);

            // Check if argIndexed is null before calling replaceAll

            if (argIndexed != null) {

                argList = argList.replaceAll(curArg, argIndexed);

            } else {

                // Handle the case where the argument is not found in ALA (optional)

                System.out.println("Warning: Argument " + curArg + " not found in ALA");

            }

        }

        return argList;

    }

    static String[] tokenizeString(String str,String separator) {

        StringTokenizer st = new StringTokenizer(str,separator,false);

        //construct any array of the separated tokens

        String s\_arr[]= new String[st.countTokens()];

        for(int i =0; i<s\_arr.length ; i++) {

            s\_arr[i] = st.nextToken();

        }

        return s\_arr;

    }

}

**INPUT**

MACRO

INCR &X,&Y,&REG=AREG

MOVER &REG,&X

ADD &REG,&Y

MOVEM &REG,&X

MEND

MACRO

DECR &A,&B,&REG=BREG

MOVER &REG,&A

SUB &REG,&B

MOVEM &REG,&A

MEND

START 100

READ N1

READ N2

INCR N1,N2,REG=CREG

DECR NA,N2

STOP

N1 DS 1

N2 DS 1

END

**MDT**

0 INCR &X,&Y,&REG=AREG

1 MOVER #3,#1

2 ADD #3,#2

3 MOVEM #3,#1

4 MEND

5 DECR &A,&B,&REG=BREG

6 MOVER #3,#1

7 SUB #3,#2

8 MOVEM #3,#1

9 MEND

**MNT**

INCR 0

DECR 5

**OUTPUT**



