Grp A

SPOS
1st Assignment

Ankita Bomde
Roll No 19

Aim: To implement Pass I assembler

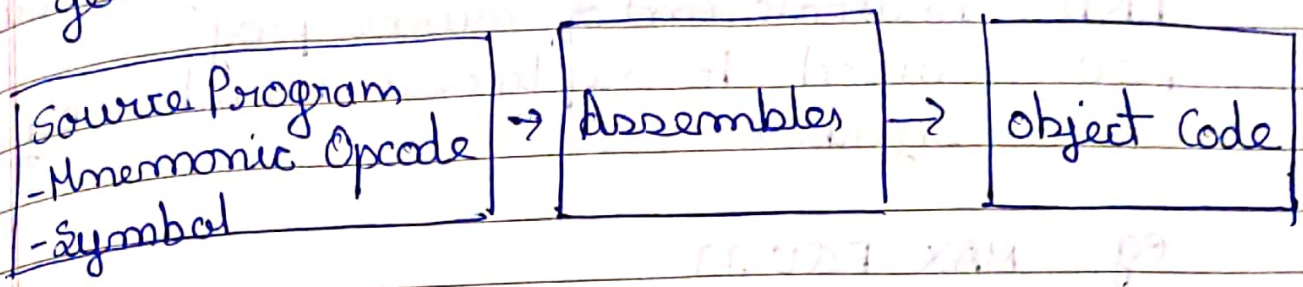
PB Statement: Design suitable DS and implement pass-I of two pass assembler for pseudo machine in Java OO feature Implementation should consists of few instruction for each category and few assembly directive

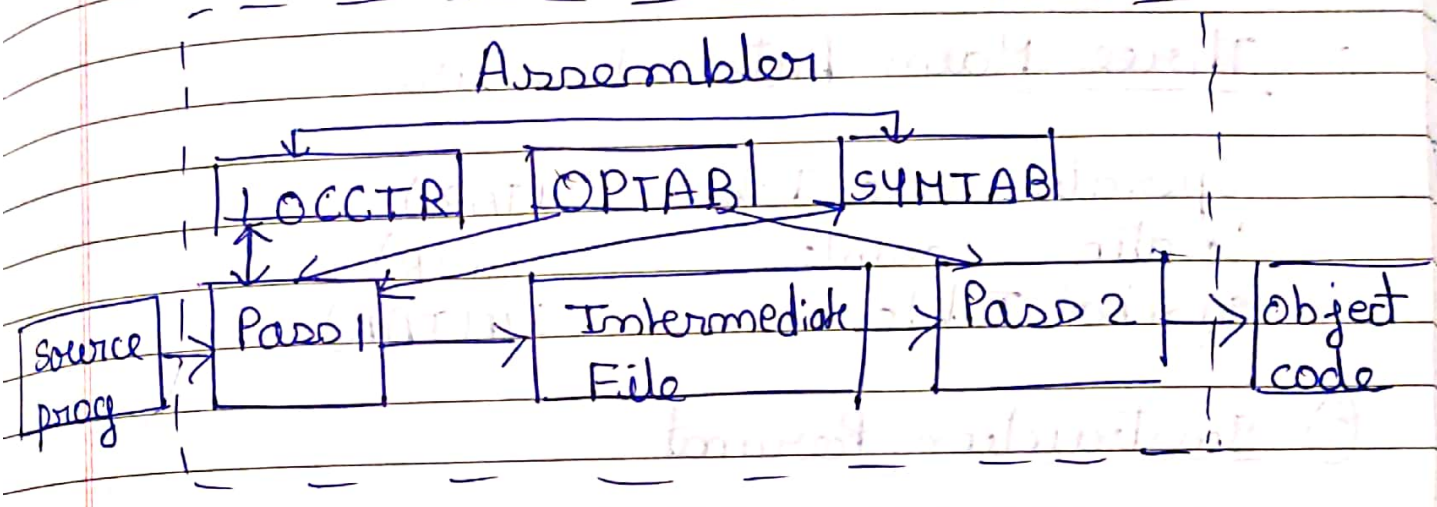
Theory

① Assembly language
- is a low level programing lang for computer or other programable deirce in which there is a very strong correspondence between the lang and architectures machine code instruction
- Each assembly lang is specific to a particular computer architecture in contrast to most high level prog lang which are generally portable across multiple architectures but require interpreting or compiling
- AS lang uses a memmonic to represent each low level machine operation or opcode.
-

② Assembler
- AS lang is converted into executable machine code by a utility prog referred to as an assembler
- The conversion process is referred to as assembly or assembling code

- Assembler is a translates that translates Assembler prog into conventional machine lang.
- It goes through prog line by line and generates machine code for that instruction





- I Pass assemblers are used when it is necessary or desirable to avoid a second pass over a source prog.
- Main problem:
  Forward refrence to both data & instruct:
- Simple way to eliminate this prob:
  req that all areas be defined before they are reffered.

③ Assembler Directives
- This are pseudo instruction

- not translated into machine instructions
- Only provide instruction / direction / info to assembler

Basic assembler Directives

START : Name & starting address of the source prog.
END : Indicate end of source prog.
EQU : used to replace a number by a symbol.

eg: MAX EQU 99

- Three Main Data Structures

Operation code Table (OPTAB)
Location Counter (LOCCTR)
Symbol Table (SYMTAB)

④ Instruction Format

① Direct addressing mode
addr of operand in instruction

② Register addressing mode
one operand is general purpose reg

③ Register indirect addressing mode
addr of operand is given by reg pair

④ Immediate addressing mode
operand is specified in instruction

④ Implicit addressing mode
Operation operate on contents of accumulator.

Program Relocation

An object prog that contains type of modification info necessary to perform modification is called as Re-locatable prog.

⑤ Literal
- It is convenient for programes to be able to write the value of a constant operand as a part of the instruction that uses it.
- Such operand is called as a literal.
- Literal identified with the prefix "=" which immediate operand ~~is use~~ "6" is literal value.

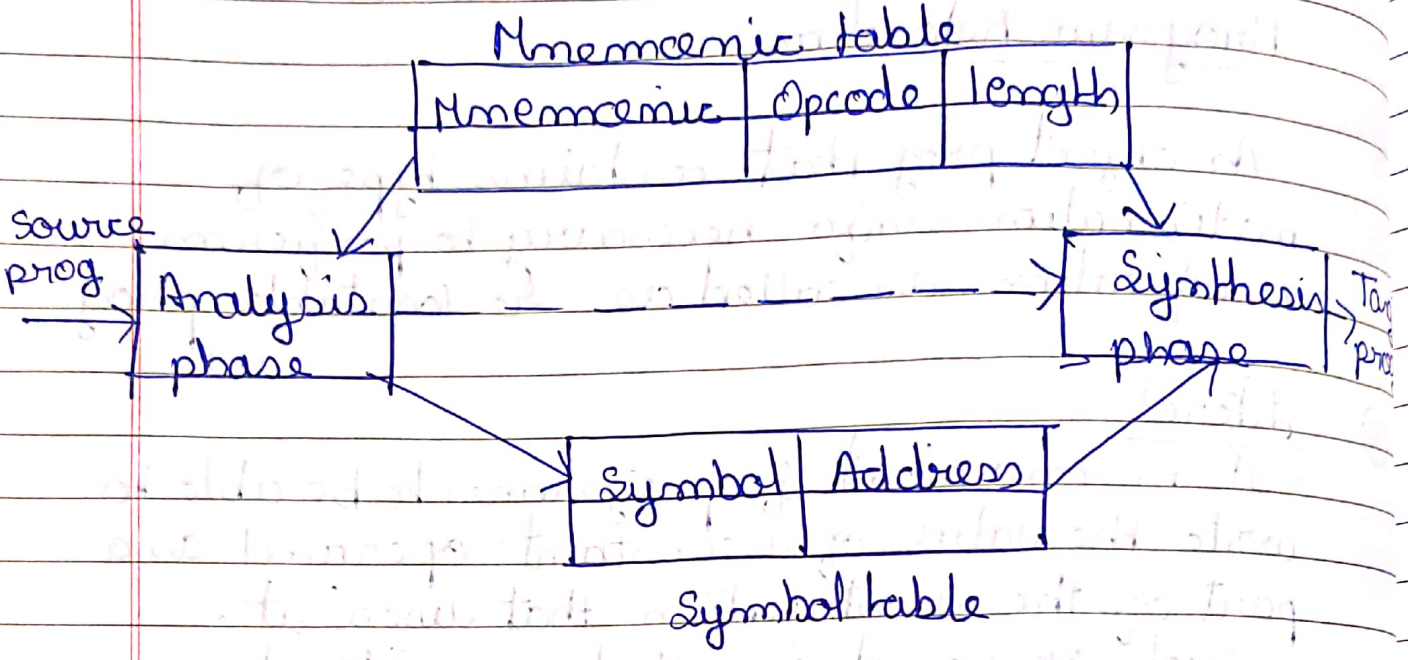
Difference between literal operands and immediate operands

- literal operand : prefix "="
immediate operand : prefix "#".

⑥ One pass assembles

- Passes over the source file exactly once in the same pass collecting the labels resolving the future ref and doing the actual assembly

- The diff part is to resolve the future label ref prob and assemble code in one pass



6.1 Forward refrence in one pass assembler

- Omits the operand address if the symbol hasnot yet been defined
- Enteres the undefined symbol in SYMTAB and indicates that as undefined
- Adds the address of this operand to the list of forward refrences associated with the SYMTAB entry
- when the definition for the symbol is encountered scans the refrences list and inserts the address
- At the end of the prog reports error if there are still SYMTAB entries indicated undefined symbols

6.2 Data structure for assembler

① opcode table
looked up for translation of mnemonic code

② Key mnemonic code

③ Symbol table
stored and looked up to assign address to labels

⑦ Algorithm for Pass I assembler

```
begin
    if starting address is given
        LOCCTR = starting address;
    else
        LOCCTR = 0;
    while OPCODE != END do ;    // or EOF
    begin
    read a line from the code
    if there is a label
        if this label in SYMTAB, then error
        else insert (label, LOCCTR) into
            SYMTAB
    search OPTAB for OPCODE
    if found
        LOCCTR += N  (N len of instructi
    else if this is an assembly directive
        update LOCCTR as directed
```

else error
write line to intermediate file
end
program size = LOCCTR - starting address
end

Conclusion

Thus we have implemented Pass I assembler using OO features

//**Name Ankita Bonde**
// TE-A 19
// ASSINGNMENT: 1

```
/*
Problem Statement: Design suitable data structures and implement pass-I of a two-pass assembler for
pseudo-machine in Java using object oriented feature. Implementation should consist of a few
instructions from each category and few assembler directives.
*/
import java.io.*;
class SymTab
{
	public static void main(String args[])throws Exception
	{
		FileReader FP=new FileReader(args[0]);
		BufferedReader bufferedReader = new BufferedReader(FP);

		String line=null;
		int line_count=0,LC=0,symTabLine=0,opTabLine=0,litTabLine=0,poolTabLine=0;

		 //Data Structures
		 final int MAX=100;
		 String SymbolTab[][]=new String[MAX][3];
		 String OpTab[][]=new String[MAX][3];
		 String LitTab[][]=new String[MAX][2];
		 int PoolTab[]=new int[MAX];
		 int litTabAddress=0;
/*-----------------------------------------------------------------------------------------------*/

		 System.out.println("_____");
		   while((line = bufferedReader.readLine()) != null)
		    {
			 String[] tokens = line.split("\t");
			if(line_count==0)
			{
				LC=Integer.parseInt(tokens[2]);
				//set LC to operand of START
				for(int i=0;i<tokens.length;i++)		//for printing the input program
					System.out.print(tokens[i]+"\t");
				System.out.println("");
			}
			else
```

```
                {
                        for(int i=0;i<tokens.length;i++) //for printing the input program
                                System.out.print(tokens[i]+"\t");
                        System.out.println("");
                        if(!tokens[0].equals(""))
                        {

                                //Inserting into Symbol Table
                                SymbolTab[symTabLine][0]=tokens[0];
                                SymbolTab[symTabLine][1]=Integer.toString(LC);
                                SymbolTab[symTabLine][2]=Integer.toString(1);
                                symTabLine++;
                        }
                        else
if(tokens[1].equalsIgnoreCase("DS")||tokens[1].equalsIgnoreCase("DC"))
                        {
                                //Entry into symbol table for declarative statements
                                SymbolTab[symTabLine][0]=tokens[0];
                                SymbolTab[symTabLine][1]=Integer.toString(LC);
                                SymbolTab[symTabLine][2]=Integer.toString(1);
                                symTabLine++;
                        }

                        if(tokens.length==3 && tokens[2].charAt(0)=='=')
                        {
                                //Entry of literals into literal table
                                LitTab[litTabLine][0]=tokens[2];
                                LitTab[litTabLine][1]=Integer.toString(LC);
                                litTabLine++;
                        }

                        else if(tokens[1]!=null)
                        {
                                        //Entry of Mnemonic in opcode table
                                OpTab[opTabLine][0]=tokens[1];


        if(tokens[1].equalsIgnoreCase("START")||tokens[1].equalsIgnoreCase("END")||tokens[1].equals
IgnoreCase("ORIGIN")||tokens[1].equalsIgnoreCase("EQU")||tokens[1].equalsIgnoreCase("LTORG"))
        //if Assembler Directive
                                {
                                        OpTab[opTabLine][1]="AD";
```

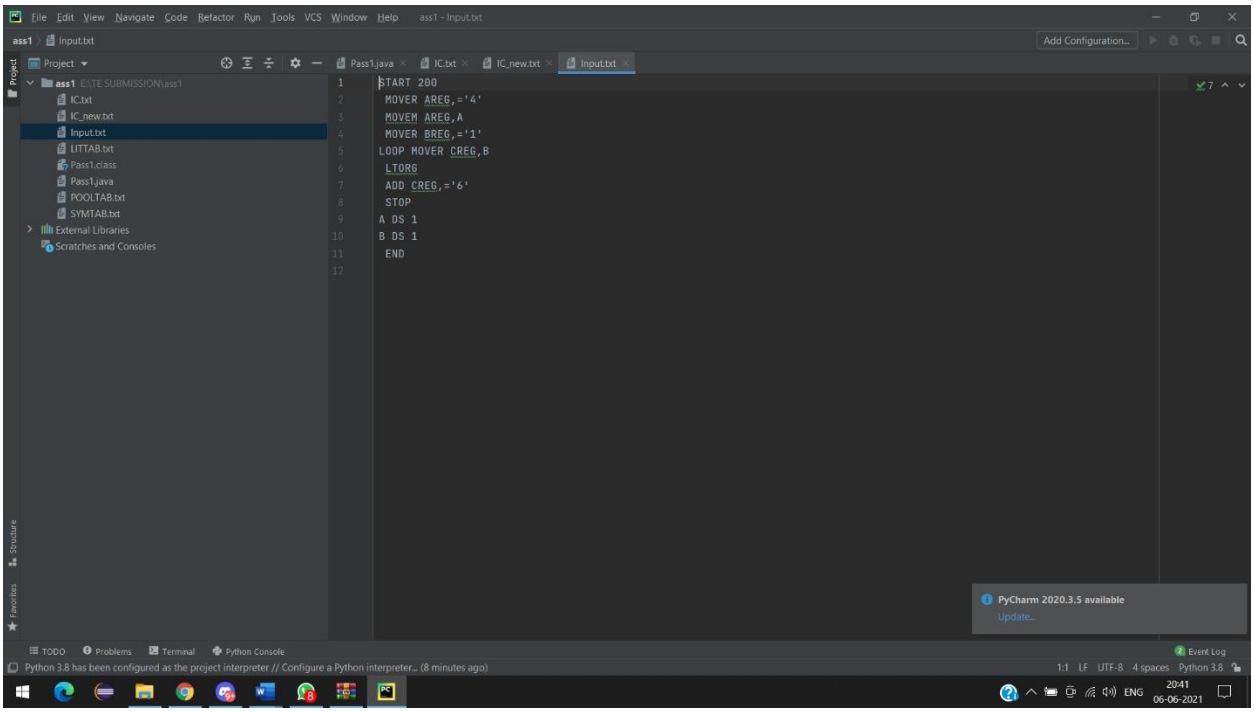
```
                                        OpTab[opTabLine][2]="R11";

                                    }
                                    else
if(tokens[1].equalsIgnoreCase("DS")||tokens[1].equalsIgnoreCase("DC"))
                                    {
                                        OpTab[opTabLine][1]="DL";
                                        OpTab[opTabLine][2]="R7";

                                    }
                                    else
                                    {
                                        OpTab[opTabLine][1]="IS";
                                        OpTab[opTabLine][2]="(04,1)";
                                    }
                                opTabLine++;
                                }
                        }
                        line_count++;
                        LC++;
                    }


        System.out.println("_____________________________________________________________");

                            //print symbol table
                            System.out.println("\n\n          SYMBOL TABLE          ");
                            System.out.println("--------------------------");
                            System.out.println("SYMBOL\tADDRESS\tLENGTH");
                            System.out.println("--------------------------");
                            for(int i=0;i<symTabLine;i++)

        System.out.println(SymbolTab[i][0]+"\t"+SymbolTab[i][1]+"\t"+SymbolTab[i][2]);
                            System.out.println("--------------------------");


                            //print opcode table
                            System.out.println("\n\n          OPCODE TABLE          ");
                            System.out.println("---------------------------");
                            System.out.println("MNEMONIC\tCLASS\tINFO");
                            System.out.println("---------------------------");
                            for(int i=0;i<opTabLine;i++)
```

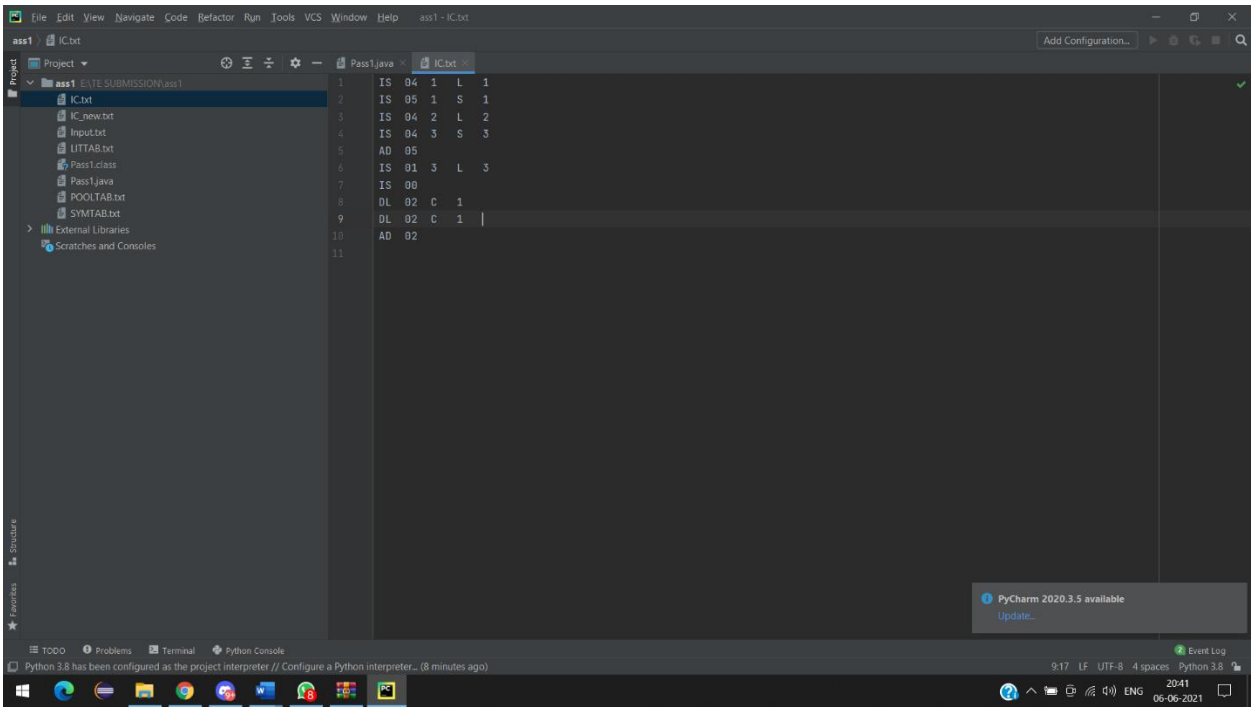
```
                System.out.println(OpTab[i][0]+"\t\t"+OpTab[i][1]+"\t"+OpTab[i][2]);
            System.out.println("----------------------------");

            //print literal table
            System.out.println("\n\n   LITERAL TABLE                    ");
            System.out.println("-----------------");
            System.out.println("LITERAL\tADDRESS");
            System.out.println("-----------------");
            for(int i=0;i<litTabLine;i++)
                System.out.println(LitTab[i][0]+"\t"+LitTab[i][1]);
            System.out.println("------------------");


            //intialization of POOLTAB
            for(int i=0;i<litTabLine;i++)
            {
                if(LitTab[i][0]!=null && LitTab[i+1][0]!=null ) //if literals are present
                {
                    if(i==0)
                    {
                        PoolTab[poolTabLine]=i+1;
                        poolTabLine++;
                    }
                    else
if(Integer.parseInt(LitTab[i][1])<(Integer.parseInt(LitTab[i+1][1]))-1)
                    {
                        PoolTab[poolTabLine]=i+2;
                        poolTabLine++;
                    }
                }
            }
            //print pool table
            System.out.println("\n\n   POOL TABLE            ");
            System.out.println("-----------------");
            System.out.println("LITERAL NUMBER");
            System.out.println("-----------------");
            for(int i=0;i<poolTabLine;i++)
                System.out.println(PoolTab[i]);
            System.out.println("------------------");


        // Always close files.
```

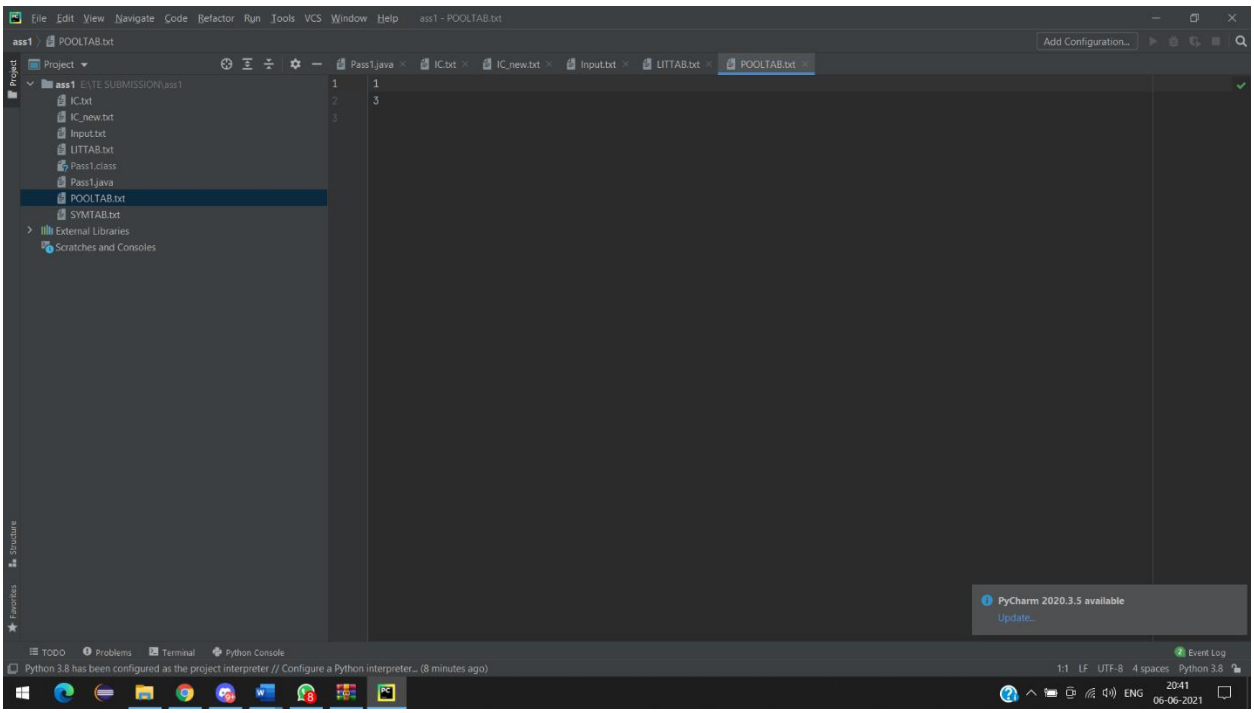
```
                bufferedReader.close();
        }
}
```

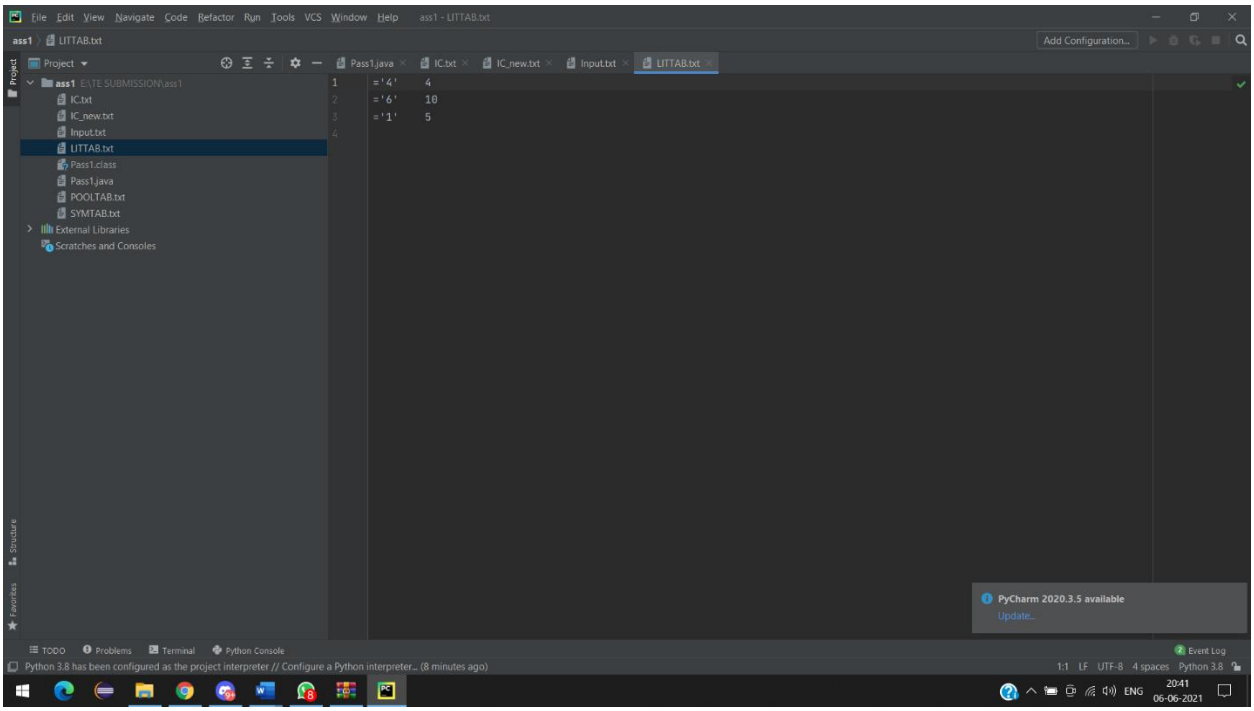
/*

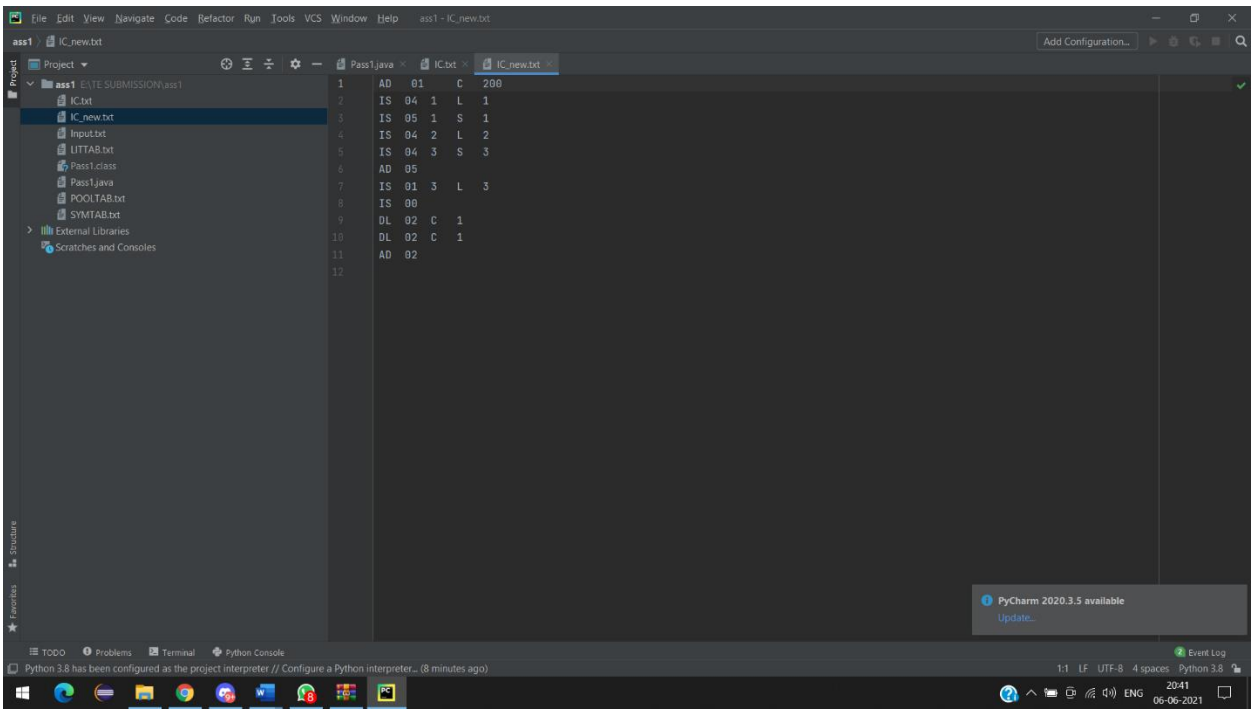
_____OUTPUT_____




File Edit View Navigate Code Refactor Run Tools VCS Window Help ass1 - IC.txt
ass1 IC.txt
Add Configuration...
Project
Pass1.java
IC.txt
ass1 E:\TE SUBMISSION\ass1
IC.txt
IC_new.txt
Input.txt
LITTAB.txt
Pass1.class
Pass1.java
POOLTAB.txt
SYMTAB.txt
External Libraries
Scratches and Consoles
IS 04 1 L 1
IS 05 1 S 1
IS 04 2 L 2
IS 04 3 S 3
AD 05
IS 01 3 L 3
IS 00
DL 02 C 1
DL 02 C 1
AD 02
PyCharm 2020.3.5 available
Update...
TODO Problems Terminal Python Console
Event Log
Python 3.8 has been configured as the project interpreter // Configure a Python interpreter... (8 minutes ago)
9:17 LF UTF-8 4 spaces Python 3.8
ENG
20:41
06-06-2021


File Edit View Navigate Code Refactor Run Tools VCS Window Help ass1 - POOLTAB.txt
ass1 POOLTAB.txt
Add Configuration...
Project
Pass1.java
IC.txt
IC_new.txt
Input.txt
LITTAB.txt
POOLTAB.txt
ass1 E:\TE SUBMISSION\ass1
IC.txt
IC_new.txt
Input.txt
LITTAB.txt
Pass1.class
Pass1.java
POOLTAB.txt
SYMTAB.txt
External Libraries
Scratches and Consoles
1
3
PyCharm 2020.3.5 available
Update...
TODO Problems Terminal Python Console
Event Log
Python 3.8 has been configured as the project interpreter // Configure a Python interpreter... (8 minutes ago)
1:1 LF UTF-8 4 spaces Python 3.8
ENG
20:41
06-06-2021


File Edit View Navigate Code Refactor Run Tools VCS Window Help ass1 - LITTAB.txt
ass1 LITTAB.txt
Add Configuration...
Project
Pass1.java IC.txt IC_new.txt Input.txt LITTAB.txt
ass1 E:\TE SUBMISSION\ass1
IC.txt
IC_new.txt
Input.txt
LITTAB.txt
Pass1.class
Pass1.java
POOLTAB.txt
SYMTAB.txt
External Libraries
Scratches and Consoles
='4' 4
='6' 10
='1' 5
PyCharm 2020.3.5 available
Update...
TODO Problems Terminal Python Console
Event Log
Python 3.8 has been configured as the project interpreter // Configure a Python interpreter... (8 minutes ago)
1:1 LF UTF-8 4 spaces Python 3.8
ENG
20:41
06-06-2021


File Edit View Navigate Code Refactor Run Tools VCS Window Help ass1 - IC_new.txt
ass1 IC_new.txt
Add Configuration...
Project
Pass1.java IC.txt IC_new.txt
ass1 E:\TE SUBMISSION\ass1
IC.txt
IC_new.txt
Input.txt
LITTAB.txt
Pass1.class
Pass1.java
POOLTAB.txt
SYMTAB.txt
External Libraries
Scratches and Consoles
AD 01 C 200
IS 04 1 L 1
IS 05 1 S 1
IS 04 2 L 2
IS 04 3 S 3
AD 05
IS 01 3 L 3
IS 00
DL 02 C 1
DL 02 C 1
AD 02
PyCharm 2020.3.5 available
Update...
TODO Problems Terminal Python Console
Event Log
Python 3.8 has been configured as the project interpreter // Configure a Python interpreter... (8 minutes ago)
1:1 LF UTF-8 4 spaces Python 3.8
ENG
20:41
06-06-2021