CS480
Translators
More Lexical Analysis
Chap. 2 & 3

Crazy Semantics
https://www.destroyallsoftware.com/talks/wat
...Thank you Kevin Strasser

Mini-Translator

while (i > 0)
    i = i - 2;

T_WHILE
T_LPAREN
T_IDENTIFIER
T_LESSTHAN
T_INTCOMPONENT
T_RPAREN
T_IDENTIFIER
T_EQUALS
T_MINUS
T_INTCOMPONENT
T_SEMICOLON

Source Language
Lexical Analyzer
Token Stream
Error Messages
Token Data Structures

class Token
int tag

class Num
int value

class Word
string lexeme

struct token_t {
    int tag;
    OR
    struct token_t {
        union {
            char *lexeme;
            int value;
        }
        void *val;
    }
};

Keywords vs. Identifiers

• count = count + increment;

    <id, "count"> <=> <id, "count"> <=> <id, "increment"> =>

• How do we know count is an id vs. keyword?
• Why use a hash table?
• What is in the hash table?

How to distinguish words?

if ( peek holds a letter ) {
    collect letters or digits into a buffer k;
    s = string formed from the characters in k;
    w = token returned by words.get(s);
    if ( w is not null ) return w;
    else {
        Enter the key-value pair (s, (id, s)) into words
        return token (id, s);
    }
}

Figure 2.31: Distinguishing keywords from identifiers
Example Lexical Analyzer

1) package lexer; // File Lexer.java
2) import java.io.*; import java.util.*;
3) public class Lexer {
4)   public int line = 1;
5)   private char peak = ' ';
6)   private Hashtable words = new Hashtable();
7)   void reserve(Word k) { words.put(k.lexeme, k); }
8)   public Lexer() {
9)     reserve(new Word(Tag.TRUE, "true");
10)    reserve(new Word(Tag.FALSE, "false");
11)   }
12)   public Token scan() throws IOException {
13)     for(; ; peak = (char)System.in.read()) {
14)       if( peak == ' ' || peak == '
') continue;
15)       else if( peak == '
' ) line = line + 1;
16)       else break;
17)   }

Patterns/Regular Expressions

- Defining tokens
  - if while assign
  - () = >
  - [a-zA-Z][a-zA-Z0-9]*
  - [0-9]+
  - \s+\n
Languages/Regular Expressions

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>DEFINITION AND NOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Union of ( L ) and ( M )</td>
<td>( L \cup M = { x \mid x \in L \text{ or } x \in M } )</td>
</tr>
<tr>
<td>Concatenation of ( L ) and ( M )</td>
<td>( LM = { x \circ y \mid x \in L \text{ and } y \in M } )</td>
</tr>
<tr>
<td>Kleene closure of ( L )</td>
<td>( L^* = \cup_{n=0}^\infty L^n )</td>
</tr>
<tr>
<td>Positive closure of ( L )</td>
<td>( L^+ = \cup_{n=1}^\infty L^n )</td>
</tr>
</tbody>
</table>

- Every symbol of \( \Sigma \) is a regular expression
- \( \epsilon \) is a regular expression
- if \( r_1 \) and \( r_2 \) are regular expressions, so are
  \( (r_1) \ r_1r_2 \ r_1 \ r_1^* \)
- Nothing else is a regular expression.

Example Regular Expressions

- Given \( \Sigma = \{a,b\} \), provide regular expressions for languages below:
  - all strings beginning and ending in \( a \)
  - all strings of \( a \)'s and \( b \)'s of even length
  - all strings with an odd number of \( a \)'s
  - string of zero or more \( a \)'s followed by same number of \( b \)'s