CODE:

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.StringTokenizer;

class MntTuple { //INITIALIZATION OF MNT TUPLE (Consist of MNT Index, Macro Name, MDT Index)

    int mnti;
    String name;
    int index;

    MntTuple(int mti, String s, int i) {
        mnti = mti;
        name = s;
        index = i;
    }

    public String toString() {
        return (mnti + " " + name + ", " + index + "");
    }
}
```

```
public class MacroPass1 {

    static List<MntTuple> mnt; //MNT List
    static List<String> mdt; //MDT List
    static int mntc; //Initialized to 1
    static int mdtc; //Initialized to 1
    static int mdtp; //used in Pass 2
    static BufferedReader input; //reading Files
    static List<List<String>> ala; //Prepare Argument List Array
    static Map<String, Integer> ala_macro_binding; //used for binding ALA

    public static void main(String args[]) throws Exception {
        initializeTables(); //Initializing everything
        System.out.println("===== PASS 1 =====\n");
        pass1();

    }

    static void pass1() throws Exception {
        String s = new String(); //to be used ahead as line in a code
        input = new BufferedReader(new InputStreamReader(new
FileInputStream("C:\\Users\\hp\\eclipse-workspace\\MACRO PASS 1\\input\\input.txt"))); //reading input file

        PrintWriter output = new PrintWriter(new FileOutputStream("C:\\Users\\hp\\eclipse-workspace\\MACRO PASS 1\\output\\output.txt"), true); //writing into this file
        while ((s = input.readLine()) != null) { //while the code ends
            if (s.equalsIgnoreCase("MACRO")) { //If we get MACRO in code
                processMacroDefinition(); //go for macro processing
            } else {
                output.println(s); //otherwise, print line as it is in file
```

```
        }
    }
    System.out.println("ALA:"); //print ALA for pass 1
    showAla(1); //pass 1 ALA
    System.out.println("\nMNT:"); //print MNT for pass 1
    showMnt();
    System.out.println("\nMDT:"); //print MDT for pass 1
    showMdt();
  }

  static void initializeTables() {
    mnt = new LinkedList<>();
    mdt = new ArrayList<>();
    ala = new LinkedList<>();
    mntc = 1;
    mdtc = 1;
    ala_macro_binding = new HashMap<>();
  }
  static void showAla(int pass) throws Exception {
            PrintWriter out = new PrintWriter(new
FileOutputStream("C:/Users/hp/eclipse-workspace/MACRO PASS 1/output" + pass + ".txt"),
true); //write in this file
            for(List l : ala) { //till all Arguments reached
                System.out.println(l); //print
                out.println(l); //write to file
            }
        }

        static void showMnt() throws Exception {
            PrintWriter out = new PrintWriter(new
FileOutputStream("C:\\Users\\hp\\eclipse-workspace\\MACRO PASS
1\\output\\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("C:\\Users\\hp\\eclipse-workspace\\MACRO PASS
1\\output\\out_mdt.txt"), true);
		for(String l : mdt) {
			System.out.println(l);
			out.println(l);
		}
	}
	static void processMacroDefinition() throws Exception {
		String s = input.readLine(); //reading line of code
		String macro_name = s.substring(0, s.indexOf(" ")); //reading
MACRO_NAME
		mnt.add(new MntTuple(mntc, macro_name, mdtc)); //make entry in MNT
		mntc++; //increment MNT Counter/Index
		pass1Ala(s); //call to ALA of pass 1
		StringTokenizer st = new StringTokenizer(s, " ,", false); //convert next line
into tokens for MDT
		String x = st.nextToken(); //read next token in x
		for(int i=x.length() ; i<12 ; i++) { //max 12 characters allowed in token
			x += " ";
		}
		String token = new String(); //to be used to store tokens in MDT
		int index;
		token = st.nextToken();
		x += token; //appending all tokens in a line MDT
```

```
                while(st.hasMoreTokens()) { //read until all tokens reached
                        token = st.nextToken();
                        x += "," + token;
                }
                mdt.add(x); //add x into mdt
                mdtc++; //increment MDT Counter
                addIntoMdt(ala.size()-1); //add all ALA into MDT
        }
        static void addIntoMdt(int ala_number) throws Exception {
                String temp = new String(); //to be used
                String s = new String(); //to be used
                List l = ala.get(ala_number); //add all ALA in List l
                boolean isFirst; //to be used
                while(!s.equalsIgnoreCase("MEND")) { //until MEND is reached
                        isFirst = true; //keep this true
                        s = input.readLine(); //read all MACRO Lines/Instructions
                        String line = new String(); //just initialized
                        StringTokenizer st = new StringTokenizer(s, " ,", false); //convert line
into tokens
                        temp = st.nextToken(); //keep next token in temp
                        for(int i=temp.length() ; i<12 ; i++) { //check for instruction length
                                temp += " ";
                        }
                        line += temp; //append temp into line
                        while(st.hasMoreTokens()) {
                                temp = st.nextToken(); //read tokens
                                if(temp.startsWith("&")) { //check if it is argument
                                        int x = l.indexOf(temp);
                                        temp = ",#" + x; //reformatting
                                        isFirst = false; //now make it false as it is last keyword
in an instruction
```

```
                        } else if(!isFirst) { //if not argument then
                                temp = "," + temp; //keep adding into temp
                        }
                        line += temp; //append again
                }
                mdt.add(line); //finally add line into MDT
                mdtc++; //increment MDTC
        }
    }
    static void pass1Ala(String s) {
        StringTokenizer st = new StringTokenizer(s, " ,", false); //converting line into
words
        String macro_name = st.nextToken(); //Macro Name stored
        List<String> l = new ArrayList<>(); //ArrayList for adding ALA in one Line
        int index; //used as index for tokens
        while(st.hasMoreTokens()) { //till all tokens are covered
            String x = st.nextToken(); //reading next tokens in x
            if((index = x.indexOf("=")) != -1) { //if parameter is like this
(&ARG=DATA1)
                x = x.substring(0, index); //then take only part before '=' as an
Argument
            }
            l.add(x); //finally add all arguments into l i.e. in one line
        }
        ala.add(l); //pass to ala
        ala_macro_binding.put(macro_name, ala_macro_binding.size()); //store all
arguments under one MACRO NAME
    }
}
```

OUTPUT:

Input:-

```
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
```

ALA:-

[&FIRST, &SECOND]

[&ARG1, &ARG2]

MDT:-

```
INCR1       &FIRST,&SECOND=DATA9
A           1,#0
L           2,#1
MEND
INCR2       &ARG1,&ARG2=DATA5
L           3,#0
ST          4,#1
MEND
```

MNT:-

```
1 INCR1, 1
2 INCR2, 5
```

Output:-

```
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
```