Lexing

CS 132: Compiler Design

January 24, 2011
DEFINITIONS

✓ What is lexing (a.k.a. tokenizing)?
Definitions

✓ What is lexing (a.k.a. tokenizing)?

✓ What is parsing?
DEFINITIONS

✓ What is lexing (a.k.a. tokenizing)?

✓ What is parsing?

✓ What is abstract syntax?
Concrete Syntaxes

Pascal vs. C vs. Python

if space_left <> 0 then
begin
    processLine(line);
    getNextLine
end
else
begin
    getNextLine;
    skipped := 1
end

if (space_left != 0) {
    processLine(line);
    getNextLine();
} else {
    getNextLine();
    skipped = 1;
}

if (space_left != 0):
    processLine(line)
    getNextLine()
else:
    getNextLine()
    skipped = 1
**TRADITIONAL PARSING**

\[( x - 32 ) \geq 7 \times y\]

**LEXING**

LPAREN ID DASH INT RPAREN GEQ NUM STAR ID

"x" 32 7 "y"

**PARSING**

Geq

BinOp

BinOp BinOp

Var Num Var Num

"x" 32 7 "y"
Question: Why bother with this two-step process?
What are the usual "primitive" regular expressions?
Regular Expression Review

✓ What are the usual "primitive" regular expressions?

✓ What other regular expressions are you familiar with?
From the RX library

M[ou]'am+[ae]r .*([AEae]l[- ])?[GKQ] h?[aeu]+([dtz][dhz]?)+af[iy]
<table>
<thead>
<tr>
<th>Name</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Muammar Qaddafi</td>
<td>Moamar Gaddafi</td>
</tr>
<tr>
<td>Mo'amar Gadhafi</td>
<td>Mu'amar Qadhdhafi</td>
</tr>
<tr>
<td>Muammar Kaddafi</td>
<td>Muammar al-Khaddafi</td>
</tr>
<tr>
<td>Muammar Qadhafi</td>
<td>Mu'amar al-Kadafi</td>
</tr>
<tr>
<td>Moammar El Kadhafi</td>
<td>Muammar Ghaddafy</td>
</tr>
<tr>
<td>Muammar Gadafi</td>
<td>Muammar Ghadafi</td>
</tr>
<tr>
<td>Mu'ammar al-Qadafi</td>
<td>Muammar Ghaddafi</td>
</tr>
<tr>
<td>Moamer El Kazzafi</td>
<td>Muamar Kaddafi</td>
</tr>
<tr>
<td>Moamar al-Gaddafi</td>
<td>Muammar Quathafi</td>
</tr>
<tr>
<td>Mu'ammar Al Qathafi</td>
<td>Muammar Gheddafi</td>
</tr>
<tr>
<td>Muammar Al Qathafi</td>
<td>Muamar Al-Kaddafi</td>
</tr>
<tr>
<td>Mo'ammar el-Gadhafi</td>
<td>Moamar Khadafy</td>
</tr>
<tr>
<td>Moamar El Kadhafi</td>
<td>Moammar Qudhafi</td>
</tr>
<tr>
<td>Muammar al-Qadhafi</td>
<td>Mu'ammar al-Qaddafi</td>
</tr>
<tr>
<td>Mu'ammar al-Qadhdhafi</td>
<td>Mu'ammar Muhammad Abu Minyar al-Qadhafi</td>
</tr>
<tr>
<td>Mu'ammar Qadafi</td>
<td></td>
</tr>
</tbody>
</table>

*From the RX library*
**Applying Regular Expressions**

Give a regular expression for Ada identifiers, which

- Can contain letters, digits and underscores
- Begin with a letter
- Have no consecutive underscores or an underscore at the end

Give a regular expression for dollar amounts on US checks. (Note: many printed checks contain extra asterisks to prevent fraud, i.e., $****1.00.)
Finite Automata

Draw NFAs and DFAs for the following regular expressions:

✓ If keyword: if

✓ Case-insensitive if keyword: [Ii][Ff]

✓ Identifiers: [a-zA-Z_][a-zA-Z_0-9]*
PRACTICAL ISSUES

✓ Keywords: if int end
✓ Identifiers: [a-zA-Z_][a-zA-Z_0-9]*

✓ Should intend be one token or two?
PRACTICAL ISSUES

✓ Keywords: if   int   end
✓ Identifiers: [a-zA-Z_][a-zA-Z_0-9]*

✓ Should intend be one token or two?
✓ Is if a keyword or an identifier?
### Practical Issues

- **Keywords**: `if` `int` `end`
- **Identifiers**: `[a-zA-z_][a-zA-Z_0-9]`*

- Should **intend** be one token or two?
- Is **if** a keyword or an identifier?
- How many tokens are there in `3-2`?
**PRACTICAL ISSUES**

✓ Keywords: `if int end`

✓ Identifiers: `\[a-zA-Z_][a-zA-Z_0-9]*`

✓ Should `intend` be one token or two?

✓ Is `if` a keyword or an identifier?

✓ How many tokens are there in `3-2`?

✓ How many tokens are there in `3 + -2`?
PRACTICAL ISSUES

✓ Keywords: if int end
✓ Identifiers: [a-zA-Z_][a-zA-Z_0-9]*

✓ Should intend be one token or two?
✓ Is if a keyword or an identifier?
✓ How many tokens are there in 3-2?
✓ How many tokens are there in 3 + -2?
✓ What about 3-2147483648?
NOTE ON RESERVED WORDS

Many languages have reserved words (keywords) that cannot be used as identifiers (if, return, etc.)
Note on Reserved Words

Many languages have reserved words (keywords) that cannot be used as identifiers (if, return, etc.)

On the other hand, there are languages like PL/I:

IF THEN THEN THEN = ELSE; ELSE ELSE ELSE = THEN;
BUILDING A LEXER

Pascal included tokens of the following form:

✓ Integers: \([0-9]+\)
✓ Real numbers: \([0-9]+\).\([0-9]+\)
✓ Range marker: \(".."\)

How do we combine them into a lexer?
BUILDING A LEXER

Pascal included tokens of the following form:

✓ Integers: [0-9]+
✓ Real numbers: [0-9]+"."[0-9]+
✓ Range marker: ".."

How do we combine them into a lexer?

What is the worst-case running time for a DFA-based lexer?
Lexing Challenges: Fortran

✓ Pre-1977 Hollerith Constants

15HCOMPILER DESIGN
LEXING CHALLENGES: FORTRAN

✓ Pre-1977 Hollerith Constants
   15HCOMPILER DESIGN

✓ Fixed-Column Layouts
   CALL DOIT(X,Y,Z)
      CALL DOIT(X,Y,Z)
Lexing Challenges: Fortran

 ✓ Pre-1977 Hollerith Constants
   15HCOMPILER DESIGN

 ✓ Fixed-Column Layouts
   CALL DOIT(X,Y,Z)
   CALL DOIT(X,Y,Z)

 ✓ Whitespace is not significant.
   DO 10 I = 1,15
   DO 10 I = 1.15
   REAL X
   REAL X = 3.5
   INTEGER FUNCTION A(I)
PARSENG CHALLENGES: C, C++, C#, JAVA, ...

✓ Often can be resolved by extra information from the lexer.

```c
X * Y;
X (Y);
X < Y > Z;
X ? Y : Z;
X ? Y = Z;
(T)-(U);
```
**Parsing Challenges: C, C++, C#, Java, …**

✓ Often can be resolved by extra information from the lexer.

\[
X \times Y; \\
X (Y); \\
X < Y > Z; \\
X ? Y : Z; \\
X ? Y = Z; \\
(T) - (U);
\]

✓ Not all challenges are this easy, though

\[
\text{int}(x), y, *\text{const } z; \quad \text{// variable declarations} \\
\text{int}(x), y, \text{new int;} \quad \text{// list of expressions} \\
\text{int}(x), y, z = 0; \quad \text{// variable declarations}
\]
Once we have a DFA, how do we turn it into code?