**LPCC ASSIGNMENT-1(WEEK-3)**

****

**SUBMITTED TO:**

**PROF. PALLAVI REGE**

**VISHWAKARMA INSTITUTE OF INFORMATION TECHNOLOGY, PUNE**

**COMPUTER ENGINEERING DEPARTMENT**

**BY:**

**NAME: ABHISHEK MORE**

**G.R No.: 21810033**

**ROLL NO.: 323036**

**CLASS: T.Y COMP**

**BATCH: COMP C2**

**LANGUAGE USED: JAVA**

**ASSIGNMENT-1 (WEEK-3)**

**AIM:**

To accept the input (Machine Code) from a file line-by-line and display its:-

1. Intermediate Code
2. Symbol Table
3. Literal Table
4. Pool Table

**INPUT CODE:**

START 200

MOVER AREG,='5'

MOVEM AREG,X

L1 MOVER BREG,='3'

ORIGIN L1 + 3

LTORG

NEXT ADD AREG,='1'

SUB BREG,='2'

BC LT,BACK

LTORG

BACK EQU L1

ORIGIN NEXT + 5

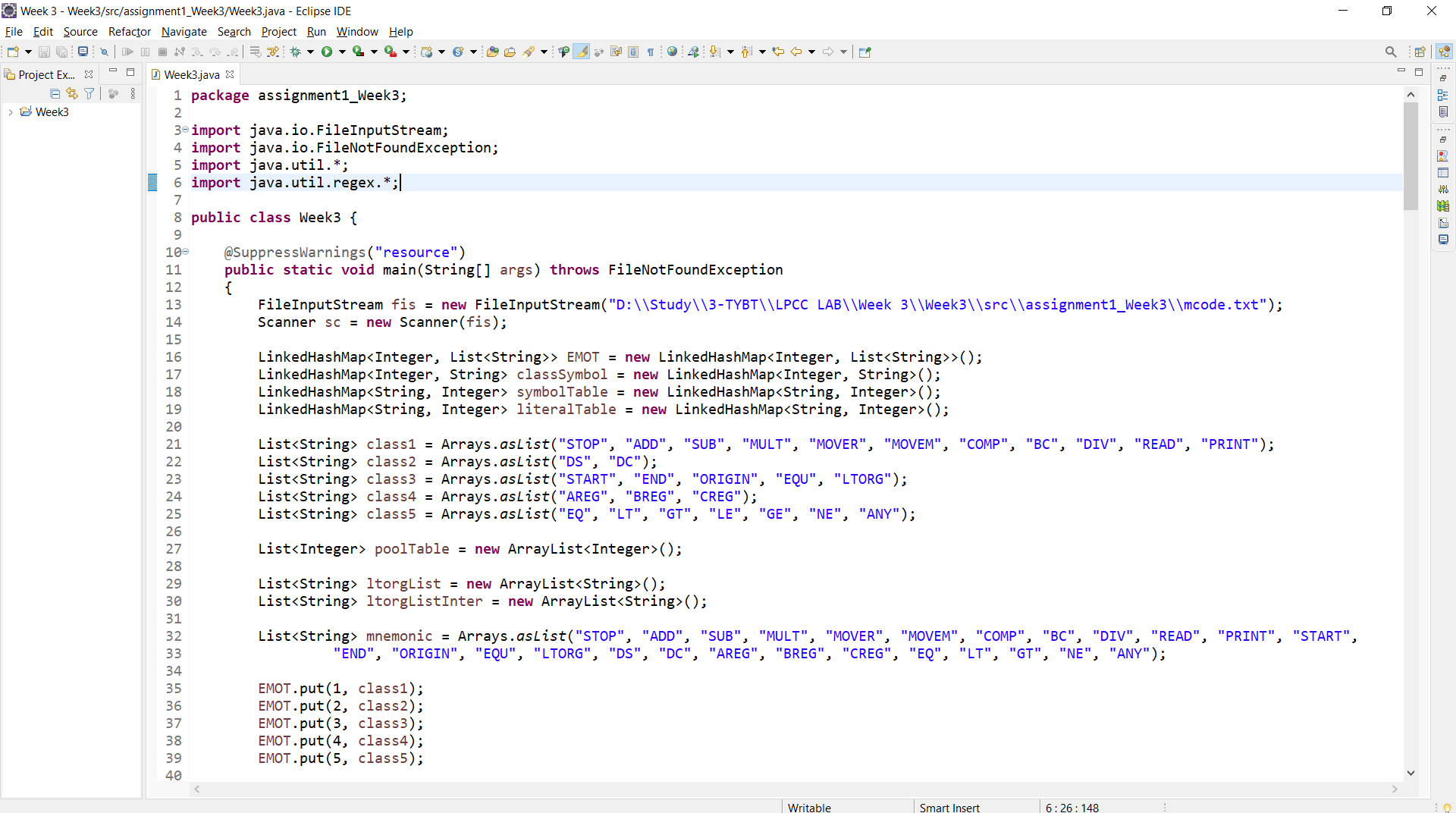
MULT CREG,='4'

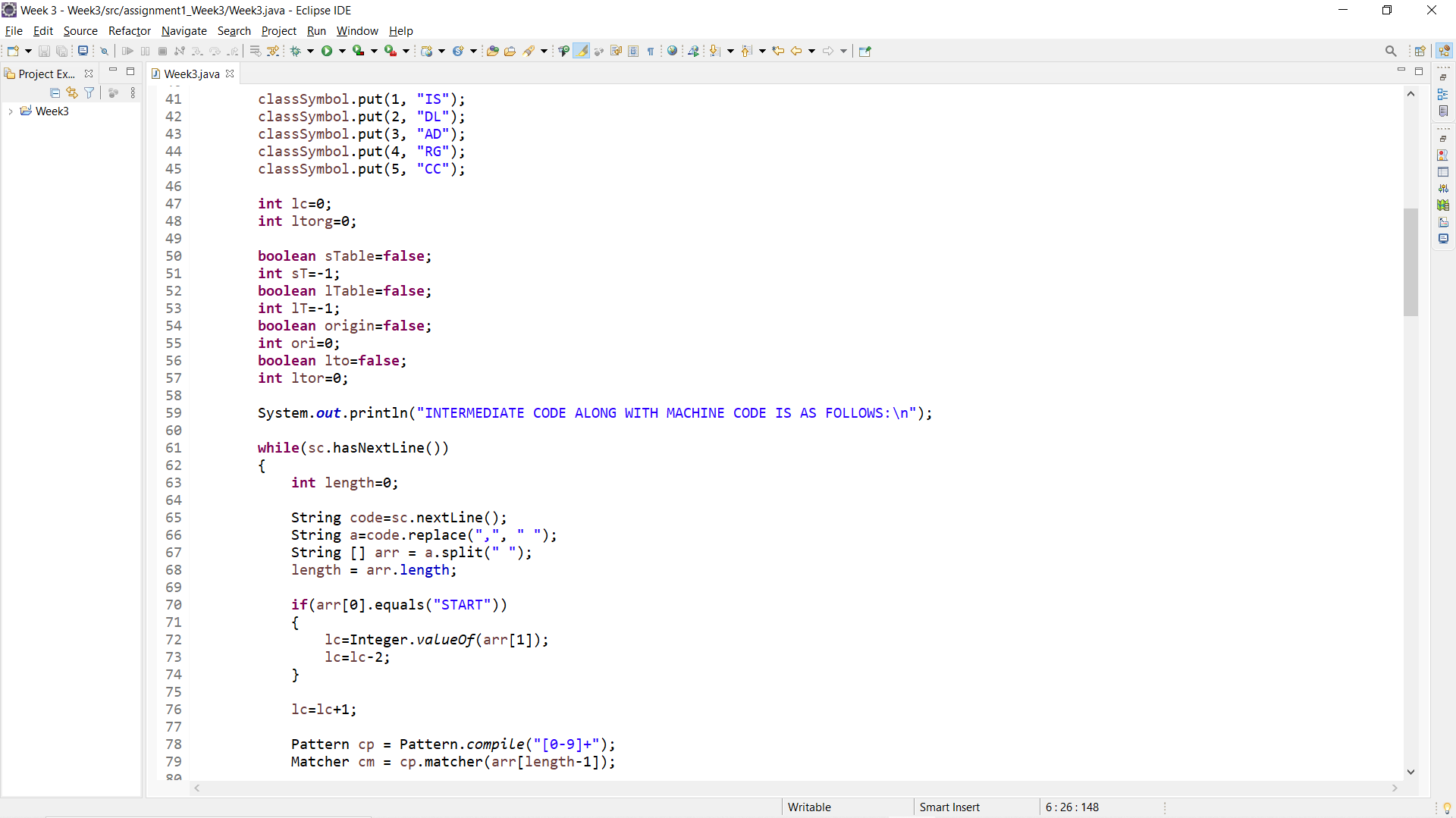
STOP

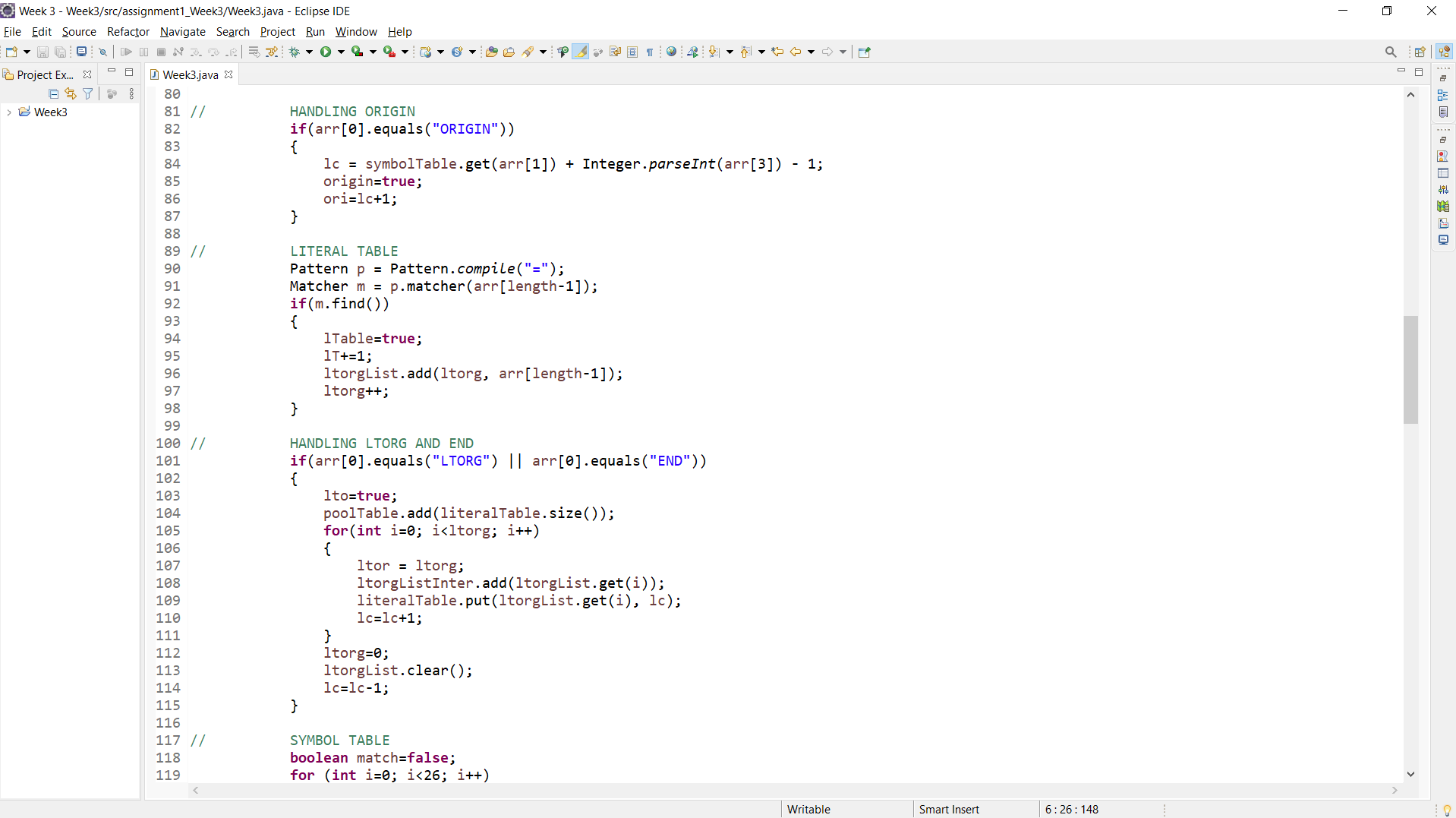
X DS 1

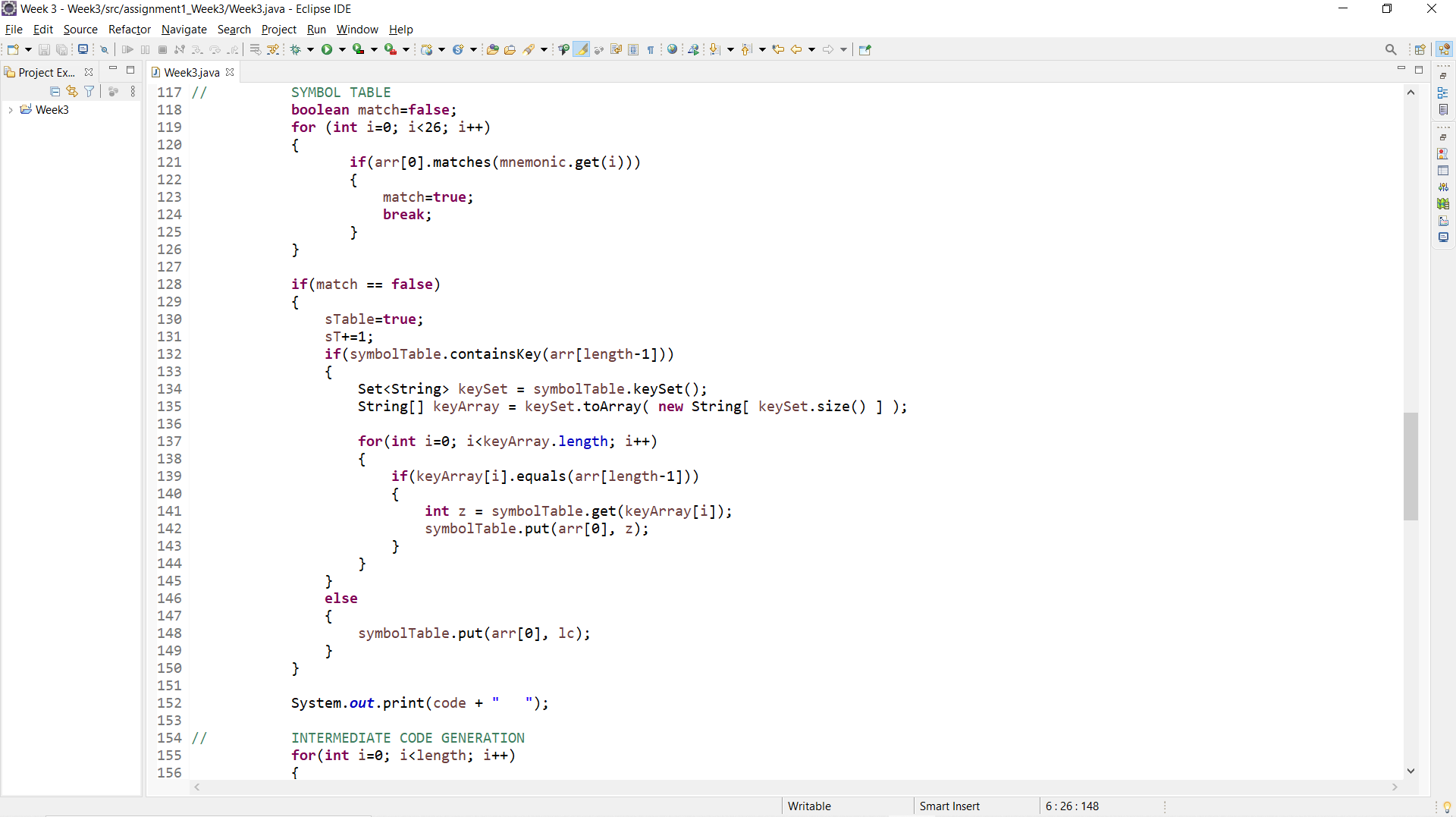
END

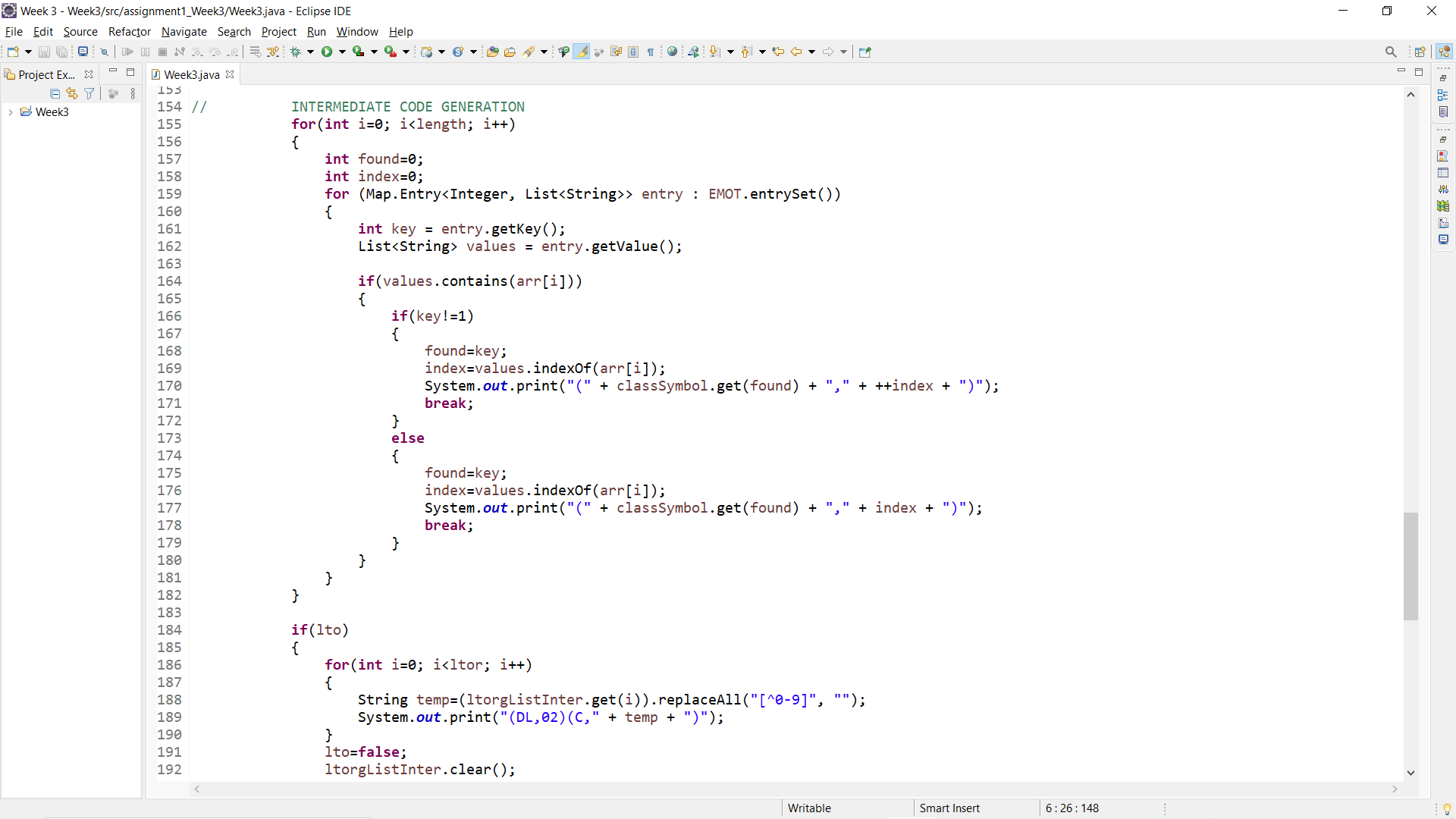
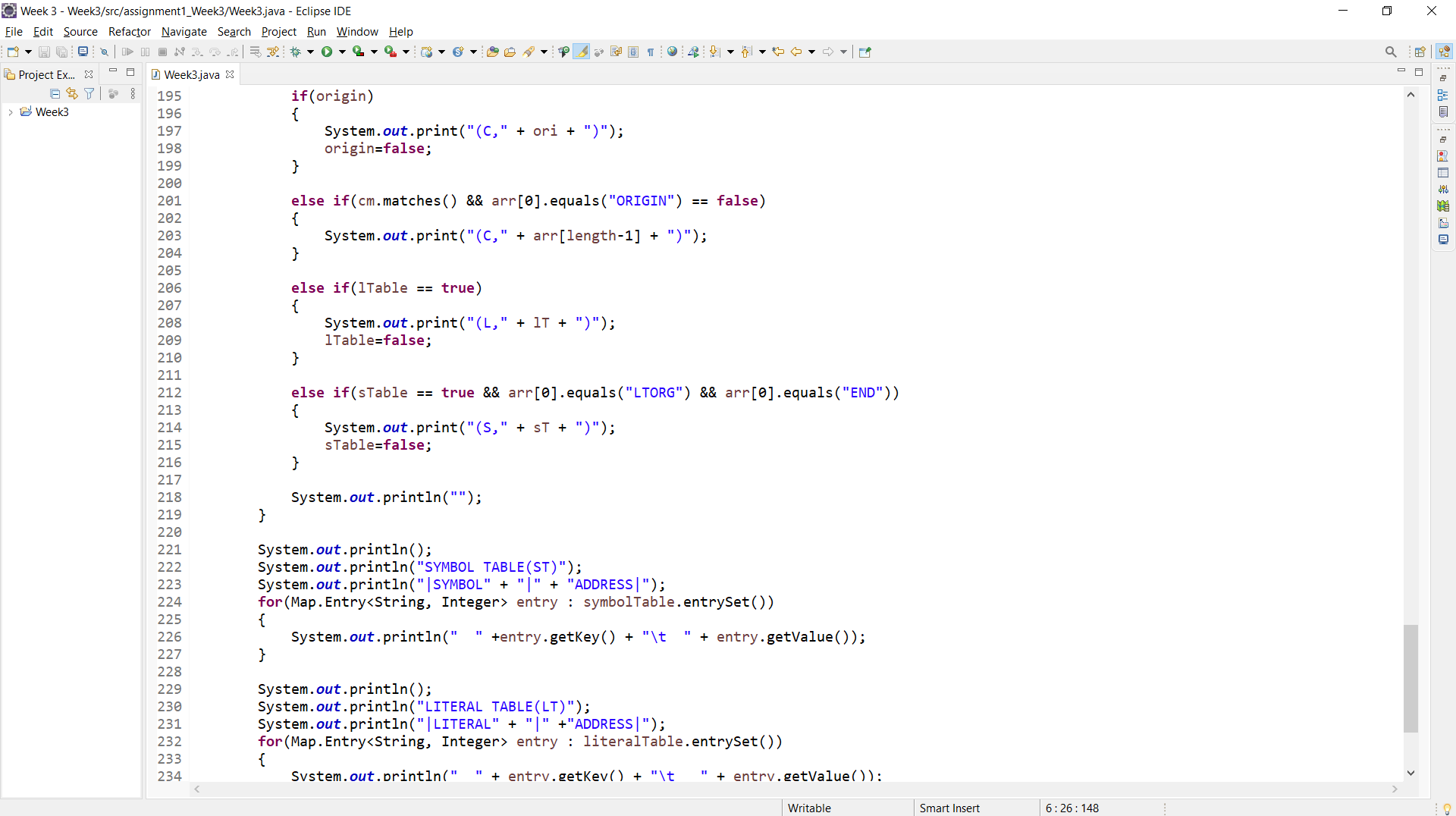
**SOURCE CODE (SCREENSHOTS):**

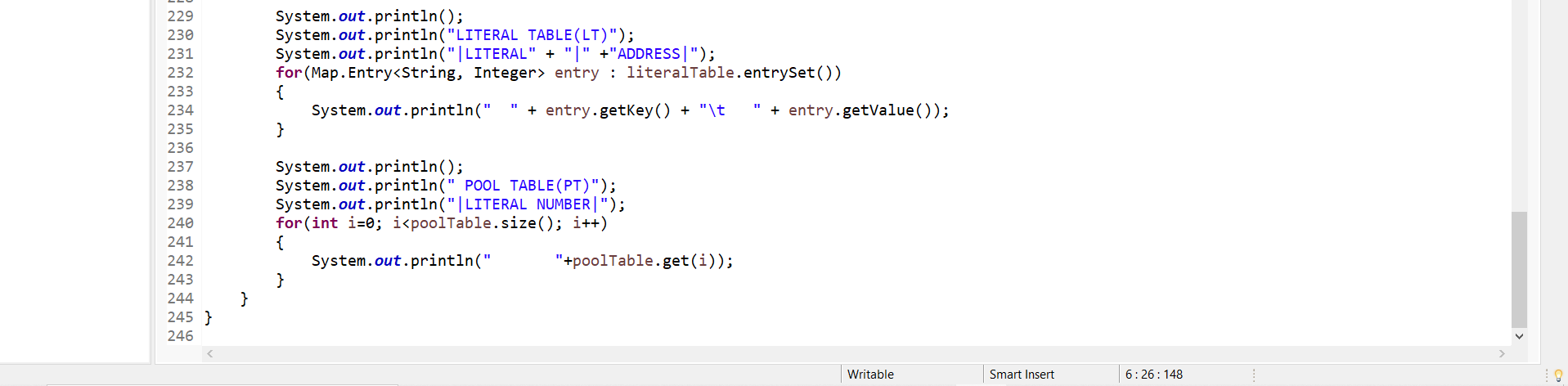
****

****

****

****

****

****

**SOURCE CODE**

**package** assignment1\_Week3;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.util.\*;

**import** java.util.regex.\*;

**import** java.text.Format;

**public** **class** Week3 {

@SuppressWarnings("resource")

**public** **static** **void** main(String[] args) **throws** FileNotFoundException

{

FileInputStream fis = **new** FileInputStream("D:\\Study\\3-TYBT\\LPCC LAB\\Week 3\\Week3\\src\\assignment1\_Week3\\mcode.txt");

Scanner sc = **new** Scanner(fis);

LinkedHashMap<Integer, List<String>> EMOT = **new** LinkedHashMap<Integer, List<String>>();

LinkedHashMap<Integer, String> classSymbol = **new** LinkedHashMap<Integer, String>();

LinkedHashMap<String, Integer> symbolTable = **new** LinkedHashMap<String, Integer>();

LinkedHashMap<String, Integer> literalTable = **new** LinkedHashMap<String, Integer>();

List<String> class1 = Arrays.*asList*("STOP", "ADD", "SUB", "MULT", "MOVER", "MOVEM", "COMP", "BC", "DIV", "READ", "PRINT");

List<String> class2 = Arrays.*asList*("DS", "DC");

List<String> class3 = Arrays.*asList*("START", "END", "ORIGIN", "EQU", "LTORG");

List<String> class4 = Arrays.*asList*("AREG", "BREG", "CREG");

List<String> class5 = Arrays.*asList*("EQ", "LT", "GT", "LE", "GE", "NE", "ANY");

List<Integer> poolTable = **new** ArrayList<Integer>();

List<String> ltorgList = **new** ArrayList<String>();

List<String> ltorgListInter = **new** ArrayList<String>();

List<String> mnemonic = Arrays.*asList*("STOP", "ADD", "SUB", "MULT", "MOVER", "MOVEM", "COMP", "BC", "DIV", "READ", "PRINT", "START",

"END", "ORIGIN", "EQU", "LTORG", "DS", "DC", "AREG", "BREG", "CREG", "EQ", "LT", "GT", "NE", "ANY");

EMOT.put(1, class1);

EMOT.put(2, class2);

EMOT.put(3, class3);

EMOT.put(4, class4);

EMOT.put(5, class5);

classSymbol.put(1, "IS");

classSymbol.put(2, "DL");

classSymbol.put(3, "AD");

classSymbol.put(4, "RG");

classSymbol.put(5, "CC");

**int** lc=0;

**int** ltorg=0;

**boolean** sTable=**false**;

**int** sT=-1;

**boolean** lTable=**false**;

**int** lT=-1;

**boolean** origin=**false**;

**int** ori=0;

**boolean** lto=**false**;

**int** ltor=0;

System.***out***.println("INTERMEDIATE CODE ALONG WITH MACHINE CODE IS AS FOLLOWS:\n");

**while**(sc.hasNextLine())

{

**int** length=0;

String code=sc.nextLine();

String a=code.replace(",", " ");

String [] arr = a.split(" ");

length = arr.length;

**if**(arr[0].equals("START"))

{

lc=Integer.*valueOf*(arr[1]);

lc=lc-2;

}

lc=lc+1;

Pattern cp = Pattern.*compile*("[0-9]+");

Matcher cm = cp.matcher(arr[length-1]);

// HANDLING ORIGIN

**if**(arr[0].equals("ORIGIN"))

{

lc = symbolTable.get(arr[1]) + Integer.*parseInt*(arr[3]) - 1;

origin=**true**;

ori=lc+1;

}

// LITERAL TABLE

Pattern p = Pattern.*compile*("=");

Matcher m = p.matcher(arr[length-1]);

**if**(m.find())

{

lTable=**true**;

lT+=1;

ltorgList.add(ltorg, arr[length-1]);

ltorg++;

}

// HANDLING LTORG AND END

**if**(arr[0].equals("LTORG") || arr[0].equals("END"))

{

lto=**true**;

poolTable.add(literalTable.size());

**for**(**int** i=0; i<ltorg; i++)

{

ltor = ltorg;

ltorgListInter.add(ltorgList.get(i));

literalTable.put(ltorgList.get(i), lc);

lc=lc+1;

}

ltorg=0;

ltorgList.clear();

lc=lc-1;

}

// SYMBOL TABLE

**boolean** match=**false**;

**for** (**int** i=0; i<26; i++)

{

**if**(arr[0].matches(mnemonic.get(i)))

{

match=**true**;

**break**;

}

}

**if**(match == **false**)

{

sTable=**true**;

sT+=1;

**if**(symbolTable.containsKey(arr[length-1]))

{

Set<String> keySet = symbolTable.keySet();

String[] keyArray = keySet.toArray( **new** String[ keySet.size() ] );

**for**(**int** i=0; i<keyArray.length; i++)

{

**if**(keyArray[i].equals(arr[length-1]))

{

**int** z = symbolTable.get(keyArray[i]);

symbolTable.put(arr[0], z);

}

}

}

**else**

{

symbolTable.put(arr[0], lc);

}

}

System.***out***.print(code + " ");

// INTERMEDIATE CODE GENERATION

**for**(**int** i=0; i<length; i++)

{

**int** found=0;

**int** index=0;

**for** (Map.Entry<Integer, List<String>> entry : EMOT.entrySet())

{

**int** key = entry.getKey();

List<String> values = entry.getValue();

**if**(values.contains(arr[i]))

{

**if**(key!=1)

{

found=key;

index=values.indexOf(arr[i]);

System.***out***.print("(" + classSymbol.get(found) + ",0" + ++index + ")");

**break**;

}

**else**

{

found=key;

index=values.indexOf(arr[i]);

System.***out***.print("(" + classSymbol.get(found) + ",0" + index + ")");

**break**;

}

}

}

}

**if**(lto)

{

**for**(**int** i=0; i<ltor; i++)

{

String temp=(ltorgListInter.get(i)).replaceAll("[^0-9]", "");

System.***out***.print("(DL,02)(C," + temp + ")");

}

lto=**false**;

ltorgListInter.clear();

}

**if**(origin)

{

System.***out***.print("(C," + ori + ")");

origin=**false**;

}

**else** **if**(cm.matches() && arr[0].equals("ORIGIN") == **false**)

{

System.***out***.print("(C," + arr[length-1] + ")");

}

**else** **if**(lTable == **true**)

{

System.***out***.print("(L," + lT + ")");

lTable=**false**;

}

**else** **if**(sTable == **true** && arr[0].equals("LTORG") && arr[0].equals("END"))

{

System.***out***.print("(S," + sT + ")");

sTable=**false**;

}

System.***out***.println("");

}

System.***out***.println();

System.***out***.println("SYMBOL TABLE(ST)");

System.***out***.println("|SYMBOL" + "|" + "ADDRESS|");

**for**(Map.Entry<String, Integer> entry : symbolTable.entrySet())

{

System.***out***.println(" " +entry.getKey() + "\t " + entry.getValue());

}

System.***out***.println();

System.***out***.println("LITERAL TABLE(LT)");

System.***out***.println("|LITERAL" + "|" +"ADDRESS|");

**for**(Map.Entry<String, Integer> entry : literalTable.entrySet())

{

System.***out***.println(" " + entry.getKey() + "\t " + entry.getValue());

}

System.***out***.println();

System.***out***.println(" POOL TABLE(PT)");

System.***out***.println("|LITERAL NUMBER|");

**for**(**int** i=0; i<poolTable.size(); i++)

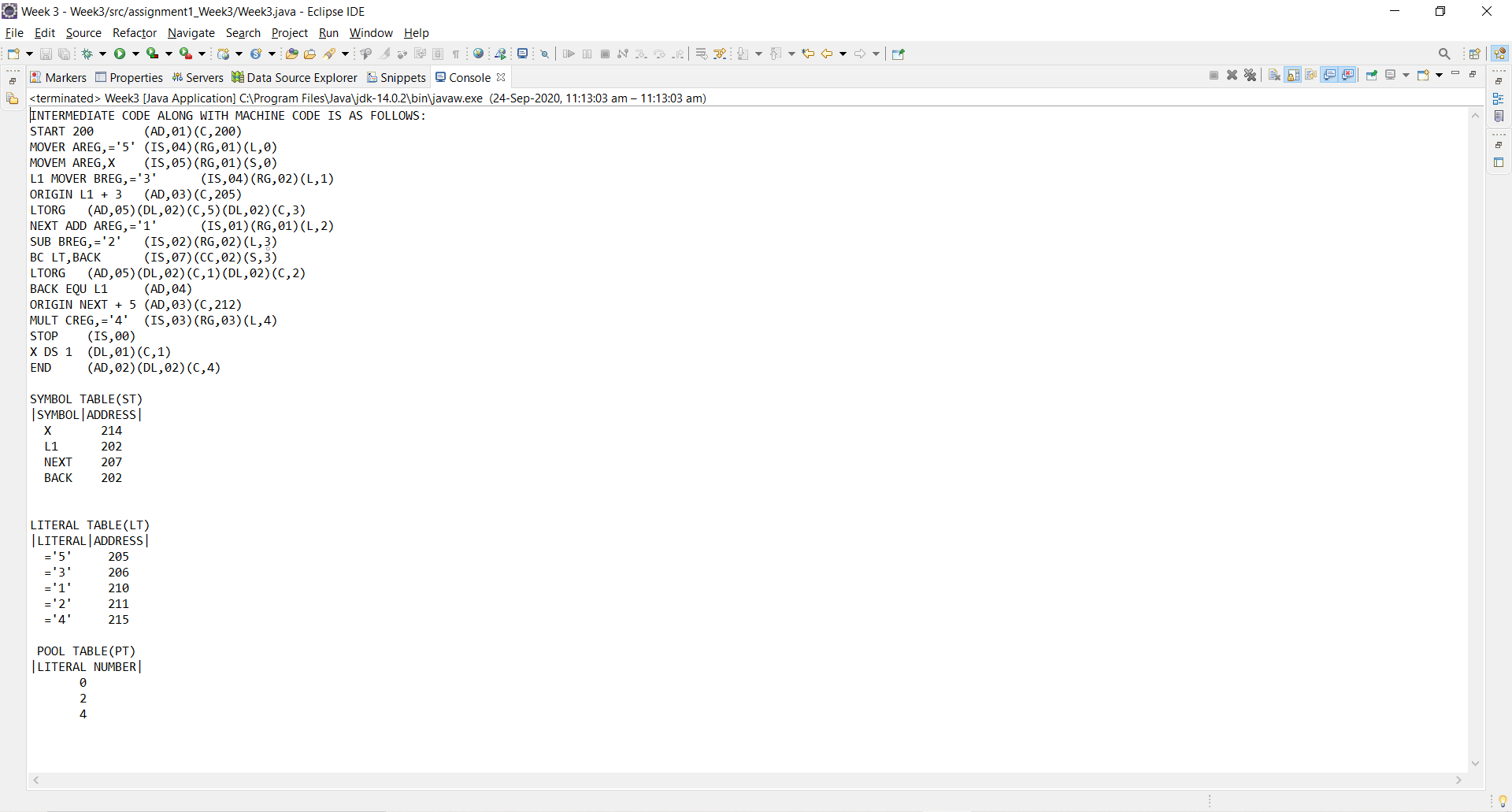
{

System.***out***.println(" "+poolTable.get(i));

}

}

}

**OUTPUT:**

**CONCLUSION:**

Successfully generated Symbol table, Literal table, Pool table & Intermediate code of a two-pass Assembler for the given source code in JAVA programming language.