Today’s topics

Java
  Syntax of Computer Language
Upcoming
  More Java

Reading
  *Great Ideas*, Chapter 2

Grammar

- English and other natural languages have structure
  
  \[
  \langle S \rangle \Rightarrow \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB-PHRASE} \rangle
  \]
  
  \[
  \langle \text{NOUN-PHRASE} \rangle \Rightarrow \langle \text{NOUN} \rangle ~ | ~ \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle
  \]
  
  \[
  \langle \text{VERB-PHRASE} \rangle \Rightarrow \langle \text{VERB} \rangle ~ | ~ \langle \text{VERB} \rangle \langle \text{NOUN-PHRASE} \rangle
  \]
  
  \[
  \langle \text{NOUN} \rangle \Rightarrow \text{DOG} ~ | ~ \text{FLEAS} ~ | ~ \text{PERSON} ~ | ~ \ldots
  \]
  
  \[
  \langle \text{VERB} \rangle \Rightarrow \text{RAN} ~ | ~ \text{BIT} ~ | ~ \ldots
  \]

- Process of taking sentence and fitting it to grammar is called *parsing*

  \[
  \text{DOG BIT PERSON}
  \]

  \[
  \text{<NOUN> <VERB> <NOUN>}
  \]

  \[
  \text{<NOUN-PHRASE> <VERB-PHRASE>}
  \]

- Parsing English is complex because of *context dependence*

Formal specifications

- Need a precise notation of syntax of a language
- Grammars can also be used for generation
- Grammar rule:
  \[
  \langle \text{name} \rangle \Rightarrow \text{sequence of letters and/or digits that begins with a letter}
  \]
- Grammar use:
  \[
  \langle \text{name} \rangle \Rightarrow \text{guessB}
  \]
  \[
  \langle \text{name} \rangle \Rightarrow \text{msg42}
  \]
- Substitute as many times as necessary. All legal statements can be generated this way
- Want: \[
  \text{person} = \text{firstn} + " " + \text{lastn};
  \]
  
  How do we get this from our grammar?

A Grammar for Java

- Need a set of rules
- Our first one was a good start:
  
  \[
  \langle \text{name} \rangle \Rightarrow \text{any string of alphanumeric symbols that begins with a letter}
  \]

- Let’s add something to define a simple statement:
  
  \[
  \langle \text{statement} \rangle \Rightarrow \langle \text{name} \rangle = \langle \text{expression} \rangle ;
  \]

- And then work on the details:
  
  \[
  \langle \text{expression} \rangle \Rightarrow \langle \text{string-expression} \rangle | \langle \text{int-expression} \rangle | \langle \text{oth-expression} \rangle
  \]
  
  \[
  \langle \text{string-expression} \rangle \Rightarrow \langle \text{string} \rangle
  \]
  
  \[
  \langle \text{string} \rangle \Rightarrow " \text{any sequence of characters}"\]
A Simple Statement

- Now have enough to generate a statement like: \texttt{msg = "hello";}.
  - Start with:
    - \texttt{<statement> => <name> = <expression> ;}
  - Then using: \texttt{<name> => any string of alphanumeric symbols that begins with a letter}
    \texttt{msg = <expression> ;}
  - Then, using: \texttt{<expression> => <string-expression> | <int-expression> | <oth-expression>}
    \texttt{msg = <string-expression> ;}
  - Using: \texttt{<string-expression> => <string>}
    \texttt{msg = <string> ;}
  - Using: \texttt{<string> => "any sequence of characters"}
    \texttt{msg = "hello" ;}

A Grammar for Java

- Including more rules to describe programs we have:
  1. \texttt{<name> => any string of alphanumeric symbols that begins with a letter}
  2. \texttt{<statement> => <name> = <expression> ;}
  3. \texttt{<statement> => <name> = new <class> (<arguments>) ;}
  4. \texttt{<statement> => <name> , <method> (<arguments>) ; |}
    \texttt{<method> (<arguments>) ;}
  5. \texttt{<arguments> => possibly empty list of <expression>s separated by commas}
  6. \texttt{<expression> => <string-expression> | <int-expression> |}
    \texttt{<oth-expression>}
  7. \texttt{<string-expression> => <string-expression> + <string-expression>}
  8. \texttt{<string-expression> => <string> + <string-expression>}
  9. \texttt{<string> => "any sequence of characters"}
 10. \texttt{<string> => <name>}

Using our Grammar

- Use this to generate: \texttt{person = firstn + " " + lastn;}

<table>
<thead>
<tr>
<th>Rule</th>
<th>Statement being Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>\texttt{&lt;statement&gt; =&gt; &lt;name&gt; = &lt;expression&gt; ;}</td>
</tr>
<tr>
<td>2</td>
<td>\texttt{&lt;statement&gt; =&gt; person = &lt;expression&gt; ;}</td>
</tr>
<tr>
<td>3</td>
<td>\texttt{&lt;statement&gt; =&gt; person = &lt;string&gt; + &lt;string-expression&gt; ;}</td>
</tr>
<tr>
<td>4</td>
<td>\texttt{&lt;statement&gt; =&gt; person = firstn + &lt;string&gt; + &lt;string-expression&gt; ;}</td>
</tr>
<tr>
<td>5</td>
<td>\texttt{&lt;statement&gt; =&gt; person = firstn + &quot; &quot; + &lt;string&gt; ;}</td>
</tr>
<tr>
<td>6</td>
<td>\texttt{&lt;statement&gt; =&gt; person = firstn + &quot; &quot; + &lt;name&gt; ;}</td>
</tr>
</tbody>
</table>

Proving Grammatical Correctness

- Why go through the process we went through?
  - Shows that desired statement can be generated from this grammar
  - Actually Proves that the statement is grammatically correct!
  - Same rigor as a mathematical proof
  - (Doesn’t prove that logic is correct, though)

- Actually need more rules to handle the level of Java we’ve covered so far
  - Summary of rules shown on pages 78-79 of Great Ideas
  - Also give an example for a complete applet
  - Too long to go through in class – Please Read!