Pritam Mogal TE(B)-17

Assignment No 1

PASS-I ASSEMBLER.

Aim: To implement Pass-1 assembler.

Problem Statement: Design suitable data structure and implement Pass-1 of a two pass assembler for pseudo-machine in Java using object orient feature. Implementation should consist of a few instrunctions from each cateogory and few assembler directives.

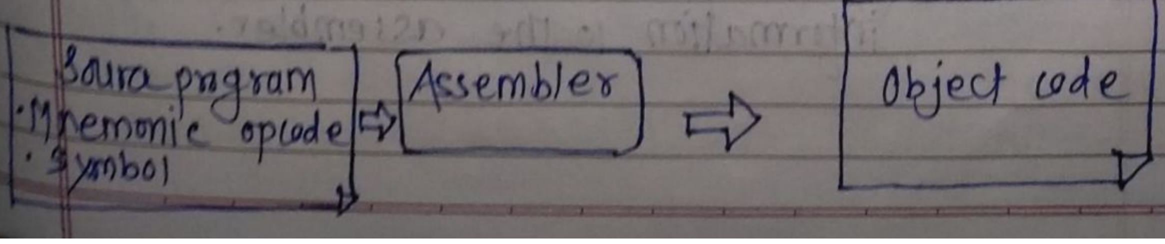
Theory:

Assembler language:

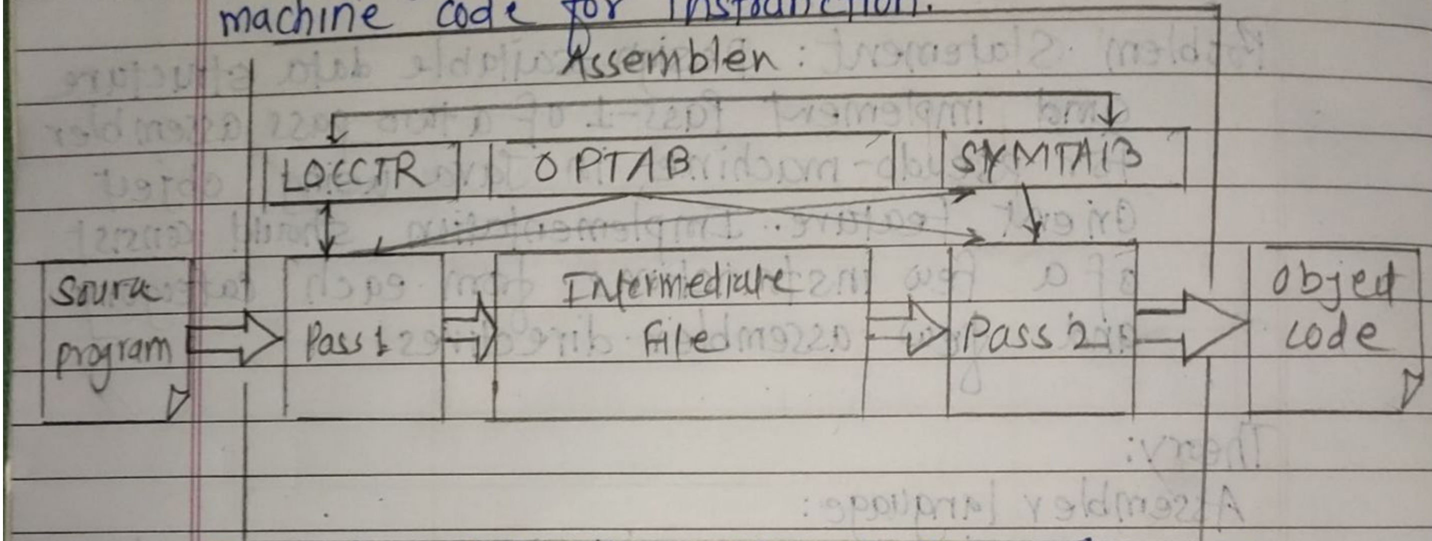
An assembly langugue is low level programming languge for computer, or the programmable device, in which there is very strong (generally one-to-one) correspondance between the languge and the architectures machine code instrunctions. Each assembly langugue is specific to a particular computer architecture, in contrast to most high-level Programming language.

Assembler:

Assembly langugue is converted into executable machine code by utility program reffered to an an assembler the conversion process is reffered to as assembly, or assembling the code.



An assembler is a translator that translates an assembler into a conventional machine languge program. Basically, the assembler goes through the program one line at a time; and generate machine code for instrunction.



The indermediate file include each Soura Statement, assigned address and error indicator.

Translate assembly languge programs to object programs to machine code is called as assembler.
One simple way to eliminate this problem: require that all areas be defined before they are referenced It is possible, although inconnienut, to do so for data items.

Assemble directives:

- Assembler directives are pseudo instrunctions.
  - They will not be translated into machine instrunctions.
  - They one provide instrunction/direction/information to the assembler.

- Basic assembles directives :
  - START : Specify name and Starting address for the program.
  - ENP: Indicate The end of the source program.
  - EQU: The EQU directive is used to replace a number by a symbol. For example: MAXIMUM EQU 99. After using this directive, erery appearance for the label "Maximum" in the program will be interpreted by the assembler.

Three main Data structures
- Operation Code table (OPTAB)
- Location Counter (LOCCTR)
- Symbol Table (SYMTAB).

Instrunction formats::
- Addressing modes: Direct addressing (address of operand is given in instrunction itself). Register addressing (one of the operand is general purpose register) Register indirect addressing (address of operand is specified by register pair. Immediate addressing (operand - data is specified in the instrunction itself) Implicit addressing (mostly the operation operates on the contents of accumulator).

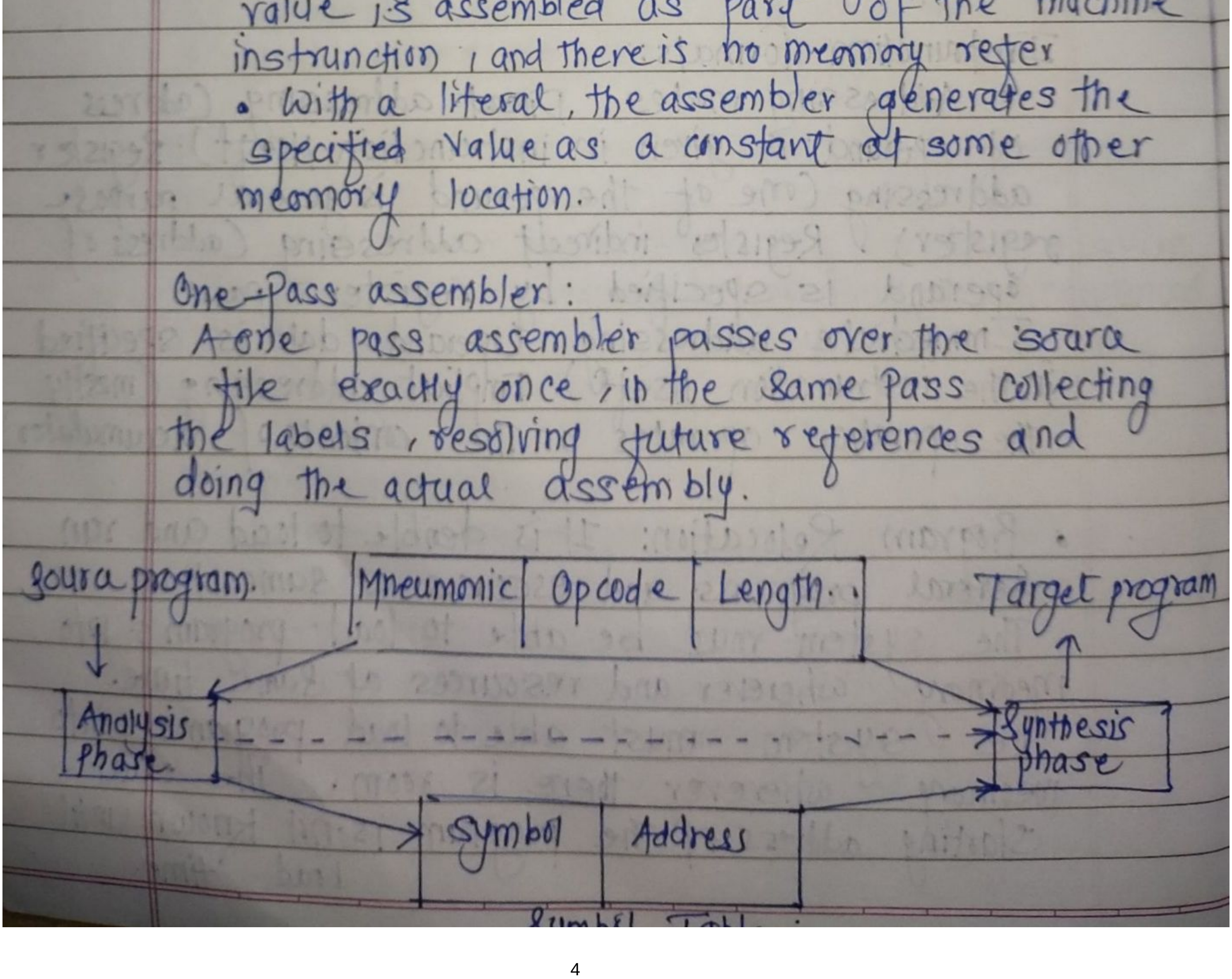
- Program Relocation: It is desible to load and run several programs and resources at same time. The system must be able to load programs into meomory wherever and resources at same time. The system must able to load programs into meomory wherever there is room. The exact Starting address of the program is not known until load time.

Q. ~~22~~
~~macro name table~~

Literal :
it is convinient for the Programmer to be able to write the value of a constant operand as a part of the instrunction that uses it. Such an operand is called a literal.
The difference between literal operands and immediate operands

- For literal operand we use '=' as prefix, and with immediate operand we use '#' as prefix.
- During immediate addressing, the operand value is assembled as part of the machine instrunction, and there is no meomory refer
- With a literal, the assembler generates the specified value as a constant at some other meomory location.

One-Pass assembler :
A one pass assembler passes over the soura file exactly once, in the same pass collecting the labels, resolving future references and doing the actual assembly.



forward referena in one assembler..

- Omits the operand address if the symbol has not yet been defined.
- Enters this undefined symbol into SYMTAB and indicates that it is undefined
- Adds the address of this operand address to a list of forward references associated with the SYMTAB entry.
- When the defination for the symbol is encountered, scans the referena list and inserts the address.
- At the end defination for the symbol is encountered, scans the reference list and inserts the address.
- At the end of the program the error if there are still SYMTAB entries indicated undefined symbols.

Data structures for assembler:

Op code Table

Looked up for the translation of mneomonic code.

Key: mneomonic code.

Algorithm for pass 1 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 is in SYMTAB, then error
```

```
        else insert (label, LOCCTR) into SYMTAB
  search OPTAB for the op code.
  if found
        LOCCTR += N ;; N is the length of this
                      instruction (4 for MIPS)
  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.
```

Algorithm 4.1 (Assembler first pass)

```
  1. loc_cntr:= 0 (default value)
     pooltab_ptr:=1; POOLTAB[1]:=1;
       littab_ptr:= 1;
  2. while next statement is not an END
     statement.
     (a) if label is present then
           this_label:= symbol in label field;
         Enter(this, label, loc_cntr) in SYMTAB.
     (b) If an LTORG statement then
     (i) Process literals LITAB [POOLTAB
           [POOLTAB_ptr]] ... LITTAB (lit-tab_ptr-1)
     to allocate memory and put the address
     in the address field.
     (ii) pooltab_ptr:= pooltab_ptr+1;
     (iii) POOLTAB [pooltab_ptr]:= littab_ptr,
     (c) If a START or ORIGIN statement then
       loc_cntr := value specified in operand
       field;
```

```
(d) If an EQU Statement then :
   (i) this_addr := value of <address spec>;
   (ii) Correct the symtab entry for this label to
        (this_label, this_addr).
(e) If a declaration statement then
   (i) code := code of the declaration statement;
   (ii) size := size of meomory area required by
        DC / DS.
   (iii) loc_cntr := loc-cntr + size;
   (iv) Generate IC (DL code)....'

(f) If an imperative statement then
   (i) code := machine opcode from OPTAB;
   (ii) loc_cntr := loc_cntr + instruction length from
        OPTAB;
   (iii) If operated is a literal then
           this_literal := literal in operand field;
           LITTAB [littab_ptr] := this_literal;
           littab_ptr := littab_ptr + 1;
         else (i,e) operand is a symbol)
           this_entry := SYMTAB entry number of
              operand;
         Generate IC '(IS, code) (S, this_entry)';

3. (Processing of END Statement)
   (a) Perform step 2(b).
   (b) Generate IC '(AD, 02).
   (c) Go to Pass II
```

Input :

```
START  200
MOVER  AREG, ='4'
```

```
MOVEM AREG, A
MOVER BREG, ='1'
LOOP MOVER CREG, B
 LTORG
 ADD GREG, ='6'
  STOP
 A DS 1
 B DS 1
   END
```

Expected Output : Symbol Table

A	208
LOOP	203
B	209

Intermediate Code

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			

Conclusion :

Thus we have implement PASS-1 Assembler using object oriented features.

Name: Pritam Gotiram Mogal **DivRoll**: TE -B-17

Assignment No. 01 [Pass 1 Assembler]

Problem Satement: Design suitable data structures and implement pass-I of a twopass assembler for pseudo-machine in Java using object oriented feature. Implementation should consist of a few instructions from each category and few assembler directives

1. Pass 1 Program:

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

public class Pass1 { public static void
        main(String[] args) {

                BufferedReader br = null;
                FileReader fr = null;

                FileWriter fw = null;
                BufferedWriter bw = null;

                try {
                        String inputfilename = "/home/sagar-ravan/Desktop/Input.txt";
                        fr = new FileReader(inputfilename); br = new
                        BufferedReader(fr);

                        String OUTPUTFILENAME = "/home/sagar-ravan/Desktop/IC.txt";
                        fw = new FileWriter(OUTPUTFILENAME);
                        bw = new BufferedWriter(fw);

                        Hashtable<String, String> is = new Hashtable<String, String>();
                        is.put("STOP", "00"); is.put("ADD", "01"); is.put("SUB",
                        "02"); is.put("MULT", "03"); is.put("MOVER", "04");
                        is.put("MOVEM", "05"); is.put("COMP", "06"); is.put("BC",
                        "07"); is.put("DIV", "08"); is.put("READ", "09");
                        is.put("PRINT", "10");

                        Hashtable<String, String> dl = new Hashtable<String, String>();
                        dl.put("DC", "01"); dl.put("DS", "02");
                        Hashtable<String, String> ad = new Hashtable<String, String>();

                        ad.put("START", "01");
                        ad.put("END", "02");
                        ad.put("ORIGIN", "03");
```

```
            ad.put("EQU", "04");
            ad.put("LTORG", "05");

            Hashtable<String, String> symtab = new Hashtable<String, String>();
            Hashtable<String, String> littab = new Hashtable<String, String>();
            ArrayList<Integer> pooltab = new ArrayList<Integer>();

            String sCurrentLine; int
            locptr = 0; int litptr = 1; int
            symptr = 1; int pooltabptr =
            1; sCurrentLine =
            br.readLine();

            String s1 = sCurrentLine.split(" ")[1];
            if (s1.equals("START")) {
                bw.write("AD \t 01 \t");
                String s2 = sCurrentLine.split(" ")[2];
                bw.write("C \t" + s2 + "\n");
                locptr = Integer.parseInt(s2);
            }

            while ((sCurrentLine = br.readLine()) != null) { int mind_the_LC = 0;
                String type = null; int flag2 = 0; // checks whether addr is
                assigned to current symbol

                String s = sCurrentLine.split(" |\\,")[0]; // consider the first word in the
line

                for (Map.Entry m : symtab.entrySet()) { // allocating addr to arrived
symbols if (s.equals(m.getKey())) {
                        m.setValue(locptr);
                        flag2 = 1;
                    }
                }
                if (s.length() != 0 && flag2 == 0) { // if current string is not " " or
addr is not assigned,

    // then the current string must be a new symbol.
                 symtab.put(s, String.valueOf(locptr));
                    symptr++;
                }
                int isOpcode = 0; // checks whether current word is an opcode or not

                s = sCurrentLine.split(" |\\,")[1]; // consider the second word in the
line

                for (Map.Entry m : is.entrySet()) { if (s.equals(m.getKey())) {
                    bw.write("IS\t" + m.getValue() + "\t"); // if match found
in imperative stmt
```

```
                        type = "is";
                        isOpcode = 1;
                    }
                }

                for (Map.Entry m : ad.entrySet()) { if (s.equals(m.getKey())) {
                    bw.write("AD\t" + m.getValue() + "\t"); // if match
found in Assembler Directive type = "ad"; isOpcode = 1;
                    }
                }
                for (Map.Entry m : dl.entrySet()) { if (s.equals(m.getKey())) {
                    bw.write("DL\t" + m.getValue() + "\t"); // if match
found in declarative stmt type = "dl"; isOpcode = 1;
                    }
                }

                if (s.equals("LTORG")) {
                    pooltab.add(pooltabptr);
                    for (Map.Entry m : littab.entrySet()) { if (m.getValue() == "")
                        { // if addr is not assigned to the
literal
                            m.setValue(locptr);
                            locptr++;
                            pooltabptr++;
                            mind_the_LC = 1;
                            isOpcode = 1;
                          }
                    }
                }

                if (s.equals("END")) {
                    pooltab.add(pooltabptr);
                    for (Map.Entry m : littab.entrySet()) {
                        if (m.getValue() == "") {
                            m.setValue(locptr);
                            locptr++; mind_the_LC =
                            1;
                          }
                    }
                }

                if (s.equals("EQU")) { symtab.put("equ",
                    String.valueOf(locptr));
                }

                if (sCurrentLine.split(" |\\,").length > 2) { // if there are 3 words
                    s = sCurrentLine.split(" |\\,")[2]; // consider the 3rd word

                    // this is our first operand.
```

```
                        // it must be either a
                        Register/Declaration/Symbol if
                        (s.equals("AREG")) { bw.write("1\t"); isOpcode
                        = 1;
                        } else if (s.equals("BREG")) {
                                bw.write("2\t");
                                isOpcode = 1;
                        } else if (s.equals("CREG")) {
                                bw.write("3\t");
                                isOpcode = 1;
                        } else if (s.equals("DREG")) {
                                bw.write("4\t");
                                isOpcode = 1;
                        } else if (type == "dl") {
                                bw.write("C\t" + s + "\t");
                        } else { symtab.put(s, ""); // forward referenced
                        symbol }
                }

                if (sCurrentLine.split(" |\\,").length > 3) { // if there are 4 words

                        s = sCurrentLine.split(" |\\,")[3]; // consider 4th word.


    // this is our 2nd operand

    // it is either a literal, or a symbol if
                        (s.contains("=")) {
                        littab.put(s, "");
                      bw.write("L\t" + litptr + "\t");
                                isOpcode = 1;
                                litptr++;
                        } else { symtab.put(s, ""); // Doubt : what if the current
                                symbol
is already present in SYMTAB?
                                                                        // Overwrite?
                                bw.write("S\t" + symptr + "\t");
                                symptr++;
                                        }
                }

                    bw.write("\n"); // done with a line.

                if (mind_the_LC == 0)
                        locptr++;
        }

        String f1 = "/home/sagar-ravan/Desktop/SYMTAB.txt";
                FileWriter fw1 = new FileWriter(f1);
        BufferedWriter bw1 = new BufferedWriter(fw1); for
        (Map.Entry m : symtab.entrySet()) { bw1.write(m.getKey()
                + "\t" + m.getValue() + "\n");
                System.out.println(m.getKey() + " " + m.getValue());
                }

                String f2 = "/home/sagar-ravan/Desktop/LITTAB.txt";
                        FileWriter fw2 = new FileWriter(f2);
                BufferedWriter bw2 = new BufferedWriter(fw2); for
                (Map.Entry m : littab.entrySet()) { bw2.write(m.getKey() +
                "\t" + m.getValue() + "\n"); System.out.println(m.getKey()
                + " " + m.getValue());
                }

                String f3 = "/home/sagar-ravan/Desktop/POOLTAB.txt";
                        FileWriter fw3 = new FileWriter(f3);
                BufferedWriter bw3 = new BufferedWriter(fw3);
                for (Integer item : pooltab) {
                        bw3.write(item + "\n");
                        System.out.println(item);
                }

                bw.close();
                bw1.close();
                bw2.close();
                bw3.close();

            } catch (IOException e) {
                e.printStackTrace();
            }

        }

}
```

PASS 1 - ASSEMBLER OUTPUT:

```
Pritam-spos@pritam-HP:~/Desktop$ javac Pass1.java


Note: Pass1.java uses unchecked or unsafe operations.


Note: Recompile with –Xlint:unchecked for details.


Pritam-spos@-HP:~/Desktop$ java Pass1 Input.txt


A  8


LOOP 3
```

B 9

= ‘4’ 4

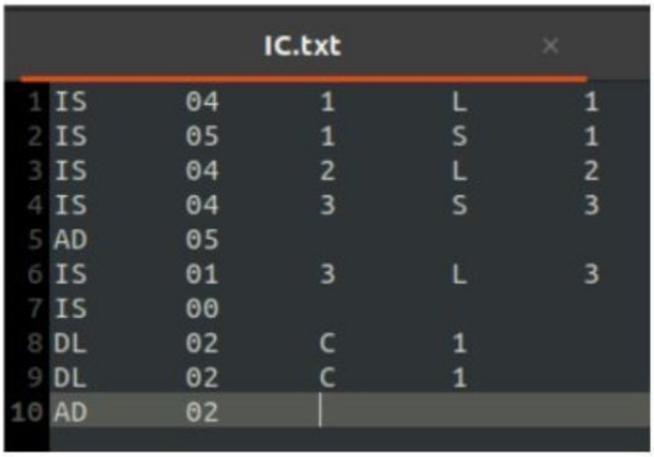
= ‘6’ 10

= ‘1’ 5

1

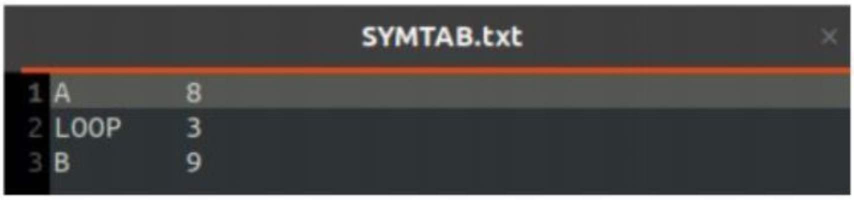
3

IC.txt



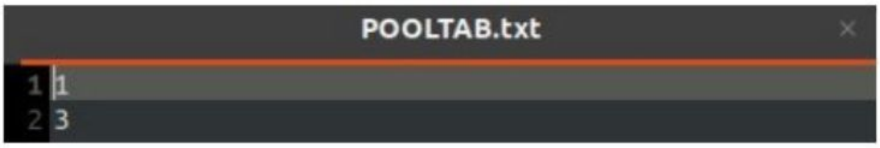
```
IS      04      1       L       1
IS      05      1       S       1
IS      04      2       L       2
IS      04      3       S       3
AD      05
IS      01      3       L       3
IS      00
DL      02      C       1
DL      02      C       1
AD      02
```

SYMTAB.txt

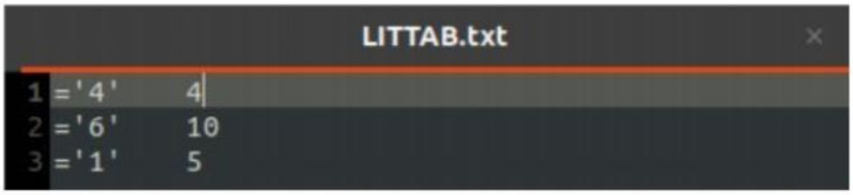


```
A       8
LOOP    3
B       9
```

LITTAB.txt
POOLTAB.txt



```
1
3
```



```
='4'     4
='6'     10
='1'     5
```