a. (4 point)

Q.6. (**60 points**) Write a **parser** in **JAVA** for the following grammar. The parser should accept a file from the command line: **`java -jar parser.jar <filename>`**.

```
program --> stmts
stmts --> stmt eos stmts | stmt eos
stmt --> assign-stmt | write-stmt | read-stmt
assign-stmt --> id assignop expr
read-stmt --> read id
write-stmt --> write id
expr --> term { addop term }
term --> factor { mulop factor }
factor --> id | num

id --> letter+
num --> digit+

letter --> [a-zA-Z]
digit --> [0-9]
eos --> ';'
addop --> '+' | '-'
mulop --> '*' | '/'
assignop --> ':='
```

The input/output statements begin with the reserved words **read** and **write**. The read statement can read only one variable (the id) at a time, and a write statement can write only one variable (the id) at a time. One of the programs (**sample.txt**) written using the above grammar is as follows:

```
read x;
x := x * 10;
write x;
y := x + 10;
write y;
x := x + y * 10;
write x;
z := y + x / 10;
write z;
```

The parser should output the **source code (with annotations – tokens etc)** and the **Syntax Tree** of the grammar. As an example, the annotated source code and the syntax tree of the above program is listed below:

```
Source code: sample.txt
   1: read x;
        1: reserved word: read
        1: ID, name= x
        1: ;
   2: x := x * 10;
        2: ID, name= x
        2: :=
        2: ID, name= x
        2: *
        2: NUM, val= 10
        2: ;
   3: write x;
        3: reserved word: write
        3: ID, name= x
        3: ;
   4: y := x + 10;
        4: ID, name= y
        4: :=
        4: ID, name= x
        4: +
        4: NUM, val= 10
        4: ;
   5: write y;
        5: reserved word: write
        5: ID, name= y
        5: ;
   6: x := x + y * 10;
        6: ID, name= x
        6: :=
```

```
        6: ID, name= x
        6: +
        6: ID, name= y
        6: *
        6: NUM, val= 10
        6: ;
   7: write x;
        7: reserved word: write
        7: ID, name= x
        7: ;
   8: z := y + x / 10;
        8: ID, name= z
        8: :=
        8: ID, name= y
        8: +
        8: ID, name= x
        8: /
        8: NUM, val= 10
        8: ;
   9: write z;
        9: reserved word: write
        9: ID, name= z
        9: ;
        10: EOF

Syntax tree: sample.txt
  Read: x
  Assign to: x
    Op: *
      Id: x
      Const: 10
  Write
    Id: x
  Assign to: y
    Op: +
      Id: x
      Const: 10
  Write
    Id: y
  Assign to: x
    Op: +
      Id: x
      Op: *
        Id: y
        Const: 10
  Write
    Id: x
  Assign to: z
    Op: +
```

```
      Id: y
      Op: /
        Id: x
        Const: 10
  Write
    Id: z
```

RULES:

1. Obey honor code principles.
2. Read your homework carefully and follow the directives about the I/O format (data file names, file formats, etc.) and submission format strictly. Violating any of these directives will be penalized.
3. Obey coding convention.
4. Your **online submission** should include the following file and NOTHING MORE (no data files, object files, etc):

   A02_<Firstname>_<Lastname>_<student number>_parser.zip.

   IMPORTANT

   It should contain the java source code files, the Makefile, and the manifest file.
   Your source code should compile (using the latest javac compiler and jar tool) without any errors, otherwise you will not get any marks for Q.6.
   A sample program (suis.zip – not a parser) is posted on the moodle to give you an idea of writing a Makefile and a manifest file for your parser.
   Submissions without a Makefile or a manifest file will be penalized.
5. Do not use non-English characters in any part of your homework (in body, file name, etc.).
6. Answers to Q.1 – Q.5 will also be submitted online.