

CS 316 (Kong): TinyJ Assignment 1

To be submitted no later than: Tuesday, November 29. [Note: I expect euclid to be up until midnight that evening, but there is no guarantee that it will be: If euclid unexpectedly goes down after 6 p.m., the deadline will **not** be extended. If you try to submit after 6 p.m. that evening and find that euclid is down, you may have to make a **late** submission! Try to submit no later than noon that day, and on an earlier day if possible.] This assignment counts **1.5%** towards your grade if the grade is computed using rule A.

The TinyJ language is an extremely small subset of Java. Every valid TinyJ program is a valid Java program, and has the same semantics whether it is regarded as a TinyJ or a Java program. The syntax of TinyJ is given by the EBNF specification that is shown below. *In this EBNF specification each terminal is a token of TinyJ, and each nonterminal $\langle X \rangle$ denotes the set of all sequences of tokens that are syntactically valid for the TinyJ construct X .* In particular, a piece of source code is a *syntactically valid* TinyJ program if and only if its sequence of tokens belongs to the language generated by this EBNF specification. A piece of source code is a valid TinyJ program if and only if it is *both* a syntactically valid TinyJ program *and* a valid Java 8 program, with a few exceptions: TinyJ does not allow non-decimal (i.e., hexadecimal, octal, or binary) or long integer literals, underscores in integer literals, method name overloading, program arguments, printing of Boolean values, “return;” statements within the **main()** method, escape sequences other than `\n`, `\\`, and `\"`, and ints that are $\geq 2^{31} - 2^{16} = 2,147,418,112$.

Reserved words of TinyJ are shown in boldface in this EBNF specification. Some names used by Java library packages, classes, and their methods (e.g., **java**, **Scanner**, and **nextInt**) are reserved words of TinyJ, as is **main**. Otherwise, IDENTIFIER here means any Java identifier consisting of ASCII characters.

```
<program> ::= [<importStmt>] class IDENTIFIER '{' {<dataFieldDecl>
                                     <mainDecl> {<methodDecl>} '}'

<importStmt> ::= import java . util . Scanner ;
<dataFieldDecl> ::= static <varDecl>
<varDecl> ::= int <singleVarDecl> { , <singleVarDecl> } ;
              | Scanner IDENTIFIER = new Scanner '(' System . in ')' ;
<singleVarDecl> ::= IDENTIFIER { '[' ']' } [ = <expr3> ]
<mainDecl> ::= public static void main '(' String IDENTIFIER '[' ']' ')'
               <compoundStmt>
<methodDecl> ::= static ( void | int { '[' ']' } ) IDENTIFIER
                  '(' <parameterDeclList> ')' <compoundStmt>
<parameterDeclList> ::= [<parameterDecl> { , <parameterDecl> } ]
<parameterDecl> ::= int IDENTIFIER { '[' ']' }
<compoundStmt> ::= '{' { <statement> } '}'
<statement> ::= ; | return [<expr3>] ; | <varDecl> | <assignmentOrInvoc>
              | <compoundStmt> | <ifStmt> | <whileStmt> | <outputStmt>
<assignmentOrInvoc> ::= IDENTIFIER ( { '[' <expr3> ']' } = <expr3> ; | <argumentList> ; )
<argumentList> ::= '(' [<expr3> { , <expr3> } ] ')'
<ifStmt> ::= if '(' <expr7> ')' <statement> [else <statement>]
<whileStmt> ::= while '(' <expr7> ')' <statement>
<outputStmt> ::= System . out . ( print '(' <printArgument> ')' ;
                  | println '(' [<printArgument> ] ')' ;
                  )
<printArgument> ::= CHARSTRING | <expr3>
<expr7> ::= <expr6> { '|' <expr6> }
<expr6> ::= <expr5> { & <expr5> }
<expr5> ::= <expr4> { (== | !=) <expr4> }
<expr4> ::= <expr3> [ (> | < | >= | <=) <expr3> ]
<expr3> ::= <expr2> { (+ | -) <expr2> }
<expr2> ::= <expr1> { (* | / | %) <expr1> }
<expr1> ::= '(' <expr7> ')' | (+ | - | !) <expr1> | UNSIGNEDINT | null
              | new int '[' <expr3> ']' { '[' ']' }
              | IDENTIFIER ( . nextInt '(' ')' | [<argumentList> ] { '[' <expr3> ']' } )
```

This is the first of three TinyJ assignments. After completing all three assignments you will have a program that can compile any TinyJ program into a simple virtual machine code, and then execute the virtual machine code it has generated. (Execution should produce the same run-time behavior as you would get if you compiled the same TinyJ program using `javac` into a `.class` file and then executed that `.class` file using a Java VM.) There will be exam questions relating to the TinyJ assignments.

TinyJ Assignment 1 will not deal with compilation of TinyJ programs, nor with execution of virtual machine code, but only with *syntax analysis* of TinyJ programs. The goal of TinyJ Assignment 1 is to complete a program that will:

- (a) determine if the sequence of tokens of its input file belongs to `<program>` (as defined by the above EBNF rules), and
- (b) output a parse tree of the sequence of tokens of its input file, if that sequence belongs to `<program>`.

Regarding (a), note that the sequence of tokens of the input file belongs to `<program>` if, and only if, the input file is a *syntactically* valid TinyJ program. However, a syntactically valid TinyJ program may still contain errors like “undeclared variable” or “array index out of range”. A “sideways” representation of ordered trees, described below, will be used for (b).

A Sideways Representation of an Ordered Rooted Tree T

If T has just one node, then	representation of T = the unique node of T
Otherwise,	representation of T = the root of T
	representation of the 1 st subtree of the root of T
	representation of the 2 nd subtree of the root of T
	...
	representation of the last subtree of the root of T
	... node has no more children

In this sideways representation, sibling nodes always have the *same* indentation, but each non-root node is further indented than its parent; *the indentation of a node is proportional to the depth of that node in the tree*. Here are the “ordinary” and the “sideways” representations of a tree:

<pre> <expr4> <expr3> / \ <expr2> + <expr2> / \ / \ \ <expr1> <expr1> * <expr1> UNSIGNEDINT IDENTIFIER UNSIGNEDINT </pre>	<pre> <expr4> <expr3> <expr2> <expr1> UNSIGNEDINT ... node has no more children ... node has no more children + <expr2> <expr1> IDENTIFIER ... node has no more children * <expr1> UNSIGNEDINT ... node has no more children ... node has no more children ... node has no more children ... node has no more children </pre>
--	---

How to Install the TinyJ Assignment 1 Files on euclid, venus, and (optionally) Your PC


Do 1 – 5, and optionally 6 – 11, before our class on **Wednesday, November 16**. (See “Files That May Be Discussed in Class ...” on p. 3.) Remember that Unix/Linux file and command names are case-sensitive when following the instructions below!

1. Login to euclid and enter: `/users/kong300/316/TJ1setup` [The 1 in TJ1setup is the digit 1, not the letter l.]
2. Wait for the line “TJ1setup done” to appear on the screen, and then enter the following command on *euclid*:
`java -cp TJ1solclasses:. TJ1asn.TJ CS316ex12.java 12.sol`
 Note the period after the *colon* in this command. This command executes my solution to this assignment with CS316ex12.java as the input file and 12.sol as the output file. A listing of CS316ex12.java should be displayed on the screen, and 12.sol should contain a sideways representation of the program’s parse tree afterwards. **There should not be any error message**. To view the tree, you can use `less 12.sol` or just open 12.sol in an editor.
3. Logout from euclid and login to *venus*.
4. Enter the following on venus: `/home/faculty/ykong/TJ1setup`
 [Again, the 1 in TJ1setup is the digit 1, not the letter l.]
5. Repeat step 2 above on venus.

The following 6 steps are needed *only if* you are interested in doing TinyJ assignments on your PC rather than euclid or venus. (Regardless of where you do your work, you must submit on *euclid*.) These steps assume your PC is connected to the Internet.


6. Open a **command-line window*** on your PC and enter the following at its prompt: `javac -version`

*You can open a command-line window on your Windows PC as follows:

1. Type **Win-r** (i.e., hold down the Windows-logo key and type r) to open the Run dialog box.
2. Type `cmd` into the Open: textbox and press .

If you get an error message after entering `javac -version`, or if the version number that is printed is older than **1.8.0**, then download and install a new version of the Java JDK—e.g., the Java SE Development Kit 19—from <https://www.oracle.com/technetwork/java/javase/downloads/index.html> and set the System PATH environment variable to include the directory that contains the compiler `javac.exe` and the program `jar.exe` that are part of the JDK you installed: For a typical installation of the Java SE Development Kit 19, `c:\program files\java\jdk-19\bin` is likely to be the directory that should be included in your System PATH. If you don't know how to add a directory to your System PATH then see, e.g., <https://www.computerhope.com/issues/ch000549.htm> for instructions.[†]

[†]If you have difficulty with steps 1 – 3 of these instructions, try the following instead of those steps:

1. Type **Win-r** (i.e., hold down the Windows-logo key and type r) to open the Run dialog box.
2. Type `sysdm.cpl`, 3 into the Open: textbox and press .

Note: If the System PATH already includes a directory for a *previous* Java installation, then move the *new* directory up until it appears *before* all such directories: This is so you will use the *new* versions of `javac.exe` and `java.exe` by default.

7. In the command-line window, enter the following: `mkdir c:\316java`

Note: You may use another location for your **316java** directory if you wish, but then you will need to replace `c:\316java` with the pathname of your **316java** directory in the above command and all other instructions that refer to `c:\316java`, *including any such instructions you may receive in the future!*

8. Make `c:\316java` your working directory by entering the following in the command-line window: `cd /d c:\316java`
9. Use an scp or sftp client to download **TJ1asn.jar** from your home directory on *venus* or *euclid* into the `c:\316java` folder on your PC. So if `c:\316java` is your working directory in the command-line window (see step 8), then you can download **TJ1asn.jar** by entering the following in the command-line window:

```
scp xxxxx316@euclid.cs.cuny.edu:TJ1asn.jar .
```

Here **xxxxx316** means your *euclid* username. *Note the space followed by a period at the end of this command!*

10. Enter the following *two* commands in the command-line window: `jar xvf TJ1asn.jar`
`javac -cp . TJ1asn/TJ.java`

11. Enter the following command in the command-line window:

```
java -cp "TJ1solclasses;." TJ1asn.TJ CS316ex12.java 12.sol
```

The comments on step 2 also apply here, except that a *semicolon* rather than a colon precedes the period and you can use **more 12.sol** (instead of **less 12.sol**) to view the tree. If you are unfamiliar with the **more** command, see, e.g.: <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/more>


Files That May Be Discussed in Class Starting on Wednesday, November 16

From your **TJ1asn** directory on euclid:

```
OutputFileHandler.java.txt  Parser.java.txt  SourceFileErrorException.java.txt  TJ.java.txt
```

From your **TJ1lexer** directory on euclid (the **l** in **TJ1lexer** is the *letter* l, not the digit 1):

```
LexicalAnalyzer.java.txt  SourceHandler.java.txt  Symbols.java.txt
```

These are the source files of the program, with **line numbers added**. The actual source files (without line numbers) are in the same directories and have the same names, but their extension is `.java`. *If you have done steps 6 – 11 above, you can find the same files in `c:\316java\TJ1asn` and `c:\316java\TJ1lexer` on your PC, and these files can be opened using, e.g., any of the editors recommended in the last paragraph of p. 4 of the Lisp Assignment 2 document. Otherwise, you can e-mail the files to yourself—e.g., you can send `TJ1asn/TJ.java.txt` to yourself by entering the following on euclid:*
`alpine -attach TJ1asn/TJ.java.txt your-email-address` *[After alpine starts up, enter  y to send the file.]*

How to Execute My Solution to This Assignment

Steps 1 and 4 put 16 files named `CS316exk.java` ($k = 0 - 15$) into your home directories on *euclid* and *venus*. These are all valid TinyJ source files. If you did step 10, it will have put copies of the same 16 files on your PC. You should be able to execute my solution to this assignment either on *euclid* or on *venus* by entering the following command:

```
java -cp TJ1solclasses:. TJ1asn.TJ TinyJ-source-file-name output-file-name
```

[Your current working directory has to be your home directory for this to work.] This also works in a command-line window on your PC if you have done steps 6 – 11, **except that you need a semicolon instead of a colon after** `TJ1solclasses` on a PC:

```
java -cp "TJ1solclasses;." TJ1asn.TJ TinyJ-source-file-name output-file-name
```

[Your working directory has to be `C:\316java` for this to work (see step 8).]

How to Do TinyJ Assignment 1

The file `TJlasm/Parser.java` is incomplete. It was produced by taking a complete version of that file and replacing parts of the code with comments of the following two forms:

```
/* ???????? */           or (in two places)  /* ????????
                                     default: throw ...
*/
```

To complete this assignment, replace every such comment in `TJlasm/Parser.java` with appropriate code, and recompile the file. On *venus* or *euclid*, you can use any text editor to edit the file. If you are working on your PC, do not use Notepad as your editor; I suggest you use one of the editors listed in the last paragraph on p. 4 of the Lisp Assignment 2 document. (For the second type of comment, the appropriate code should include the `default: throw ...` statement.)

Do **not** put `Parser.java` or `Parser.class` into any directory other than `TJlasm`. Do **not** change or move other `.java` and `.class` files.

To recompile `TJlasm/Parser.java` after editing it, enter the following command:

```
javac -cp . TJlasm/Parser.java
```

IMPORTANT: If you are doing this on *venus* or *euclid*, your current working directory has to be your home directory. If you are doing this on your PC (in a command-line window), your working directory has to be `c:\316java` (see installation step 8); otherwise `javac` will not be able to find other classes that are used in `Parser.java`!

As stated on p. 3 of the first-day announcements, keep a backup copy of your edited version of `Parser.java` on *venus* and another backup copy on a different machine.

How to Test Your Solution

To test your completed version of `Parser.java`, first recompile it using `javac -cp . TJlasm/Parser.java` and then execute `TJlasm.TJ` with each of the 16 files `CS316exk.java` ($k = 0 - 15$) as the TinyJ source file and `k.out` as the output file, as follows: `java -cp . TJlasm.TJ CS316exk.java k.out`

If you are doing this on *venus* or *euclid*, your current working directory has to be your home directory. If you are doing this on your PC (in a command-line window), your working directory has to be `c:\316java` (see installation step 8).

If your program is correct then in each case the output file `k.out` should be identical to the output file `k.sol` that is produced by running my solution with the same source file as follows:

```
java -cp TJlsolclasses:. TJlasm.TJ CS316exk.java k.sol      [on euclid or venus]
java -cp "TJlsolclasses;." TJlasm.TJ CS316exk.java k.sol    [on a PC]
```

On *euclid* and *venus*, use `diff -c` to compare the output files produced by your and my solutions. (This outputs a report of the differences, if any, between the two files.) On a PC, use `fc.exe /n` to compare files. For example, the commands `diff -c k.sol k.out > k.dif` [on *venus* or *euclid*] and `fc.exe /n k.sol k.out > k.dif` [on a PC] output to `k.dif` the differences between `k.sol` and `k.out`. (If your solution is correct, there should be no differences.)

How to Submit a Solution to This Assignment

This assignment is to be submitted *no later than* the due date stated on p. 1. [Note: If *euclid* unexpectedly goes down after 6 p.m. on this due date, the deadline will not be extended. Try to submit no later than noon that day, and on an earlier day if possible.]

To submit:

1. Add a comment at the beginning of your completed version of `Parser.java` that gives your name and the names of the students you worked with (if any). As usual, you may work with up to two other students, but see the remarks about this on p. 3 of the first-day announcements document.
2. Leave your final version of `Parser.java` on *euclid* in your `TJlasm` directory, so it replaces the original version of `Parser.java`, before midnight on the due date. When two or three students work together, each of the students must leave his/her completed file in his/her directory. If you are working on *venus* or your PC, you can transfer `Parser.java` to your `TJlasm` directory on *euclid* by following the instructions on the next page.
3. Be sure to test your submission on euclid—see the **How to Test Your Solution** instructions above. Note that if your modified version of `Parser.java` cannot even be compiled without error on *euclid*, then you will receive no credit at all for your submission!

IMPORTANT: Do NOT open your submitted file `Parser.java` in an editor on *euclid* after the due date, unless you are resubmitting a corrected version of your solution as a *late* submission. Also do not execute `mv`, `chmod`, or `touch` with your submitted file as an argument after the due date. (However, it is OK to view a submitted file using the `less` file viewer after the due date.) Remember that, as stated on page 3 of the first-day announcements document, you are required to keep a backup copy of your submitted file on *venus*, and another copy elsewhere—see the final paragraph on the next page.

How to Transfer `TJlasn/Parser.java` from `venus` or a PC to `euclid`'s `TJlasn` Directory

The following instructions assume that `xxxxx316` is your username on `euclid`.

If you are working on `venus`, and your current working directory is your home directory, enter the following command to transfer `TJlasn/Parser.java` to your `TJlasn` directory on `euclid`:

```
scp TJlasn/Parser.java xxxxx316@euclid.cs.qc.cuny.edu:TJlasn
```

You will be asked to enter your `euclid` password.

If you are working on a PC and your working directory is `c:\316java`, then you can transfer the file `TJlasn/Parser.java` into your `TJlasn` directory on `euclid` by entering the following command in a command-line window:

```
scp TJlasn/Parser.java xxxxx316@euclid.cs.qc.cuny.edu:TJlasn
```

You will be asked to enter your `euclid` password.

IMPORTANT REMINDER: After you have transferred `TJlasn/Parser.java` to your `TJlasn` directory on `euclid`, be sure to *test your code on euclid*—see the **How to Test Your Solution** instructions on the previous page. (It is not enough to have tested your code on `venus` or your PC, because testing on a machine other than `euclid` does not test the file you actually submitted!)

As stated on page 3 of the first-day announcements document, you are required to keep a backup copy of your submitted file on `venus`, and another copy elsewhere. You can enter the following two commands on `euclid` to email a copy of your submitted file to yourself and to put a copy of the file on `venus`:

```
echo . | mailx -s "copy of submission" -a TJlasn/Parser.java $USER
```

```
scp TJlasn/Parser.java your venus username@mars.cs.qc.cuny.edu:
```

The colon at the end of the second command is needed!