Today’s topics

Java
  Syntax of Computer Language

Upcoming
  More Java

Reading
  *Great Ideas*, Chapter 2
Grammar

- English and other natural languages have structure
  \[
  S \rightarrow N \cdot P \cdot N \cdot P \cdot V \cdot P
  \]
  \[
  N \cdot P \rightarrow N \mid A \cdot N
  \]
  \[
  V \cdot P \rightarrow V \mid V \cdot N
  \]
  \[
  N \rightarrow D O G \mid F L E A S \mid P E R S O N \mid \ldots
  \]
  \[
  V \rightarrow R A N \mid B I T \mid \ldots
  \]
- Process of taking sentence and fitting it to grammar is called **parsing**

DOG BIT PERSON

- 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:
  \[ \text{<name>} \Rightarrow \text{sequence of letters and/or digits that begins with a letter} \]
- Grammar use:
  \[ \text{<name>} \Rightarrow \text{guessB} \]
  \[ \text{<name>} \Rightarrow \text{msg42} \]
- 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:
  - `<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:*

```markdown
msg = "hello";
```

- Start with:
  ```
  <statement> => <name> = <expression> ;
  ```
- Then using: `<name> => any string of alphanumerical symbols that begins with a letter`

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

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

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

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

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

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

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

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

Use this to generate:

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

<table>
<thead>
<tr>
<th>Rule</th>
<th>Statement being Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td></td>
</tr>
<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 give an example for a complete applet
  - Too long to go through in class – Please Read!