Name: Digvijay Jondhale
Roll No: PC 92
PRN: 1032201770
Panel: C

SSC LCA 04

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

Theory:

- **Algorithm for Phass II:**

1. **Initialization:** Initialize necessary data structure and counters for Pass II processing.
2. **Read Intermediate code:** Read the intermediate code, perform the following steps:
   a.) Parse Line: Parse the intermediate code line to identify its components, such as opcode, operands, and symbols.
   b) Resolve Symbols: If there are unresolved symbols in the line, look up their values in the symbol table. If a symbol is found, replace it with its corresponding value. If a symbol is not found, report an error.
   c.) Generate Machine Code: Translate the intermediate code into machine code or another target representation, taking into account the resolved symbols and macro instructions.

d.) Output Machine Code: Write the generated machine code or target representation to the output file or memory.

4.) Repeat: Repeat the above steps for each line of intermediate code until you have processed the entire program.

5.) Finish processing: close any open files, release allocated memory, and perform any cleanup tasks.

6.) End of Pass II: Pass II is complete, and the output file should now contain the final assembled code, ready for execution.

- Data Structures Required for Two-Pass Macro processor:

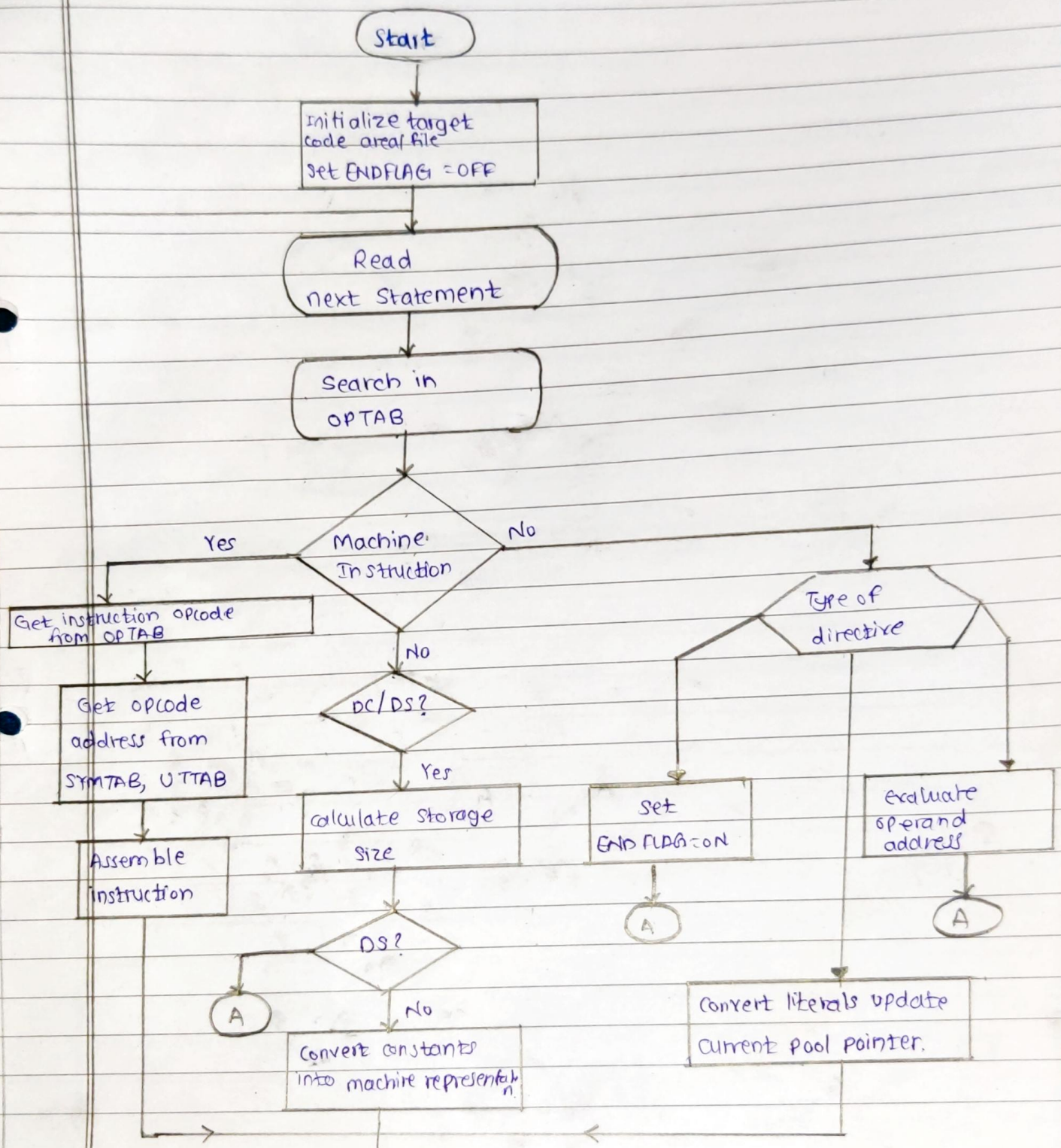
1. Intermediate Code Buffer: This data structure holds the intermediate code generated during Pass I. It can be implemented as an array or a linked list, with each element representing a line of intermediate code.
2. Symbol Table: A symbol table is used to store symbols encountered during Pass I and their corresponding values or addresses. This table is essential for resolving symbols during Pass II. It can be implemented as a hash table, binary search tree, or a simple array.
3. Macro Definition Table: This table stores information about macros defined in this source code during Pass I. It includes the macro name, its parameters, and the corresponding macro code. It allows the macroprocessor to expand macros correctly during Pass II.

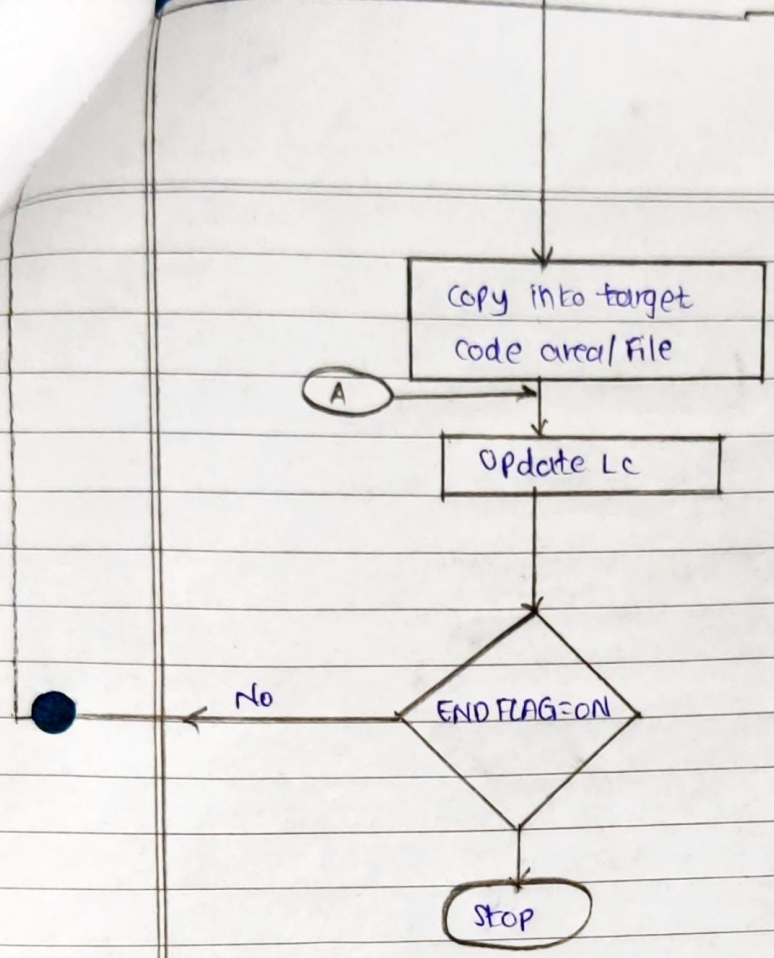
4. Output Buffer: An output buffer is used to temporaily store the generated machine code or target representation before writing it to the output file. It helps in efficiently managing the output data.

5. Error Reporting Mechanism: A mechanism for reporting errors encountered during Pass II is essential. This may include an error log or a console output to inform the user about any issues in the source code.

6. Counters and Flags: Various counters and flags are needed to keep track of the current processing state, such as the line number being processed and whether unresolved symbols were encountered.

- Flowchart of Pass II




Copy into target code area/File
A
Update LC
END FLAG=ON
No
Stop

CODE:

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

class M_Pass2
{
	public static void main(String[] args) throws IOException
	{
		File input = new File("m_intermediate.asm");
		input.createNewFile();
		File output = new File("m_output.asm");
		output.createNewFile();
		File tables = new File("m_tables.asm");
		tables.createNewFile();
		FileWriter fw = new FileWriter("m_output.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, alaPtr = 0;
		String[] tokens;

		//Reading tables from Pass 1
		Scanner fileReader = new Scanner(tables);
		int tableFlag = 0;
		int counter = 0;
		String[] a = new String[4];
		while(fileReader.hasNextLine())
		{
			String i_str = fileReader.nextLine();
			tokens = i_str.split("[ ,//n]");
			counter = 0;
			for(String str : tokens)
			{
				if(str.equals("[MDT]"))
				{
					tableFlag = 1;
					break;
				}
				else if(str.equals("[MNT]"))
				{
					tableFlag = 2;
					break;
				}
				else if(str.equals("[ALA]"))
				{
					tableFlag = 3;
					break;
				}
				switch(tableFlag)
				{
				case 2:
					a[counter++] = str;
					if(counter == 4)
					{
						MNT.add(new String[] {a[0],a[1],a[2],a[3]});
						counter = 0;
					}
					break;
```

```
                case 3:
                        a[counter++] = str;
                        if(counter == 2)
                        {
                                ALA.add(new String[] {a[0],a[1]});
                                counter = 0;
                        }
                        break;
                }
        }
        if(tableFlag == 1 && !i_str.equals("[MDT]"))
                MDT.add(i_str);
}
fileReader.close();

//Macroprocessor Pass 2
fileReader = new Scanner(input);
String[] newALA;
while(fileReader.hasNextLine())
{
        String i_str = fileReader.nextLine();
        String newstring = "";
        int CallCheckFlag = 0;    //0=Regular Code, 1=Macro Call
        tokens = i_str.split("[ ,//n]");
        CallCheckFlag = 0;
        String newline;
        for(String str : tokens)
        {
                if(str.equals(""))
                        continue;
                if(CallCheckFlag == 0)
                {
                        for(String[] m : MNT)
                        {
                                if(str.trim().equals(m[1]))      //Checks if token is in MNT
                                {
                                        alaPtr = Integer.parseInt(m[2]);
                                        mdtPtr = Integer.parseInt(m[3])+1;
                                        CallCheckFlag = 1;
                                        break;
                                }
                        }
                        if(CallCheckFlag == 0)    //Outputs non-Macro-name tokens
                        {
                                newstring = newstring + str + " ";
                        }
                }
                else if(CallCheckFlag == 1)      //Sets Dummy Args to values from function
call
                {
                        newALA = ALA.get(alaPtr);
                        newALA[1] = str;
                        ALA.set(alaPtr++, newALA);
                }
        }
        while(CallCheckFlag == 1) //Expanding Macro
        {
                tokens = MDT.get(mdtPtr++).split("[ ,//n]");
                newline = "";
                for(String str : tokens)
                {
```

```
                                if(str.charAt(0) == '#')  //Inserts Actual Arguments
                                {
                                        newline = newline +
ALA.get(Integer.parseInt(str.substring(1,str.length())))[1] + " ";
                                }
                                else if(str.equals("MEND"))
                                        CallCheckFlag = 0;
                                else
                                {
                                        newline = newline + str + " ";
                                }
                        }
                        newstring = newstring + "\t" + newline.trim();
                        if(CallCheckFlag != 0)
                                newstring = newstring + "\n";
                }
                if(newstring != "")
                {
                        if(newstring.charAt(0)=='\t')
                                newstring = "\t" + newstring.trim();
                        bw.write(newstring);
                        if(newstring.charAt(newstring.length()-1) != '\n')
                                bw.write("\n");
                }
        }

        fileReader.close();

        bw.close();
        System.out.println("MDT: " + 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\nFinal Code");
        while(fileReader.hasNextLine())
        {
                System.out.println(fileReader.nextLine());
        }
        fileReader.close();
    }
}
```

INPUT:

m_intermediate.asm

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

m_tables.asm

```
[MDT]
INCR &ARG1 &ARG2
ADD AREG #0
ADD BREG #1
MEND
[MNT]
0 INCR 0 0
[ALA]
0 &ARG1
1 &ARG2
```

OUTPUT:

```
Problems  @ Javadoc  Declaration  Console ×  Debug
<terminated> M_Pass2 [Java Application] C:\Users\Lenovo\.p2\pool\plugins
MDT: [INCR &ARG1 &ARG2, ADD AREG #0, ADD BREG #1, MEND]

MNT:
[0, INCR, 0, 0]

ALA:
[0, D1]
[1, D2]


Final Code
START
        MOVER AREG S1
        MOVER BREG S1
        ADD AREG D1
        ADD BREG D2
S1      DC 5
D1      DC 2
D2      DC 3
END
```