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 + symbol table

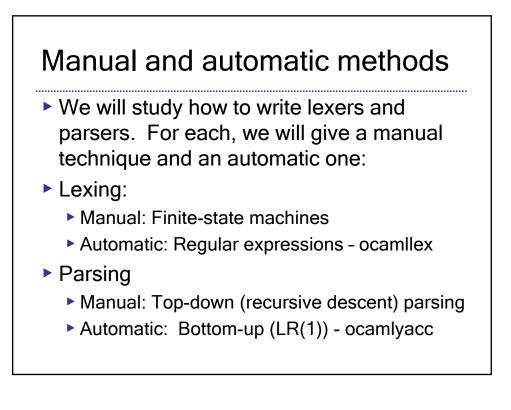
Back-End

Takes Abstract Syntax Tree + symbol table

 Returns machine executable binary code, or virtual machine code, or just interprets program

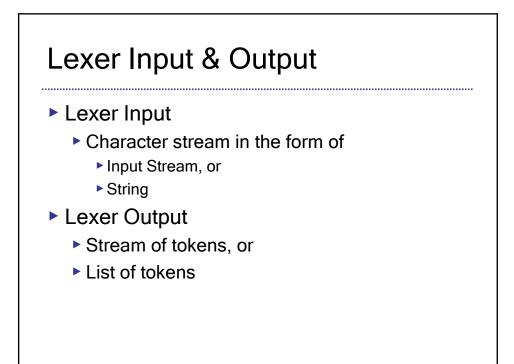
Front-end structure

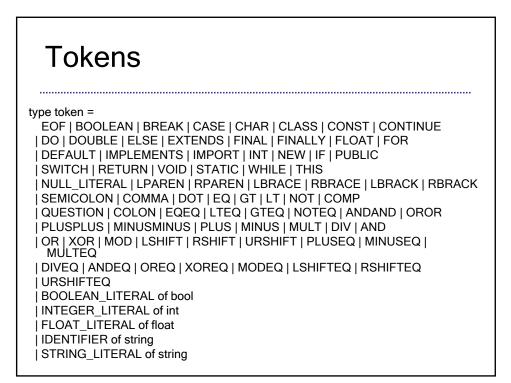
- Lexer (aka scanner, tokenizer)
 - Transforms program to list of tokens
 - Produces name table (usually hash table)
- Parser
 - Transforms list of tokens to AST
- Symbol table construction
 - Fills in name table with information about names in program - type, location, etc.

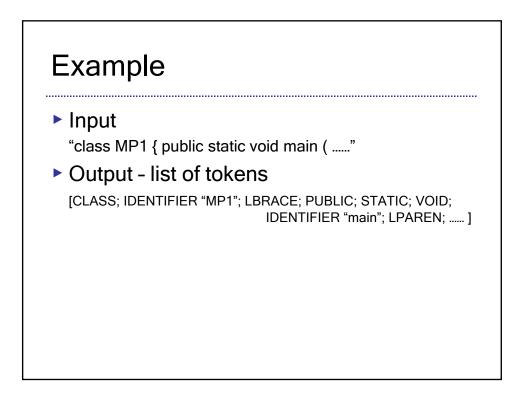


Lexer

- 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

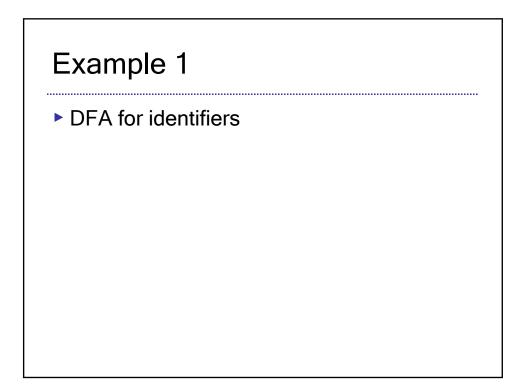








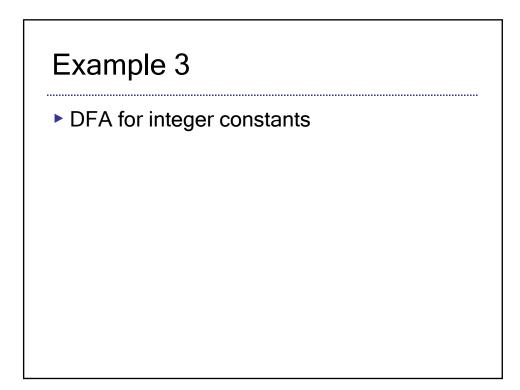
- Words recognized by corresponding finite state automaton
- Deterministic Finite Automaton (DFA)
 - A directed graph whose vertices are labeled from a set Tokens U {Error, Discard} and whose edges are labeled with sets of characters. Also, if the outgoing edges from vertex v are {e₁, ..., e_n}, then the sets label(e₁), ..., label(e_n) are disjoint. Also, a vertex is specified as the start vertex.



Example 2

- DFA for Operators
 - ; { + += < <= << <=

.....



Example 4

DFA for integers and floats

Completing the DFA

- Need to create a single DFA for all tokens recall that all outgoing edges must have disjoint label sets.
- For keyword:
 - Use DFA for identifiers, but look in table when token is complete to check if it is a keyword.

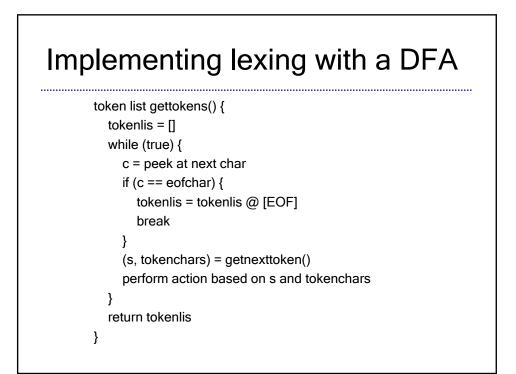
Completing the DFA

Implementing lexing with a DFA

- Define a transition function. Give each state a number.
 - ▶ transition: state x character -> state ∪ {-1}
- Label
 - ► state -> token ∪ {discard, error}
- Assume start state = 0

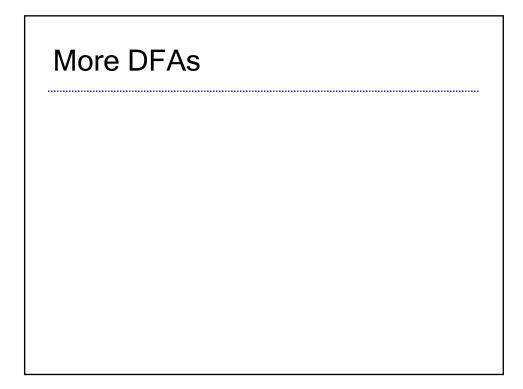
Implementing lexing with a DFA

```
Function to get a single token:
(state × 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)
    }
```



Typical lexer actions

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



More DFAs

.....