**Pointers to material**

- Parse trees: text, section 23.36
- Definition of Java Language, sometimes useful:
  docs.oracle.com/javase/specs/jls/se7/html/index.html
- Grammar for most of Java, for those who are curious:
  docs.oracle.com/javase/specs/jls/se7/html/jls-18.html
- Tree traversals – preorder, inorder, postorder: text, sections 23.13 .. 23.15.

---

**Expression trees**

Can draw a tree for \((2 + 3) \ast (1 + (5 - 4))\)

```
public abstract class Exp {
    /* return the value of this Exp */
    public abstract int eval();
}
```

```
public class Int extends Exp {
    int v;
    public int eval() {
        return v;
    }
}
```

```
public class Add extends Exp {
    Exp left;
    Exp right;
    public int eval() {
        return left.eval() + right.eval();
    }
}
```

---

**Expression trees**

Can draw a tree for \((2 + 3) \ast (1 + -4)\)

```
public abstract class Exp {
    /* return the value of this Exp */
    public abstract int eval();
}
```

```
public class Int extends Exp {
    int v;
    public int eval() {
        return v;
    }
}
```

```
public class Add extends Exp {
    Exp left;
    Exp right;
    public int eval() {
        return left.eval() + right.eval();
    }
}
```

---

**Expression trees**

Can draw a tree for \((2 + 3) \ast (1 + 4 - 4)\)

```
public abstract class Exp {
    /* return the value of this Exp */
    public abstract int eval();
}
```

```
public class Int extends Exp {
    int v;
    public int eval() {
        return v;
    }
}
```

```
public class Add extends Exp {
    Exp left;
    Exp right;
    public int eval() {
        return left.eval() + right.eval();
    }
}
```

---

**Expression trees**

In about 1974, Gries paid $300 for an HP calculator, which had some memory and used postfix notation! Still works. Come up to see it.

```
2 3 + 1 4 - + *
```

Postorder traversal:
1. Visit left subtree, in postorder
2. Visit right subtree, in postorder
3. Visit the root

Prefix and postfix notation proposed by Jan Lukasiewicz in 1951

Postfix (we see it later) is often called RPN for Reverse Polish Notation
Postfix notation

Then: (HP 45) RPN. 9 memory locations, 4-level stack, 1-line display
Now: (HP 35S) RPN and infix. 30K user memory, 2-line display

```
expression trees
```

tree for \((2 + 3) \times (1 + - 4)\)

```
postfix is easy to compute.
process elements left to right.
number? push it on a stack
binary operator? remove two top stack elements,
apply operator to it, push result on stack
unary operator? remove top stack element, apply
operator to it, push result on stack
```

Motivation for grammars

- The cat ate the rat.
- The small cat ate the big rat slowly.
- The small cat ate the big rat on the mat slowly.
- The small cat that sat in the hot ate the big rat on the mat slowly, then got sick.
- ... 

```
A Grammar
```

```
expression trees
```

```
Sentence  →  Noun  Verb  Noun
Noun  →  boys
Noun  →  girls
Noun  →  bunnies
Verb  →  like
   | see
```

```
http://docs.oracle.com/javase/specs/jls/se7/html/index.html
```

Inorder traversal:
1. Visit left subtree, in inorder
2. Visit the root
3. Visit right subtree, in inorder

To help out, put parens around expressions with operators

```
(2 + 3) \times (1 + - 4)
```

```
Inorder:
1. Visit left subtree, in inorder
2. Visit the root
3. Visit right subtree, in inorder
```

```
Expression trees
```

```
java public class Add extends Exp {
Exp left;
Exp right;
/** Return the value of this exp. */
public int eval() {return left.eval() + right.eval();}
/** Return the preorder. */
public String pre() {return '+' + left.pre() + right.pre();}
/** Return the postorder. */
public String post() {return left.post() + right.post() + '+' + '-' + '2';}
}
```

```
A Grammar
```

```
Sentence  →  Noun  Verb  Noun
Noun  →  boys
Noun  →  girls
Noun  →  bunnies
Verb  →  like
   | see
```

Our sample grammar has these rules:
A Sentence can be a Noun followed by a Verb followed by a Noun
A Noun can be boys or girls or bunnies
A Verb can be like or see

```
Motivation for grammars
```

- Not all sequences of words are legal sentences
- The ate cat rat the
- How many legal sentences are there?
- How many legal Java programs
- How do we know what programs are legal?

http://docs.oracle.com/javase/specs/jls/se7/html/index.html
A Grammar

<table>
<thead>
<tr>
<th>Sentence</th>
<th>→</th>
<th>Noun</th>
<th>Verb</th>
<th>Noun</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noun</td>
<td>→</td>
<td>boys</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Noun</td>
<td>→</td>
<td>girls</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Noun</td>
<td>→</td>
<td>bunnies</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verb</td>
<td>→</td>
<td>like</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verb</td>
<td>→</td>
<td>see</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Examples of Sentence:

- boys see bunnies
- bunnies like girls

• The words boys, girls, bunnies, like, see are called tokens or terminals
• The words Sentence, Noun, Verb are called nonterminals


Detour

What if we want to add a period at the end of every sentence?

Sentence → Sentence and Sentence .
Sentence → Sentence or Sentence .
Sentence → Noun Verb Noun .
Noun → ...

Does this work?

No! This produces sentences like:
girls like boys . and boys like bunnies . .

Sentence Sentence Sentence

Sentences with periods

PunctuatedSentence → Sentence .
Sentence → Sentence and Sentence .
Sentence → Sentence or Sentence .
Sentence → Noun Verb Noun .
Noun → boys
Noun → girls
Noun → bunnies
Verb → like
Verb → see

- New rule adds a period only at end of sentence.
- Tokens are the 7 words plus the period (.)
- Grammar is ambiguous:
  - boys like girls
  - and girls like boys
  - or girls like bunnies

Grammars for programming languages

Grammar describes every possible legal expression

You could use the grammar for Java to list every possible Java program. (It would take forever.)

Grammar tells the Java compiler how to "parse" a Java program

docs.oracle.com/javase/specs/jls/se7/html/jls-2.html#jls-2.3

Grammar for simple expressions (not the best)

E → integer
E → ( E + E )

Simple expressions:
- An E can be an integer.
- An E can be '(' followed by an E followed by '+' followed by an E followed by ')'

Some legal expressions:
- 2
- (3 + 34)
- ((4+23) + 89)

Some illegal expressions:
- 3
- 3 + 4

Set of expressions defined by this grammar is a recursively-defined set
- Is language finite or infinite?
- Do recursive grammars always yield infinite languages?

Tokens of this grammar:
- ( + ) and any integer
Use a grammar in two ways:

- A grammar defines a language (i.e. the set of properly structured sentences)
- A grammar can be used to parse a sentence (thus, checking if a string is a sentence is in the language)

To parse a sentence is to build a parse tree: much like diagramming a sentence.

Example: Show that $(4+23)+89$ is a valid expression $E$ by building a parse tree.

Grammar is ambiguous if it allows two parse trees for a sentence. The grammar below, using no parentheses, is ambiguous. The two parse trees to right show this. We don’t know which $+$ to evaluate first in the expression $1+2+3$.

Write a set of mutually recursive methods to check if a sentence is in the language (show how to generate parse tree later). One method for each nonterminal of the grammar. The method is completely determined by the rules for that nonterminal. On the next pages, we give a high-level version of the method for nonterminal $E$:

$$E \rightarrow \text{integer}$$
$$E \rightarrow (E+E)$$

/** Unprocessed input starts an $E$. Recognize that $E$, throwing away each piece from the input as it is recognized. Return false if error is detected and true if no errors. Upon return, processed tokens have been removed from input. */

```java
public boolean parseE() {
    if (first token is an integer) remove it from input and return true;
    if (first token is not '(') return false
    else remove it from input;
    if (!parseE()) return false;
    if (first token is not '+') return false
    else remove it from input;
    if (!parseE()) return false;
    if (first token is not ')') return false
    else remove it from input;
    return true;
}
```

Same code used 3 times. Cries out for a method to do that.
The scanner constructs tokens

An object scanner of class Scanner is in charge of the input String. It constructs the tokens from the String as necessary, e.g. from the string "1464+634" build the token "1464", the token "+", and the token "634". It is ready to work with the part of the input string that has not yet been processed and has thrown away the part that is already processed, in left-to-right fashion.

Change parser to generate a tree

```java
/** … Return a Tree for the E if no error. Return null if there was an error*/
public Tree parseE() {
    if (first token is an integer) remove it from input and return true;
    …
}
```

```java
E → integer
E → (E + E)
```

Code for a stack machine

```java
Code for 2 + (3 + 4)
PUSH 2
PUSH 3
PUSH 4
ADD
ADD
ADD: remove two top values from stack, add them, and place result on stack
Stack

It's postfix notation! 2 3 4 + +
```

Use parser to generate code for a stack machine

```java
Code for 2 + (3 + 4)
parseE can generate code as follows:

- For integer i, return string “PUSH " + i + "n"
- For (E1 + E2), return a string containing Code for E1
  Code for E2
  "ADDn"

It's postfix notation! 2 3 4 + +
```
11/6/14

**Grammar that gives precedence to * over +**

- E → T { + T }
- T → F { * F }
- F → integer
- F → ( E )

**Notation:** { xxx } means 0 or more occurrences of xxx.

**E:** Expression
**T:** Term
**F:** Factor

---

**Does recursive descent always work?**

Some grammars cannot be used for recursive descent

- **Trivial example (causes infinite recursion):**
  - S → b
  - S → Sa

  Can rewrite grammar
  - S → b
  - S → bA
  - A → a
  - A → aA

For some constructs, recursive descent is hard to use

Other parsing techniques exist – take the compiler writing course

---

**Syntactic ambiguity**

- Sometimes a sentence has more than one parse tree
  - S → A | axB
  - A → x | aAb
  - B → b | bB

  axbbb can be parsed in two ways

This kind of ambiguity sometimes shows up in programming languages. In the following, which _then_ does the _else_ go with?

- if E1 then if E2 then S1 else S2

---

**Summary: What you should know**

- preorder, inorder, and postorder traversal. How they can be used to get prefix notation, infix notation, and postfix notation for an expression tree.
- Grammars: productions or rules, tokens or terminals, nonterminals. The parse tree for a sentence of a grammar.
- Ambiguous grammar, because a sentence is ambiguous (has two different parse trees).
- You should be able to tell whether string is a sentence of a simple grammar or not. You should be able to tell whether a grammar has an infinite number of sentences.
- You are not responsible for recursive descent parsing

---

**Exercises**

- Write a grammar and recursive descent parser for sentence palindromes that ignores white spaces & punctuation
  - Was it Eliot's toilet I saw? No trace, not one carton
  - Go deliver a dare, vile dog! Madam, I'm Adam

- Write a grammar and recursive program for strings A^nB^n
  - AB
  - ABB
  - AAAAAABBBBBBB

- Write a grammar and recursive program for Java identifiers
  - `<letter>` | `<letter>` or `<digit>`|^N
  - j27, but not 2j7