Assignment 1 Guidelines

Bernd Burgstaller
Yonsei University

Attention!

These are guidelines from the lecture on how to work on Assignment 1.

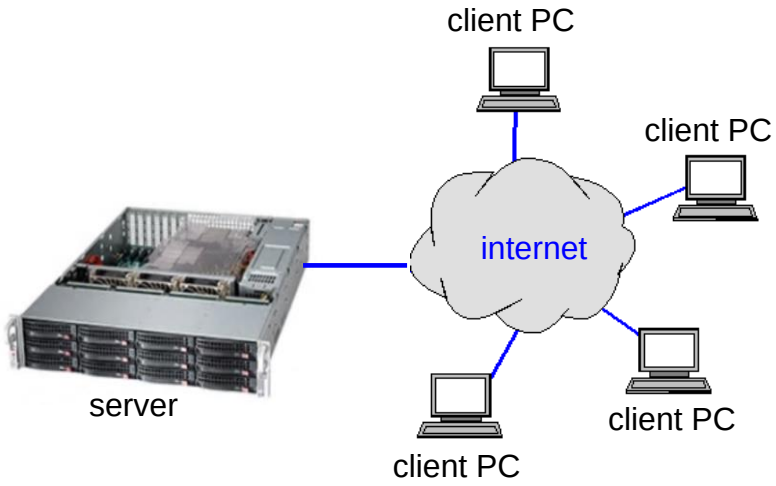
Contents:

- server access
- technical details about the code skeleton provided to you

The **assignment specification itself** can be found on LearnUs in the ``Assignments & Homeworks'' folder.

Outline

- Server access
  - Setting your password
  - Remote log-in using ssh:
    - Windows
    - macOS
    - Linux



- Assignment 1
  - Scanner/Parser Interaction
  - The provided skeleton compiler
  - Hand-crafting a scanner
  - Token representation
  - Maximal munch

Server

Our server for this course will be **csi4104.cs.yonsei.ac.kr**

- You can program your assignments on the server (recommended).
- Or any computer that has a Java compiler and a Java virtual machine (JVM) installed.
- We're going to provide a VirtualBox Linux virtual machine image that you can run on your own computer.
  - Contains the development environment.
  - Will be announced shortly on LearnUs.

- **Submission and marking will be done on the server.**
  - You must make sure that your programs compile and execute on the server!
  - You must submit your programs on the server.

Initial password

- Your account on the server has been set up.

- **Login IDs and initial passwords will be provided to you on LearnUs.**
  - Upon your first login, you'll be asked to change your password.

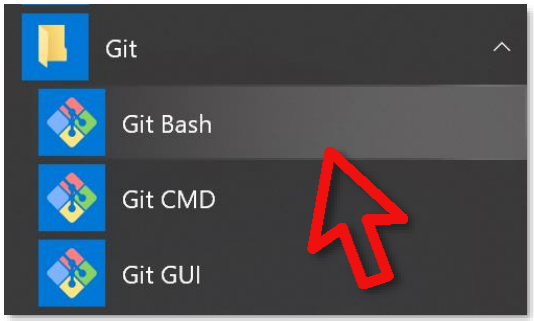
- **Please pick a strong password!**
  - E.g., use a sentence that is easy to remember for you and use the initial letters of the words of that sentence as your password. E.g.,
    My cat has four legs, one tail and one head! →
  - **Changing your password:** log in on the server, and type ``passwd <enter>''. You will be prompted for your initial password. After that, you have to provide your new password.

- Please direct enquires about the server and initial login to our TAs.

  → Please find their contact details in the ``Course Introduction'' slides.

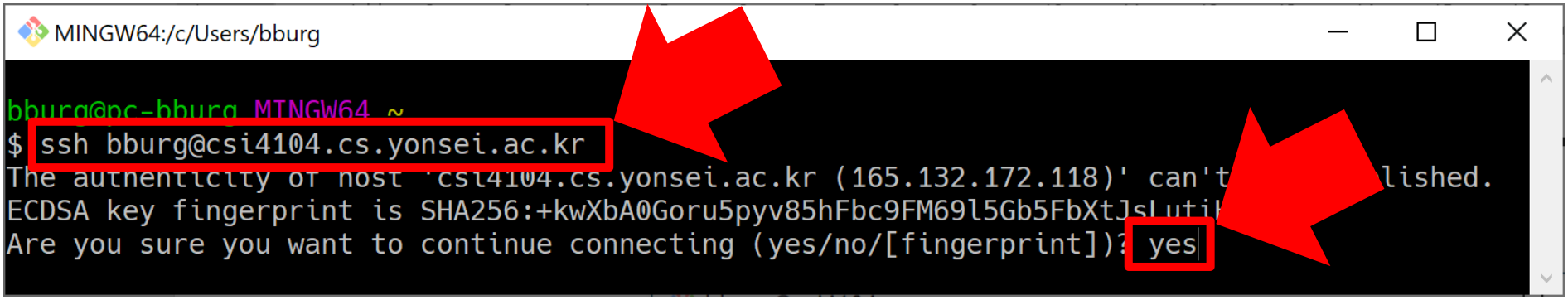
Windows: server access using Git Bash and ssh (Step 1)

- Download and install the free Git version-control system from https://git-scm.com/
- Git comes with **Git Bash**, which is a Bash shell command-line interpreter.
  - Git Bash provides the **ssh utility**.



- Start **Git Bash** on your computer
  - The next slides tell you how to proceed...

Windows: server access using Git Bash and ssh (Step 2a)



1) In the Git Bash terminal window, type in the below ssh command. ``you'' is the **login ID** you've been provided with, for example ``u2000828488''.

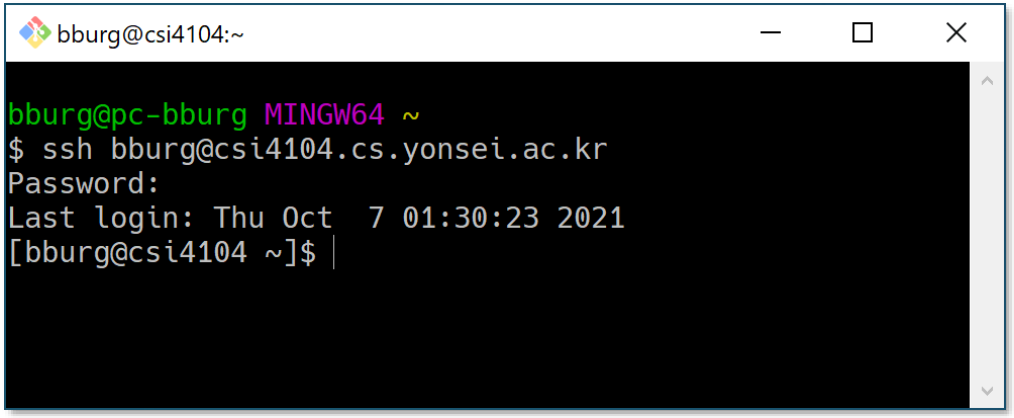
```
ssh you@csi4104.cs.yonsei.ac.kr
```

   - Note: **upon the first login**, ssh does not know the server and requires you to confirm its authenticity.
   - Please type ``yes'' as shown above.

2) When prompted, type in the **password** you've been provided with.

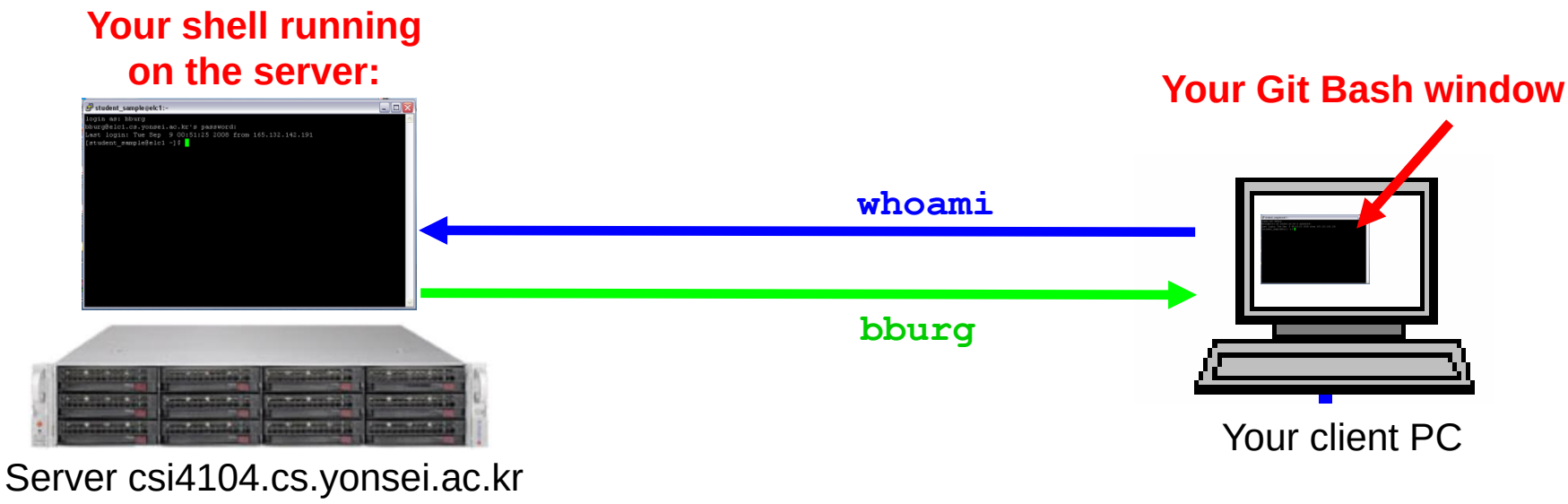
3) After verification of your **login ID and password**, you're logged in on the server.

Windows: server access using Git Bash and ssh (Step 2b)



- Upon subsequent logins, you won't be asked to confirm the authenticity of the server anymore.
  - Only your login ID and password are required. ☺

ssh, behind the scenes...

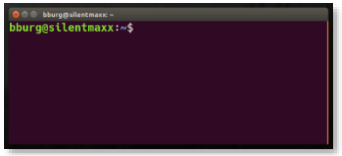


- **Git Bash** and the **ssh client** execute on your computer.
- A **shell** executes on the server.
- The ssh client maintains a network connection to the shell on the server.
  - Your **input** is sent to the shell on the server
  - The shell's **output** is sent back to your ssh client.
- You can do everything in that server shell that you could do if you very directly sitting in front of the server:
  - Editing files, compiling, executing programs, debugging, ...

ssh clients for various operating systems

- Windows
  - Included with the Git client for Windows (discussed on previous slides)
  - Putty (perhaps a bit archaic)

- Linux:
  - ssh available as part of the core OS



- macOS:
  - ssh available as part of the core OS
  - Make sure to use the free iTerm2 terminal replacement to avoid the instabilities of Apple's macOS iTerm program.

Copying files between your computer and the server

- With ssh comes a program called **scp** (``secure copy''); they belong to the same family of ``secure'' protocols.

- **From your computer:** copying file `foo.txt` from your computer to your home directory on the server:

```
scp foo.txt <username>@csi4104.cs.yonsei.ac.kr:
```

- **From your computer:** copying file `foo.txt` from your home-directory on the server to your computer:

```
scp <username>@csi4104.cs.yonsei.ac.kr:foo.txt .
```

- No other types of file-transfer are allowed!

- For more details on the scp command, please refer to our **Linux Quick-start Guide** in the ``Resources'' folder on LearnUs.

Migrating text files between Unix and Windows

- Windows uses ``**CR+LF**'' encoding for the ``newline'' character.
- But Linux uses ``**LF**'' only.
- Thus, the Windows default editor (Notebook) is not a good choice for editing files from a Linux system (you'll get one very long line).

- You're recommended to use another editor, such as Notepad++ on Windows.

Caution:

- It is **risky** to create a file on one operating system (Windows) and continue editing it on another operating system (Linux). Chances are that you end up with a **mix** of ``CR+LF'' and ``LF'' newline encodings.
- This will give **unexpected results** with your scanner, if you are matching ``\n'' (the C/Java newline escape sequence)!
- If you migrate a text file (such as MiniC source-code) from Windows to Linux, then use the **dos2unix** command to re-encode the file (this will fix the CR/LF problem).

For more information: https://en.wikipedia.org/wiki/Newline

Resource Limits

Our course environment is shared between 100+ students.

Some measures to prevent things from going out of bounds:

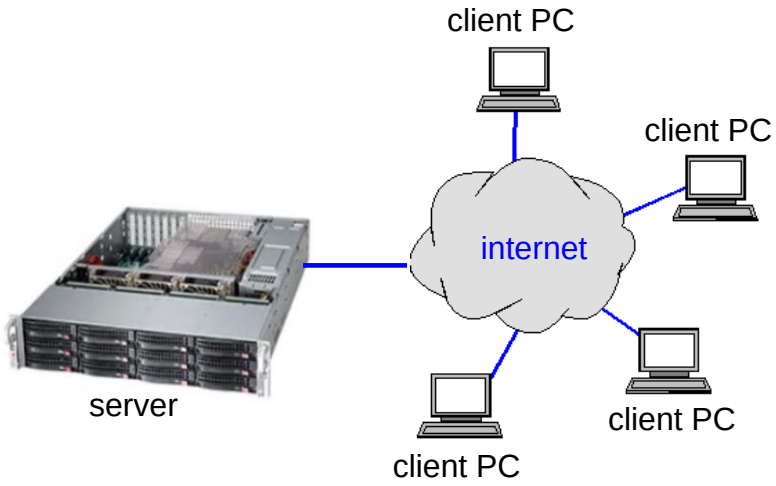
- **Disk Quotas:** 2 GB of disk space per user.
  - By far enough for the assignments...
  - The server maintains disk quotas to prevent users from using up all disk space.
  - If the server does not allow you to create files any more, please clean up your home directory.
  - Note: deleting files does not immediately effect your quotum!
- **Number of processes:** 100 processes per user.
  - Prevents ``fork bombs'' and buggy programs from making the system un-usable (by generating thousands of processes).
  - Note: sometimes crashed processes are still ``around''. Use **`ps -eLf | grep <yourlogin>`** to learn about all your processes. **man kill** tells you how to terminate processes.
- **Main Memory:** ~400MB per user.

More information, advice, help

- The Linux **man pages** provide very detailed information
  - on ssh, scp, ... Example: **`man ssh`**
- **Our Linux Quick-start guide**
  - Available on LearnUs in the ``Resources'' folder.
- **Our LearnUs Q&A board**
- **Our TAs**
  - For questions on our course environment
  - lost passwords, unable to login, ...
- **Me**, your instructor.

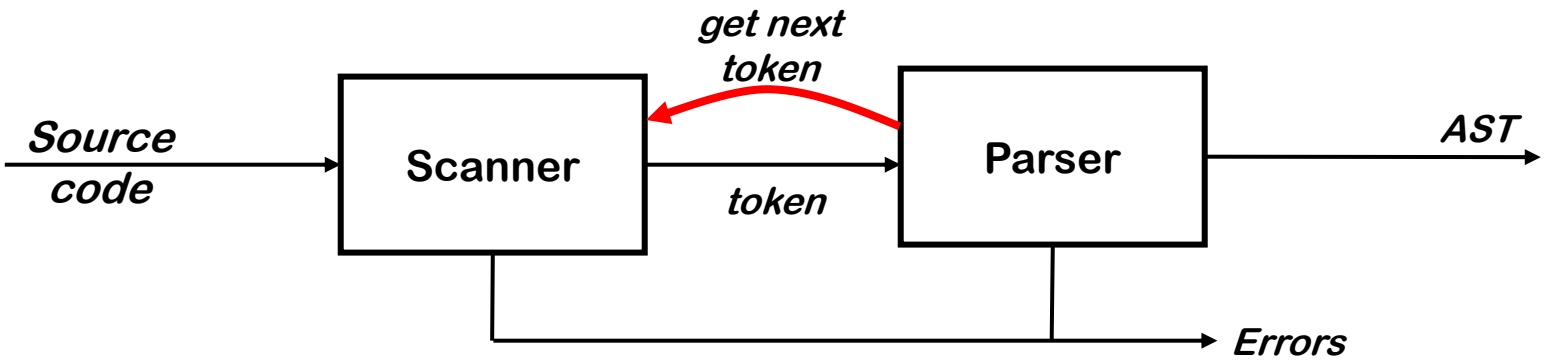
Outline

- Server access
  - Setting your password ✓
  - Remote log-in using ssh:
    - Windows ✓
    - macOS ✓
    - Linux ✓



- Assignment 1
  - Scanner/Parser Interaction
  - The provided skeleton compiler
  - Hand-crafting a scanner
  - Token representation
  - Maximal munch

Scanner/Parser Interaction



Scanner

- operates as a subroutine that is called by the parser.
- Parser calls ``**`get next token`**'' when it needs a new token from the input stream.
- Unlike Section 2.7 of the Dragon book, we store token information (such as the lexeme) with the token object itself (instead of using a symbol table).

Tokens

The tokens of our MiniC language are classified into *token types*:

- identifiers: **`i`**, **`j`**, **`initial`**, **`position`**, …
- keywords: **`if`**, **`for`**, **`while`**, **`int`**, **`float`**, **`bool`**, …
- operators: **`+ - * / <= &&`** …
- separators: **`{ } ( ) [ ] ; ,`**
- literals
  - integer literals: **`0`**, **`1`**, **`22`**, …
  - float literals: **`1.25 1. .01 1.2e2`** …
  - bool literals: **`true`**, **`false`**
  - string literals: **``string literals'', ``compiler'',** ...

- The exact token set depends on the given programming language. Pascal and Ada use ":=" for assignment, C uses "=".
- Natural languages also contain different kinds of tokens (words): verbs, nouns, articles, adjectives, … The exact token set depends on the natural language in question.

Lexemes (Spellings of Tokens)

- The lexeme of a token: the character sequence forming the token.

Source code	Token Type	Lexeme
main	ID	main
foobar	ID	foobar
+	plus operator	+
<=	less-equal operator	<=
100	INTLITERAL	100
1.2e2	FLOATLITERAL	1.2e2
true	BOOLLITERAL	true

- Need a formal notation for tokens.
  → REs, NFAs, DFAs
- (As discussed in the lecture already.)

Regular expressions for integers and reals in C

Integers:

digit: 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

INTLITERAL: digit digit*

Reals:

FLOATLITERAL: digit* fraction exponent?
| digit^{+}.
| digit^{+}.?exponent

fraction: .digit^{+}

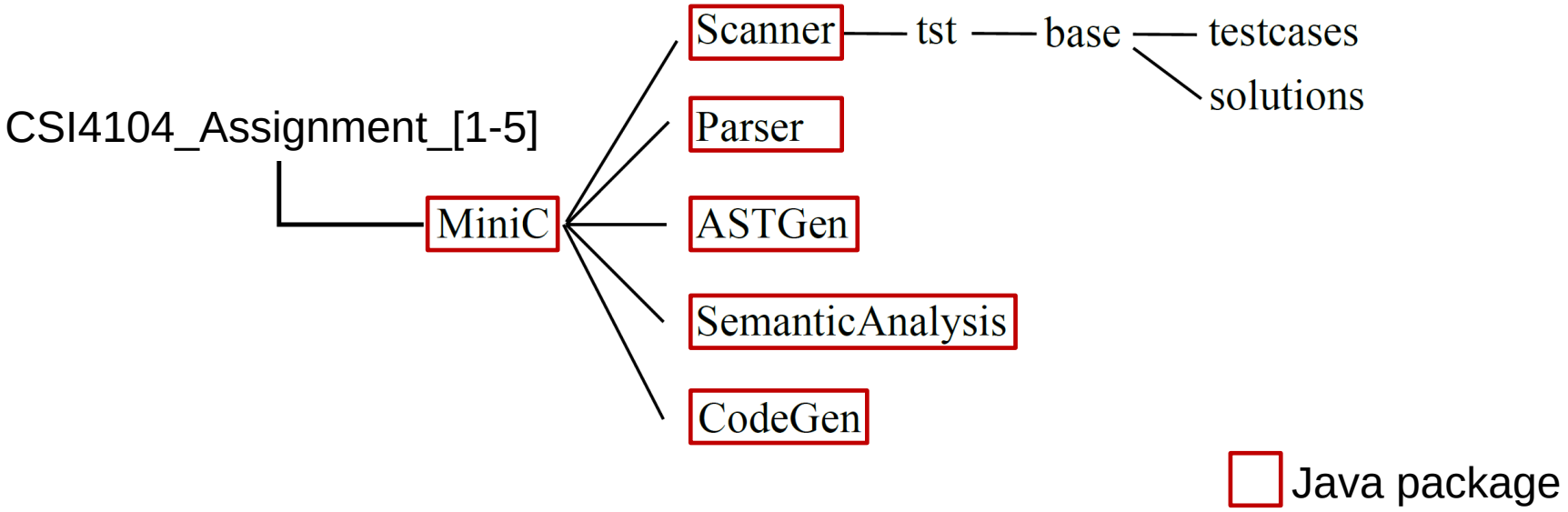
exponent: (E|e)(+|-)?digit^{+}

Please refer to the MiniC language specification for details

Source Organization of our MiniC Compiler

You are provided with a skeleton compiler.

- Through assignments, we will extend the skeleton to a full compiler.
- Below directory structure reflects the **Java package structure** of our compiler:
  - MiniC is the **driver** of the compiler, it invokes the sub-components (scanner, ...)



The MiniC skeleton compiler is provided on the server

- File /opt/ccugrad/Assignment1/CSI4104_Assignment_1.tgz
- At this stage (Assignment 1), only the 'Scanner' subdirectory is populated with files...

Source Organization of our MiniC Compiler

- **Scanner.java**: a skeleton scanner (to be completed by you)
- **Token.java**: The class for representing all MiniC tokens
  - class `Token` already provided for you
  - able to distinguish between identifiers and keywords
- **SourceFile.java**: source-file handling
- **SourcePos.java**: the class for defining the position of a token in the source file.
- **MiniC.java**: the driver program of our MiniC compiler. At the moment the driver contains a loop to repeatedly call the scanner for the next token:

```
Token t;
scanner  = new Scanner(source);
scanner.enableDebugging();
do {
    t = scanner.scan(); // scan 1 token
} while (t.kind != Token.EOF);
```

Design Issues in Hand-Crafting your Scanner

- What are the tokens of the language? – see Token.java
- Are keywords reserved? – yes, as in C and Java
- How to distinguish identifiers from keywords? – see Token.java
- How to handle the end of file? – return a special Token: Token.EOF
- How to represent tokens: see Token.java (class Token)
- How to handle whitespace and comments: throw them away
- How to structure your scanner? – see Scanner.java
- What to do in case of lexical errors? – see the description of Assignment 1
- How many characters of **look-ahead** are needed to represent a token?
  - Described in the following slides

How to represent a token?

Token	Representation
Counter1	`new Token(Token.ID, "Counter1", src_pos)`
12	`new Token(Token.INTLITERAL, "12", src_pos)`
1.2	`new Token(Token.FLOATLITERAL, "1.2", src_pos)`
+	`new Token(Token.PLUS, "+", src_pos)`
;	`new Token(Token.SEMICOLON, ";", src_pos)`

- **`src_pos`** is an instance of the class **`SourcePos`**:
  - **`StartCol`**: the column in the input file where the token starts
  - **`EndCol`**: the column where the token ends
  - **`StartLine=EndLine`**: number of the line in the input file where the token occurs
    Note: with MiniC, tokens cannot span multiple lines!

The Structure of a Hand-Written Scanner

```
public final class scanner {

   int scanToken (void) {

    // skip whitespace characters and comments...
    // produce the next token:
    switch (currentChar) {
    case '+':
         takeIt() and return token representation for '+'
    case '<':
         takeIt();   // adds '<' to the current lexeme
         if (currentChar == '=') {
             takeIt(); // adds '=' to the current lexeme
             return token representation for '<='
         } else {
             return token representation for '<'
```

Look-ahead

The Structure of a Hand-Written Scanner (cont.)

```
    case '.':
        // attempt to recognize a float:
        ...
    default:
        takeIt();
        return Error token
    }
}
...
return new Token (kind, lexeme, src_pos);
```

- You need to figure out how to efficiently recognize all MiniC tokens:
  - identifiers, keywords, literals, aso.

My takeIt() Method in the Scanner Class

```
private void takeIt() {

    currentLexeme.append(currentChar);

    currentChar = sourceFile.readChar();

    // increment line number counter, if necessary

    // increment column counter
}
```

Maintaining 2 Invariants Across Calls to the Scanner

Every time the scanner is called

to return the next token, two invariants hold:

1) **`currentChar`** is pointing either to the beginning
   - of whitespace, or
   - a comment, or
   - a token.
2) The scanner always returns the longest possible match in the remaining input (maximum munch):

Def. Invariant: a condition which does not change during program execution.

Note: our two scanner invariants hold only **across** calls.

Input	Tokens
==	"==" and not "=" and "="
//	end-of-line comments, not "/" and "/". Note: we throw away comments in the scanner.

Lexical Errors and Look-Ahead

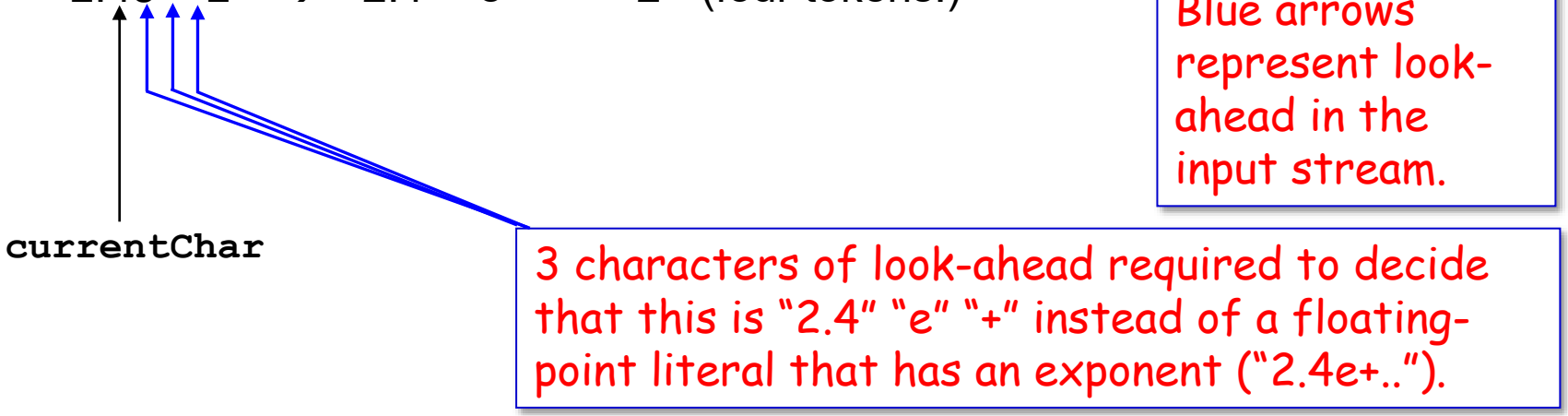
```
FLOATLITERAL        ::=   digit* fraction exponent?
                     |    digit+ .
                     |    digit+ .? exponent
fraction            ::=   . digit+
exponent            ::=   (e|E) (+|-)? digit+
```

- Example floating-point literals:

  2.3 4. .4 2e2 2E4 2.2E+2 2.4e-2 .1E3

- A tricky issue with floating-point literals:

  2.4e+ 2 → "2.4" "e" "+" "2" (four tokens!)



Language Design Problem: Fortran's Fixed Format

- Fortran's ``fixed format'' ignores blanks in the input
  - Motivated by inaccuracies of punchcards
- Example: `VAR1` is the same as `V AR1`

- `DO 5 I = 1,25`
- `DO 5 I = 1.25`

- The first is a DO-loop: `DO 5 I = 1 , 25`
- The second is an assignment: `DO5I = 1.25`

- The scanner cannot decide between the DO-loop and the assignment until after "`,`" has been reached.

Language Design Problem 2: Keywords Used As Identifiers

- PL/1 allowed the use of identifiers as keywords.
- `IF THEN THEN THEN = ELSE; ELSE ELSE = THEN`
- It becomes hard to decide how to label lexemes.

Language Design Problem 3

C++ nested template instantiations

- `vector<vector<int>> my_vector`
- `vector < vector < int >> my_vector`

Suggested Reading/Browsing

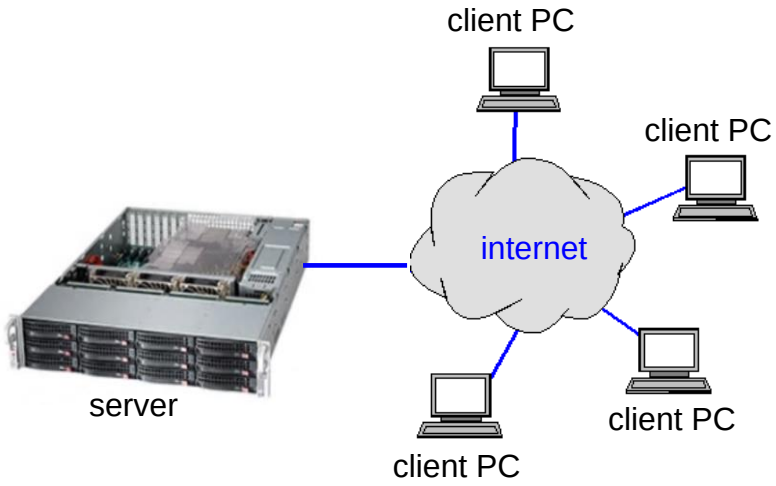
- The MiniC language specification will be provided on LearnUs.
  In the “Assignments” section.

- The Assignment 1 specification will be provided on LearnUs.
  In the “Assignments” section.

- The skeleton compiler is already available on the server.
  In file /opt/ccugrad/Assignment1/CSI4104_Assignment_1.tgz

Outline

- Server access
  - Setting your password ✓
  - Remote log-in using ssh:
    - Windows ✓
    - macOS ✓
    - Linux ✓



- Assignment 1
  - Scanner/Parser Interaction ✓
  - The provided skeleton compiler ✓
  - Hand-crafting a scanner ✓
  - Token representation ✓
  - Maximal munch ✓