Today’s topics

Java
Syntax of Computer Language

Upcoming
More Java

Reading
Great Ideas, Chapter 2
Grammar

- English and other natural languages have structure

\[
\begin{align*}
\langle S \rangle & \rightarrow \langle \text{Noun-Phrase} \rangle \langle \text{Verb-Phrase} \rangle \\
\langle \text{Noun-Phrase} \rangle & \rightarrow \langle \text{Noun} \rangle \mid \langle \text{Article} \rangle \langle \text{Noun} \rangle \\
\langle \text{Verb-Phrase} \rangle & \rightarrow \langle \text{Verb} \rangle \mid \langle \text{Verb} \rangle \langle \text{Noun-Phrase} \rangle \\
\langle \text{Noun} \rangle & \rightarrow \text{DOG} \mid \text{FLEAS} \mid \text{PERSON} \mid \ldots \\
\langle \text{Verb} \rangle & \rightarrow \text{RAN} \mid \text{BIT} \mid \ldots
\end{align*}
\]

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

DOG BIT PERSON

\[
\langle \text{Noun} \rangle \langle \text{Verb} \rangle \langle \text{Noun} \rangle
\]

\[
\langle \text{Noun-Phrase} \rangle \langle \text{Verb-Phrase} \rangle
\]

\[
\langle S \rangle
\]

- 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:
  \(<\text{name}>\) \(\Rightarrow\) sequence of letters and/or digits that begins with a letter
- Grammar use:
  \(<\text{name}>\) \(\Rightarrow\) guessB
  \(<\text{name}>\) \(\Rightarrow\) 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:
  - `<name> => any string of alphanumeric symbols that begins with a letter`
- Let’s add something to define a simple statement:
  - `<statement> => <name> = <expression> ;`
- And then work on the details:
  - `<expression> => <string-expression> | <int-expression> | <oth-expression>`
  - `<string-expression> => <string>`
  - `<string> => <name>`
  - `<string> => "any sequence of characters"`
A Simple Statement

- Now have enough to generate a statement like:
  
  ```
  msg = "hello";
  ```

- Start with:
  
  ```
  <statement> => <name> = <expression> ;
  ```

- Then using: `<name> => any string of alphanumeric symbols that begins with a letter`

  ```
  msg = <expression> ;
  ```

- Then, using: `<expression> => <string-expression> | <int-expression> | <oth-expression>`

  ```
  msg = <string-expression> ;
  ```

- Using: `<string-expression> => <string>`

  ```
  msg = <string> ;
  ```

- Using: `<string> => "any sequence of characters"`

  ```
  msg = "hello" ;
  ```
A Grammar for Java

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

Use this to generate:

```plaintext
person = firstn + " " + lastn;
```

<table>
<thead>
<tr>
<th>Rule</th>
<th>Statement being Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:</td>
<td><code>&lt;statement&gt; =&gt; &lt;name&gt; = &lt;expression&gt; ;</code></td>
</tr>
<tr>
<td>1:</td>
<td><code>&lt;statement&gt; =&gt; person = &lt;expression&gt; ;</code></td>
</tr>
<tr>
<td>6:</td>
<td><code>&lt;statement&gt; =&gt; person = &lt;str-expression&gt; ;</code></td>
</tr>
<tr>
<td>7:</td>
<td><code>&lt;statement&gt; =&gt; person = &lt;str-expression&gt; + &lt;str-expression&gt; ;</code></td>
</tr>
<tr>
<td>8:</td>
<td><code>&lt;statement&gt; =&gt; person = &lt;string&gt; + &lt;str-expression&gt; ;</code></td>
</tr>
<tr>
<td>10:</td>
<td><code>&lt;statement&gt; =&gt; person = &lt;name&gt; + &lt;str-expression&gt; ;</code></td>
</tr>
<tr>
<td>1:</td>
<td><code>&lt;statement&gt; =&gt; person = firstn + &lt;str-expression&gt; ;</code></td>
</tr>
<tr>
<td>7:</td>
<td><code>&lt;statement&gt; =&gt; person = firstn + &lt;str-expression&gt; + &lt;str-expression&gt; ;</code></td>
</tr>
<tr>
<td>8:</td>
<td><code>&lt;statement&gt; =&gt; person = firstn + &lt;string&gt; + &lt;str-expression&gt; ;</code></td>
</tr>
<tr>
<td>9:</td>
<td><code>&lt;statement&gt; =&gt; person = firstn + &quot; &quot; + &lt;str expression&gt; ;</code></td>
</tr>
<tr>
<td>8:</td>
<td><code>&lt;statement&gt; =&gt; person = firstn + &quot; &quot; + &lt;string&gt; ;</code></td>
</tr>
<tr>
<td>10:</td>
<td><code>&lt;statement&gt; =&gt; person = firstn + &quot; &quot; + &lt;name&gt; ;</code></td>
</tr>
<tr>
<td>1:</td>
<td><code>&lt;statement&gt; =&gt; person = firstn + &quot; &quot; + lastn;</code></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
- (Does not 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 78-80 of Great Ideas
  - Also gives an example for a complete applet
  - Too long to go through in class – Please Read!