Today’s topics

Java
- Syntax of Computer Language

Upcoming
- More Java

Reading
- *Great Ideas*, Chapter 2
Grammar

- English and other natural languages have structure

\[
\begin{align*}
  \langle S \rangle & \Rightarrow \langle \text{Noun-Phrase} \rangle \hspace{1em} \langle \text{Verb-Phrase} \rangle \\
  \langle \text{Noun-Phrase} \rangle & \Rightarrow \langle \text{Noun} \rangle \mid \langle \text{Article} \rangle \hspace{0.5em} \langle \text{Noun} \rangle \\
  \langle \text{Verb-Phrase} \rangle & \Rightarrow \langle \text{Verb} \rangle \mid \langle \text{Verb} \rangle \hspace{0.5em} \langle \text{Noun-Phrase} \rangle \\
  \langle \text{Noun} \rangle & \Rightarrow \text{DOG} \mid \text{FLEAS} \mid \text{PERSON} \mid \ldots \\
  \langle \text{Verb} \rangle & \Rightarrow \text{RAN} \mid \text{BIT} \mid \ldots
\end{align*}
\]

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

\[
\text{DOG BIT PERSON}
\]

\[
\begin{align*}
  \langle \text{Noun} \rangle & \hspace{1em} \langle \text{Verb} \rangle & \hspace{1em} \langle \text{Noun} \rangle \\
  \langle \text{Noun-Phrase} \rangle & \hspace{1em} \langle \text{Verb-Phrase} \rangle \\
  \langle S \rangle
\end{align*}
\]

- 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:
  \(<name> \Rightarrow \text{sequence of letters and/or digits that begins with a letter}\)
- Grammar use:
  \(<name> \Rightarrow \text{guessB}\>
  \(<name> \Rightarrow \text{msg42}\>
  \(<name> \Rightarrow \text{msg42}\>
- Substitute as many times as necessary. All legal statements can be generated this way
- Want: \text{person = firstn + " " + 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 charcters"`
    `msg = "hello" ;`
A Grammar for Java

- Including more rules to describe programs we have:
  1. `<name>` => *any string of alphanumerical 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: `person = firstn + " " + lastn;`

Rule Statement being Generated

```
# 2: <statement> => <name> = <expression>;  
1: <statement> => person = <expression>;  
6: <statement> => person = <str-expression>;  
7: <statement> => person = <str-expression> + <str-expression>;  
8: <statement> => person = <string> + <str-expression>;  
10: <statement> => person = <name> + <str-expression>;  
  1: <statement> => person = firstn + <str-expression>;  
  7: <statement> => person = firstn + <str-expression> + <str-expression>;  
  8: <statement> => person = firstn + <string> + <str-expression>;  
  9: <statement> => person = firstn + " " + <str-expression>;  
  8: <statement> => person = firstn + " " + <string>;  
10: <statement> => person = firstn + " " + <name>;  
  1: <statement> => <statement> => person = firstn + " " + lastn; 
```
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!