Today

- Language Implementation (overview)
- Lexical Analysis

Assignments

- HW-10, R-10 due Thurs
- HW-11 out (no more reading assignments\(^(*)\))
- \(^(*)\) Ch. 3 & Ch. 4 cover basic implementation information
• Example of “separation of concerns”
  – each stage does a specific task
  – makes compiler implementation easier to manage
• Our plan is to discuss the “front end” steps ...
  – lexer, parser, semantic analysis (e.g., type checking)
Lexical Analysis: Tokens

Tokens are the smallest meaningful units of a program

Some examples:

- Special words (“reserved” words)
  - int, if, while, new, class, public, and so on
- Operators
  - +, =, ==, <=, and so on
- Identifiers
  - variable names, function names, class names, etc.
- Constants (Literals)
  - int, float, double, Boolean, string values

White space and comments are (usually) not tokens

- some exceptions such as Python

Tokens include a type and a “lexeme”

- the lexeme is just the token’s value in the source file
- e.g., in the statement: x = 42;
  - the token types might be ID, ASSIGN, INT, SEMICOLON
  - and the corresponding lexemes “x”, “=” “42”, and “;”
- for some tokens, the lexemes are needed for program execution
  - e.g., the variable name of the identifier ("x") and the integer value “42”
Lexical Analysis Basics

• Goal is to simplify syntax analysis (parsing) ... and detect errors early

• The basic idea:

Source Code: int fun() { int x = 0; return x; }

Token Sequence: INT, ID("foo"), LPAREN, RPAREN, LBRACKET,
INT, ID("x"), ASSIGN, CONST("0"), SEMICOLON,
RETURN, ID("x"), SEMICOLON,
RBRACKET

– Source code is converted to a sequence (a stream) of tokens
– Removes non-tokens (white space, comments)
– Tokens stored with with their corresponding line and column numbers

A Lexer is implemented using either

• a lexical analyzer tool (Lex, Flex, JFlex, ...)
• or as an ad hoc program (hand written)

Lexer usually called one token at a time

Source Code

read more input

Scanner

nextToken() 

Parser
**Exercise**: With a partner, give the **token sequence** (token types and lexemes) for the following code snippets.

Snippet 1:

```
println("Hello World!");
```

Snippet 2:

```
while x != y do
    if x == y then
        println(x);
    else if x < y then
        y = y - x;
    else
        x = x - y;
    end
end
```