Name: Digvijay D. Jondhale
Roll No: PC-32
PRN: 1032201770
Panel: C

# System Software and Compiler lab
## Assignment-03

**Aim:** Design suitable data structure & implement pass 1 of Two Pass Macroprocessor.

**Objective:** Design suitable data structure & implement pass 1 of Two Pass Macroprocessor. Input should consist of a one macro definition and one macro call and few assembly language instructions

**Theory:**

**Description of Macroprocessor:**

A macroprocessor is a program that extends the capabilities of an assembly language by allowing the use of macros. Macros are essentially user-defined shorthand notations for sequences of assembly language instructions or other constructs. They are particularly useful for simplifying and abstracting repetitive or complex tasks in assembly programming. The microprocessor takes source code containing macros as input and expands these macros into their corresponding assembly code before the actual assembly process.

Data structures Required for a Two-Pass Macroprocessor:
A two-pass macroprocessor involves two separate passes over the source code. During Pass I, it collects information about macros and their definitions, while Pass II expands the macros and produces the final assembly code. To achieve this, several data structures are required:

a] Macro Definition Table (MDT): This table stores the macro definitions encountered during Pass I. Each entry contains the macro name and its corresponding macro code.

b) Macro Invocation Table (MIT): The MIT keeps track of macro invocations and their associated arguments during Pass I. It records the macro name and the arguments passed to it.

c) Argument Replacement Table (ART): The ART maps formal parameters of macros to the actual arguments provided during macro invocation. It is used during macro expansion in Pass II.

d) Local Symbol Table (LST): If macros introduce local symbols, the LST stores these symbols and their values within the scope of the macro. It helps maintain symbol uniqueness.

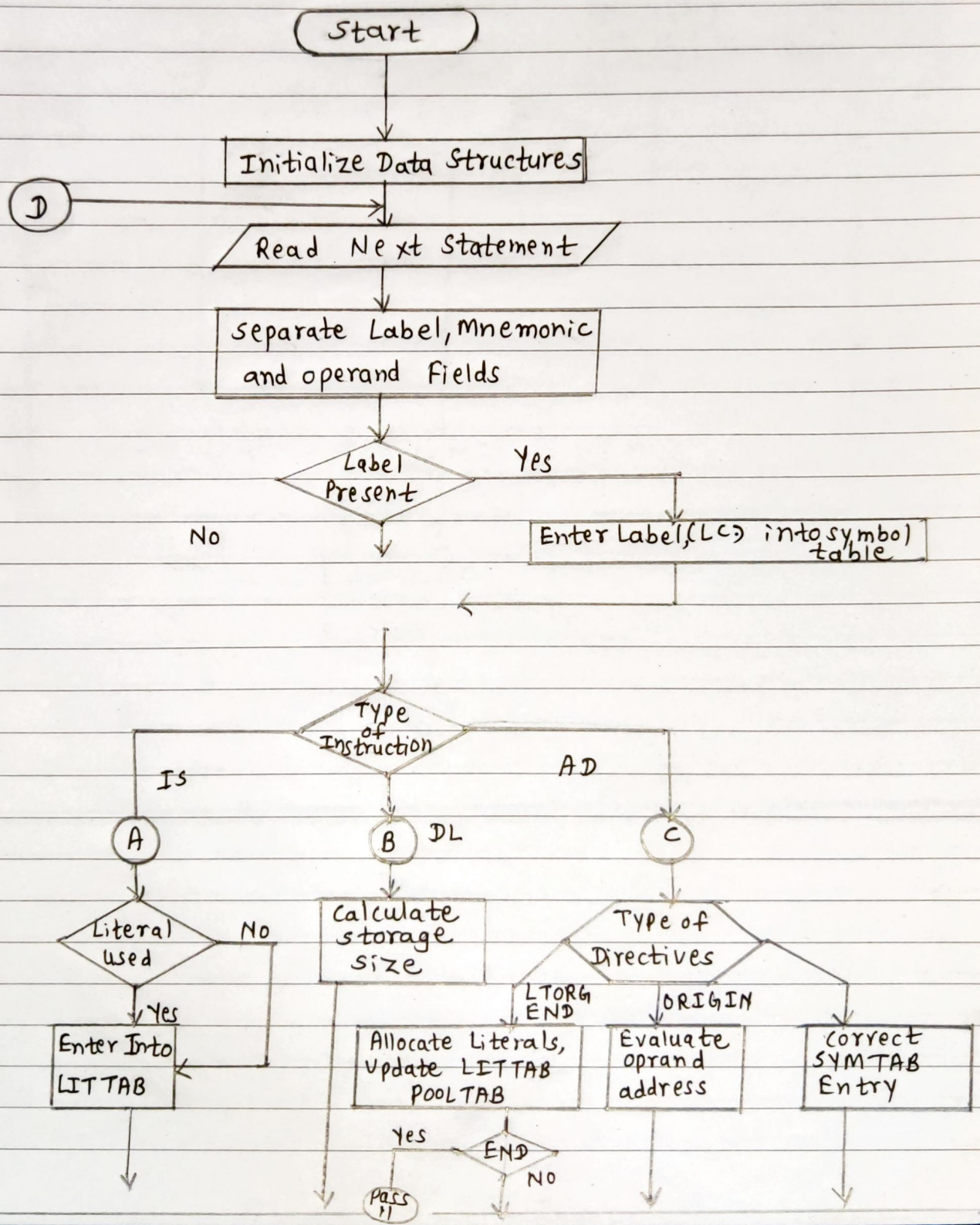
e) Global Symbol Table (GST): The GST stores global symbols of defined in the source code, allowing for cross-referencing between different macro invocations.

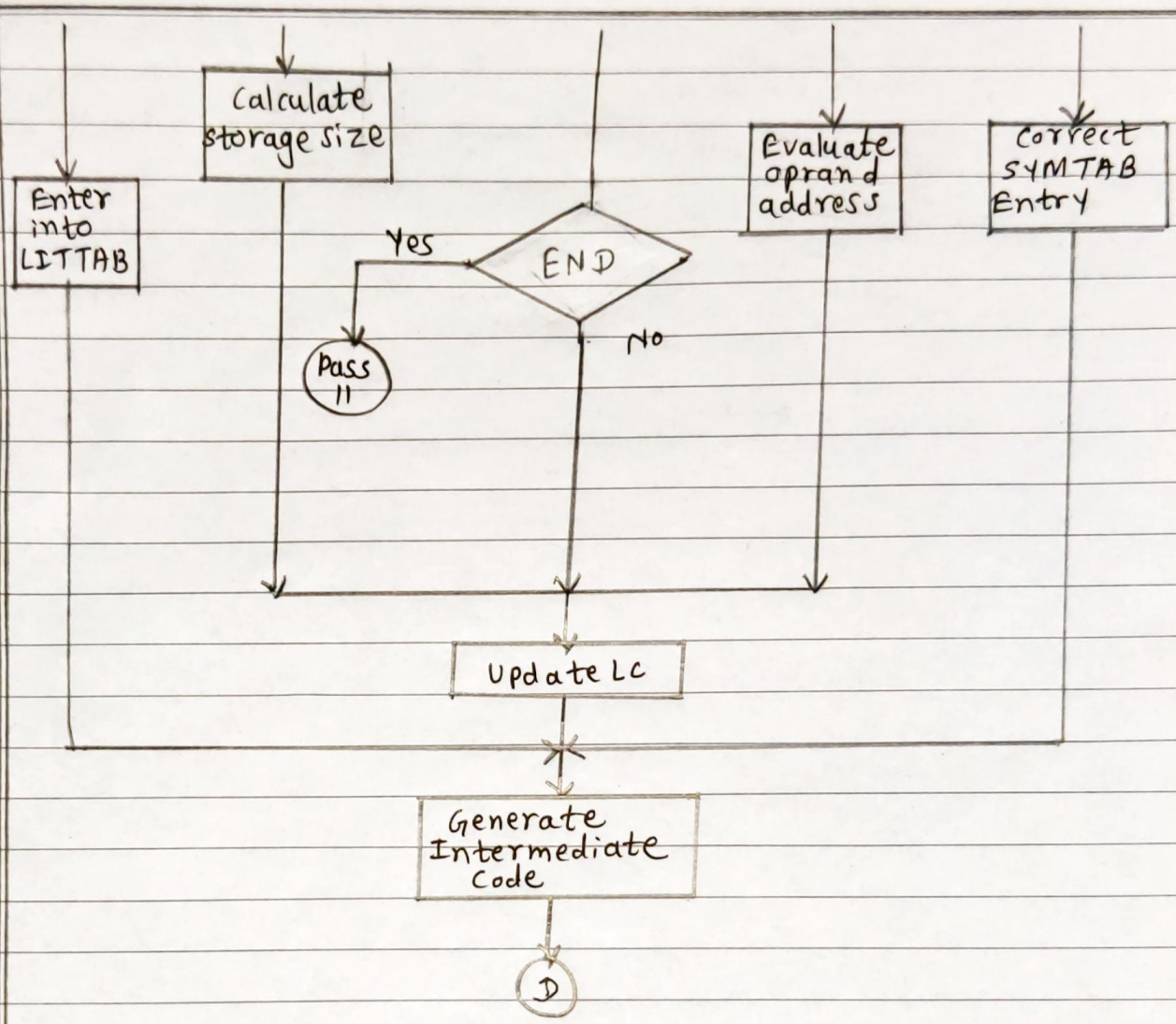
## Algorithm for Pass 1 :

Here's an algorithm that summarizes the steps involved in Pass 1 of a two-pass macroprocessor:

Initialize MDT, MIT, ART, LST, GST
Repeat until the end of source code:
Read the next line from the source code
If the line is a macro definition :
Parse the macro invocation and store it in MDT
If the line is a macro invocation :
Parse the macro invocation and store it in MIT
Add the arguments to ART
If the line contains global symbols:
Process global symbols and update GST

# Flowchart For Pass I




Enter into LITTAB
Calculate storage size
Evaluate oprand address
Correct SYMTAB Entry
END
Yes
No
Pass II
Update LC
Generate Intermediate Code
D

## CODE:

```
import java.util.*;
import java.io.*;

class M_Pass1
{
	public static void main(String[] args) throws IOException
	{
		File input = new File("m_input.asm");
		input.createNewFile();
		File output = new File("m_intermediate.asm");
		output.createNewFile();
		File tables = new File("m_tables.asm");
		tables.createNewFile();
		FileWriter fw = new FileWriter("m_intermediate.asm");
		BufferedWriter bw = new BufferedWriter(fw);

		List<String> MDT = new ArrayList<String>();
		ArrayList<String[]> MNT = new ArrayList<String[]>();
		ArrayList<String[]> ALA = new ArrayList<String[]>();
		int mdtPtr = 0, mntPtr = 0, alaPtr = 0;

		//Macroprocessor Pass 1
		Scanner fileReader = new Scanner(input);
		byte MacroDefFlag = 0;	//Stores whether in Macro (1) or after Macro Name (2)
		String[] tokens;
		while(fileReader.hasNextLine())
		{
			String i_str = fileReader.nextLine();
			if(i_str.equals("MACRO"))
			{
				MacroDefFlag = 1;
				i_str = fileReader.nextLine();
			}
			if(MacroDefFlag==1) //Processing Macro Name and arguments
			{
				tokens = i_str.split("[ ,//n]");
				for(String str : tokens)
				{
					if(str.equals(""))
						continue;
					else if(str.charAt(0) =='&')
					{
						ALA.add(new String[] {Integer.toString(alaPtr++),str});
					}
					else
					{
						//The pointer to first argument of this macro name is stored in
the 3rd value below. This will allow for multiple Macros to use the same ALA table, instead of having a
unique table for every macro.
						MNT.add(new String[]
{Integer.toString(mntPtr++),str.trim(),Integer.toString(alaPtr),Integer.toString(mdtPtr)});
					}
				}
				MacroDefFlag = 2;
				MDT.add(i_str);
				mdtPtr++;
				i_str = fileReader.nextLine();
			}

			String newstring;
```

```
        if(MacroDefFlag==2) //Processing Macro contents
        {
                tokens = i_str.split("[ ,//n]");
                newstring = "";
                for(String str : tokens)
                {
                        if(str == "")
                                continue;
                        if(str.charAt(0) == '&')  //Replacing Arguments with ALA index
                        {
                                for(int i = 0; i < ALA.size(); i++)
                                {
                                        if(ALA.get(i)[1].equals(str))
                                                newstring = newstring + " #" + ALA.get(i)[0];
                                }
                        }
                        else
                                newstring = newstring + " " + str;
                }
                if(newstring != "")
                {
                        MDT.add(newstring.trim());
                        mdtPtr++;
                }
                if(i_str.equals("MEND"))
                {
                        MacroDefFlag = 0;
                        continue;
                }
        }
        if(MacroDefFlag == 0)     //If not part of Macro
        {
                if(i_str != "")
                {
                        bw.write(i_str);
                        if(i_str.charAt(i_str.length()-1) != '\n')
                                bw.write("\n");
                }
        }
}

fileReader.close();
bw.close();
fw.close();

System.out.println("MDT: \n" + MDT);
System.out.println("\nMNT: ");
for(String[] arr : MNT)
        System.out.println(Arrays.toString(arr));
System.out.println("\nALA: ");
for(String[] arr : ALA)
        System.out.println(Arrays.toString(arr));

fileReader = new Scanner(output);
System.out.println("\n\nIntermediate Code");
while(fileReader.hasNextLine())
{
        System.out.println(fileReader.nextLine());
}
fileReader.close();
```

```
            //Writing tables to a file
            fw = new FileWriter("m_tables.asm");
            bw = new BufferedWriter(fw);
            bw.write("[MDT]\n");
            for(String str : MDT)
                  bw.write(str+"\n");
            bw.write("[MNT]\n");
            for(String[] arr : MNT)
            {
                  for(String str : arr)
                        bw.write(str+" ");
                  bw.write("\n");
            }
            bw.write("[ALA]\n");
            for(String[] arr : ALA)
            {
                  for(String str : arr)
                        bw.write(str+" ");
                  bw.write("\n");
            }
            bw.close();
            fw.close();
      }
}
```

## INPUT:

m_input.asm

```
      MACRO
      INCR &ARG1 &ARG2
            ADD AREG &ARG1
            ADD BREG &ARG2
      MEND
      START
            MOVER AREG S1
            MOVER BREG S1
            INCR D1 D2
      S1    DC 5
      D1    DC 2
      D2    DC 3
      END
```

## OUTPUT:

Problems @ Javadoc Declaration Console × Debug

```
<terminated> M_Pass1 (1) [Java Application] C:\Users\Lenovo\.p2\pool\plugi
MDT:
[INCR &ARG1 &ARG2, ADD AREG #0, ADD BREG #1, MEND]

MNT:
[0, INCR, 0, 0]

ALA:
[0, &ARG1]
[1, &ARG2]


Intermediate Code
START
        MOVER AREG S1
        MOVER BREG S1
        INCR D1 D2
S1      DC 5
D1      DC 2
D2      DC 3
END
```