Lecture 5:

- Quiz 1
- Lexical Analysis

Announcements:

- HW-1 out

Check in: Give the token sequence (token type, lexeme, line, column) for the following MyPL code snippets. Assume the token types:

```
ASSIGN, ID, INT_VAL, LPAREN, RPAREN, LBRACE, RBRACE,
LESS_EQ, PLUS, STRING_VAL, WHILE, EOS, SEMICOLON
```

Snippet 1:

```
print("Hello World!")
```

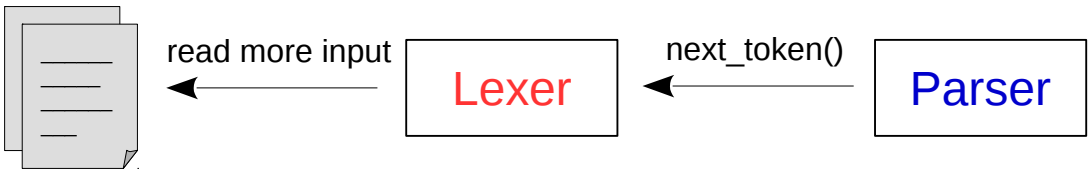
Snippet 2:

```
int x = 0;
while (x <= 10) {
  x = x + 2;
}
```

A Lexer is implemented using either

- a lexical analyzer tool (e.g., Lex, Flex, JFlex, ...)
- or as an *ad hoc* program (hand written) ... we'll do this!

The Lexer usually called one token at a time ... like an iterator



- the parser asks the lexer for the next token
- the lexer reads just enough from the source code to create a token
- the token (type, lexeme, line & column number) returned to the parser

Lexer only detects errors in forming tokens ... for example:

- unexpected characters/symbols (like an exclamation mark)
- poorly formed constant values (strings, numbers, etc)

Dealing with errors

- lexer returns a special error token ... enters "panic mode"
- lexer raises an exception ... what we'll do
- compilers stop (e.g., Python) or keep going (e.g., C++, Java)

Full Set of MyPL Tokens (for HW-1)

```
TokenType = Enum('TokenType', [
    # end-of-stream, identifiers, comments
    'EOS', 'ID', 'COMMENT',
    # punctuation
    'DOT', 'COMMA', 'LPAREN', 'RPAREN', 'LBRACKET', 'RBRACKET',
    'SEMICOLON', 'LBRACE', 'RBRACE',
    # operators
    'PLUS', 'MINUS', 'TIMES', 'DIVIDE', 'ASSIGN', 'AND', 'OR',
    'NOT',
    # relational comparators
    'EQUAL', 'NOT_EQUAL', 'LESS', 'LESS_EQ', 'GREATER',
    'GREATER_EQ',
    # values
    'INT_VAL', 'DOUBLE_VAL', 'STRING_VAL', 'BOOL_VAL',
    'NULL_VAL',
    # primitive data types
    'INT_TYPE', 'DOUBLE_TYPE', 'STRING_TYPE', 'BOOL_TYPE',
    'VOID_TYPE',
    # reserved words
    'STRUCT', 'ARRAY', 'FOR', 'WHILE', 'IF', 'ELSEIF', 'ELSE',
    'NEW', 'RETURN'
])


@dataclass
class Token:
    token_type: TokenType
    lexeme: str
    line: int
    column: int

    ...
```

The `Lexer` *class for HW-1*

Note: Must build tokens *one character-at-a-time* !

- using provided `read()` and `peek()` functions ...

```
class Lexer:

  def __init__(self, in_stream):
      self.in_stream = in_stream
      self.line = 1
      self.column = 0

  def read(self):
      self.column += 1
      return self.in_stream.read_char()

  def peek(self):
      return self.in_stream.peek_char()

  def eof(self, ch):
      return ch == ''

  def error(self, message, line, column):
      raise LexerError(f'{message} at line {line}, column {column}')

  def next_token(self):
      # read initial character
      ch = self.read()
```

Additionally:

- use `error()` function to report errors
- use `eof()` to check for end-of-file character

The MyPL Driver Code for Testing the Lexer

Repeatedly calls **`next_token()`** to print token stream

```
try:
    lexer = Lexer(in_stream)
    t = lexer.next_token()
    while t.token_type != TokenType.EOS:
        print(t)
        t = lexer.next_token()
    print(t)
except MyPLError as ex:
    print(ex)
```

To run using an input file: ... or: `python mypl.py --lex ...`

```
./mypl --lex examples/hello.mypl
2, 1: VOID_TYPE "void"
2, 6: ID "main"
2, 10: LPAREN "("
2, 11: RPAREN ")"
2, 13: LBRACE "{"
3, 3: ID "print"
3, 8: LPAREN "("
3, 9: STRING_VAL "Hello World!\n"
...
```

Can also run in standard input (Ctrl-d, or Windows Ctrl-z enter, for EOF)

```
./mypl --lex
void main() { print("Hello World\n"); }
1, 1: VOID_TYPE "void"
1, 6: ID "main"
1, 10: LPAREN "("
1, 11: RPAREN ")"
1, 13: LBRACE "{"
1, 15: ID "print"
1, 20: LPAREN "("
1, 21: STRING_VAL "Hello World\n"
...
```

Summary – Things to Know

1. Operation of the lexer (pull model, next token, etc.)