Lecture 2: Lexical Analysis

- Register yourself electronically using the “account/teams/grades” link on the class home page.
- Please register yourself in this course on Piazza (see home page).
- Homework #1 is now available on the class website. Due Thursday at midnight (don’t worry—it’s short).
- Reminder: start forming teams of 2–4 people and register them.
A sequence of translations that each:
  - Filter out errors
  - Remove or put aside extraneous information
  - Make data more conveniently accessible.

**Strategy:** find tools that partially automate this procedure.

**For lexical analysis:** convert description that uses patterns (extended regular expressions) into program.
Tokens

- Token consists of *syntactic category* (like “noun” or “adjective”) plus *semantic information* (like a particular name).

- Parsing (the “customer”) only needs syntactic category:
  - “Joe went to the store” and “Harry went to the beach” have same grammatical structure.

- For programming, semantic information might be text of identifier or numeral.

- Example from Notes:

```plaintext
if(i== j)
    z = 0; /* No work needed */
else
    z = 1;

⇒ IF, LPAR, ID("i"), EQUALS, ID("j"), RPAR, ID("z"), ASSIGN, INTLIT("0"), SEMI, ELSE, ID("z"), ASSIGN, INTLIT("1"), SEMI
```
Classical Regular Expressions

- Regular expressions denote formal languages, which are sets of strings (of symbols from some alphabet).
- Appropriate since internal structure not all that complex yet.
- Expression $R$ denotes language $L(R)$:
  - $L(\epsilon) = L(\varepsilon) = \{\varepsilon\}$.
  - If $c$ is a character, $L(c) = \{c\}$.
  - If $R_1, R_2$ are r.e.s, $L(R_1 R_2) = \{x_1x_2 | x_1 \in L(R_1), x_2 \in L(R_2)\}$.
  - $L(R_1 | R_2) = L(R_1) \cup L(R_2)$.
  - $L(R^*) = L(\epsilon) \cup L(R) \cup L(R R) \cup \cdots$.
  - $L((R)) = L(R)$.
- Precedence is '*' (highest), concatenation, union (lowest). Parentheses also provide grouping.
Abbreviations

- **Character lists**, such as `[abcf-mxy]` in Java, Perl, or Python.
- **Negative character lists**, such as `[^aeiou]`.
- **Character classes** such as `. (dot), \d, \s` in Java, Perl, Python.
- $L(R^+) = L(RR^*)$.
- $L(R?) = L(\epsilon|R)$.
Extensions

● "Capture" parenthesized expressions:
  - After \( m = \text{re.match}\left( r'^\s*\(\d+\)\s*,\s*\(\d+\)\s', \, '12,34'\right) \), have \( m\text{.group}(1) == '12', \) m.group(2) == '34'.

● Lazy vs. greedy quantifiers:
  - \( \text{re.match}\left(r'^\(\d+\).*', \, '1234ab'\right) \) makes group(1) match '1234'.
  - \( \text{re.match}\left(r'^\(\d+?\).*', \, '1234ab'\right) \) makes group(1) match '1'.

● Boundaries:
  - \( \text{re.search}\left(r'^\(\^abc|qef\)'\right), L) matches \( abc \) only at beginning of string, and qef anywhere.
  - \( \text{re.search}\left(r'^\(?m\)\(\^abc|qef\)'\right), L) matches \( abc \) only at beginning of string or of any line.
  - \( \text{re.search}\left(r'^rowr(?=baz)'\right), L) matches an instance of 'rowr', but only if 'baz' follows (does not match baz).
  - \( \text{re.search}\left(r'^(?<rowr)baz)'\right), L) matches an instance of 'baz', but only if immediately preceded by 'rowr' (does not match rowr).

● Non-linear patterns: \( \text{re.search}\left(r'^\(\S+\),\1'\right), L) matches a word followed by the same word after a comma.
An Example

SL/1 “language”:

+    -    *    /    =    ;    ,    (    )    <    >

>=    <=    -->

if    def    else    fi    while

identifiers

decimal numerals

Comments start with # and go to end of line.

(Review of programs in Chapter 2 of Course Notes.)
Problems

• Decimal numerals in C, Java.
• All numerals in C, Java.
• Floating-point numerals.
• Identifiers in C, Java.
• Identifiers in Ada.
• Comments in C++, Java.
• XHTML markups.
• Python bracketing.
Some Problem Solutions

• Decimal numerals in C, Java: \( 0 | [1-9] [0-9]* \)

• All numerals in C, Java: \([1-9] [0-9]+ | 0 [xX] [0-9a-fA-F]+ | 0 [0-7]* \)

• Floating-point numerals: \((d+ d* \d* d+) ([eE] [-+] \d+) | [0-9] + [eE] [-+] \d+ \)

• Identifiers in C, Java. (ASCII only, no dollar signs): \([a-zA-Z\_] [a-zA-Z0-9]* \)

• Identifiers in Ada: \([a-zA-Z] ([a-zA-Z0-9] | [a-zA-Z0-9]) \)

• Comments in C++, Java: //.*|\*/([^*]|\*[~/])\*/\+/
or, using some extended features: //.*|\*/(.|\n)*?\*/

• Python bracketing: Nothing much you can do here, except to note blanks at the beginnings of lines and to do some programming in the actions.