# CS421 Lecture 5 – Lexical Analysis

- Today's class
  - Lexing
  - Finite-State Machine as Lexer

# Compiler Outline

- Front-End
  - Takes Input Source Code
  - Returns Abstract Syntax Tree
- Back-End
  - Takes Abstract Syntax Tree
  - Returns Machine Executable Binary Code

# Compiler Outline (Figure)

# Manual and automatic methods

- We will study how to write lexers and parsers. For each, we will give a manual technique and an automatic one:
- Lexing:
  - Manual: Finite-state machines
  - Automatic: Regular expressions – ocamllex
- Parsing
  - Manual: Top-down (recursive descent) parsing
  - Automatic: Bottom-up (LR(1)) - ocamlyacc

# Lexer

- Divide input into “tokens”
- Tokens are smallest units that are useful for parsing. E.g. parser needs to know if “while” keyword appears; doesn’t need to know that it is made up of characters w, h, etc.
- Why? Efficiency
  - Simpler to specify grammatical structure, and implement parser, in terms of tokens

# Lexer Input & Output

- Lexer Input
  - Character stream in the form of
    - Input Stream, or
    - String
- Lexer Output
  - Stream of tokens, or
  - List of tokens

# Tokens

```
type token =
   EOF | BOOLEAN | BREAK | CASE | CHAR | CLASS | CONST | CONTINUE
 | DO | DOUBLE | ELSE | EXTENDS | FINAL | FINALLY | FLOAT | FOR
 | DEFAULT | IMPLEMENTS | IMPORT | INT | NEW | IF | PUBLIC
 | SWITCH | RETURN | VOID | STATIC | WHILE | THIS
 | NULL_LITERAL | LPAREN | RPAREN | LBRACE | RBRACE | LBRACK | RBRACK
 | SEMICOLON | COMMA | DOT | EQ | GT | LT | NOT | COMP
 | QUESTION | COLON | EQEQ | LTEQ | GTEQ | NOTEQ | ANDAND | OROR
 | PLUSPLUS | MINUSMINUS | PLUS | MINUS | MULT | DIV | AND
 | OR | XOR | MOD | LSHIFT | RSHIFT | URSHIFT | PLUSEQ | MINUSEQ |
   MULTEQ
 | DIVEQ | ANDEQ | OREQ | XOREQ | MODEQ | LSHIFTEQ | RSHIFTEQ
 | URSHIFTEQ
 | BOOLEAN_LITERAL of bool
 | INTEGER_LITERAL of int
 | FLOAT_LITERAL of float
 | IDENTIFIER of string
 | STRING_LITERAL of string
```

# Example

- Input

  “class MP1 { public static void main ( ……”

- Output

  [CLASS; IDENTIFIER “MP1”; LBRACE; PUBLIC; STATIC; VOID; IDENTIFIER “main”; LPAREN; …… ]

# Lexing with FSM

- Words recognized by corresponding finite state automaton
- Deterministic Finite Automaton (DFA)
  - A directed graph whose *vertices* are labeled from a set Tokens U {Error} and whose *edges* are labeled with sets of characters. Also, if the outgoing edges from vertex v are $\{e_1, \ldots, e_n\}$, then the sets label($e_1$), …, label($e_n$) are disjoint. Also, a vertex u specified as the start vertex.

# Example 1

- DFA for keywords

  class   case   finally

# Example 2

- DFA for Operators

  ;   {   +   +=   <   <=   <<   <<=

# Example 3

- DFA for integer constants

# Example 4

- DFA for integers and decimal

# Completing the DFA

- Need to create a single DFA for all tokens - recall that all outgoing edges must have disjoint label sets.
- DFA labels are similar to tokens in the token data type, but not necessarily identical
- For keyword & identifiers:
  - Instead of creating the DFA shown earlier, create a small DFA and use action to distinguish keywords

# Implementing lexing with a DFA

- Define a transition function
  - state x character -> state u {-1}
- Label
  - state -> token u {error}
- Assume start state = 0

# Implementing lexing with a DFA

Function to get a single token:

```
(state x string) getnexttoken() {
  s = 0;   tokenchars = “”;
  while (true) {
    c = peek at next char
    if (move(s,c) == -1)
      return (s,tokenchars)
    move c from input to tokenchars
    s = move(s,c)
  }
```

# Implementing lexing with a DFA

```
token list gettokens() {
    tokenlis = []
    while (true) {
        c = peek at next char
        if (c == eofchar) {
            tokenlis = tokenlis @ [EOF]
            break
        }
        (s, tokenchars) = getnexttoken()
        perform action based on s and tokenchars
    }
    return tokenlis
}
```

# Typical lexer actions

- Recall that label(s) is a token or error. Action depends on that label, e.g.:
  - Error:  Represents an erroneous input; abort.
  - LTLT:
  - IDENT:
  - COMMENT

# More DFAs

# More DFAs

# More DFAs