Abstract Syntax Trees
Abstract Syntax Tree

• An *abstract syntax tree (AST)* is a *tree* model of an entire program or a certain “program structure” (e.g., a statement or an expression in a Java program)

• An AST is “abstract” in the sense that some of the actual characters used in the “concrete” program text do not appear in the AST
Example: A Java Statement

```java
while (k < 7) {
    foo(k);
    k++;
}
```
Example: A Java Statement

```java
while (k < 7) {
    foo(k);
    k++;
}
```

You should see the connections! (This may not be an actual Java AST, however; it is just an illustration of the idea.)
Example: A BL Statement

WHILE true DO
move
infect
END WHILE
Example: A BL Statement

```
WHILE true DO
  move
  infect
END WHILE
```

You should see the connections! (This is an actual AST for BL; notice it uses a different “design”.)
BL Statement Kinds

instruction

IF test THEN

BLOCK

END IF

ELSE

BLOCK

ELSE

BLOCK

END IF

WHILE test DO

BLOCK

END WHILE
Any sequence of zero or more statements nested in an *IF* or *WHILE* construct is called a *block*. 
**CALL Statement**

<table>
<thead>
<tr>
<th>BL Example</th>
<th>AST</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>turnleft</em></td>
<td><strong>CALL turnleft</strong></td>
</tr>
</tbody>
</table>
### IF Statement

#### BL Example

```
IF next-is-enemy THEN
  turnleft
  move
END IF
```

#### AST

```
IF NEXT_IS_ENEMY

BLOCK

CALL turnleft
CALL move
```

7 March 2013
OSU CSE
### IF_ELSE Statement

<table>
<thead>
<tr>
<th>BL Example</th>
<th>AST</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IF</strong> next-is-enemy <strong>THEN</strong></td>
<td>![AST Diagram](IF_ELSE NEXT_IS_ENEMY)</td>
</tr>
<tr>
<td><strong>turnleft</strong></td>
<td>![CALL Diagram](CALL turnleft)</td>
</tr>
<tr>
<td><strong>ELSE</strong></td>
<td>![CALL Diagram](CALL move)</td>
</tr>
<tr>
<td><strong>move</strong></td>
<td></td>
</tr>
<tr>
<td><strong>END IF</strong></td>
<td></td>
</tr>
</tbody>
</table>
**WHILE Statement**

<table>
<thead>
<tr>
<th>BL Example</th>
<th>AST</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>WHILE next-is-enemy DO</code>&lt;br&gt;<code>turnleft</code>&lt;br&gt;<code>move</code>&lt;br&gt;<code>END WHILE</code></td>
<td><code>WHILE NEXT_IS_ENEMY</code>&lt;br&gt;<code>BLOCK</code>&lt;br&gt;<code>CALL turnleft</code>&lt;br&gt;<code>CALL move</code></td>
</tr>
</tbody>
</table>
Why *BLOCK*?

- Draw the AST for this BL code with and without the intermediate notion of *BLOCK*:

```plaintext
IF next-is-empty THEN
  move
  turnright
ELSE
  infect
END IF
```
Why **BLOCK**?

- Draw the AST for this code with and without the intermediate notion of **BLOCK**:

```
IF next-is-empty THEN
  move
turnright
ELSE
  infect
END IF
```

If it’s not clear, draw the AST for this code with and without **BLOCK**:

```
IF next-is-empty THEN
  move
ELSE
  turnright
  infect
END IF
```
AST Node Labels

• An AST for BL is a *tree of* ... what?
• Each node has *some* of the following:
  – The *kind* of statement (e.g., *BLOCK*, *WHILE*)
  – The *test* condition (e.g., *NEXT_IS_EMPTY*, *TRUE*)
  – The *call* of an instruction (e.g., *infect*, *move*), realizing that this may be an instruction defined elsewhere in the program (e.g., *FindObstacle* in an earlier BL example)
This mathematical 3-tuple of information (of which either test or call might be relevant, depending on the kind) will be called a STATEMENT_LABEL.

- The kind of statement (e.g., BLOCK, WHILE)
- The test condition (e.g., NEXT_IS_EMPTY, TRUE)
- The call of an instruction (e.g., infect, move), realizing that this may be an instruction defined elsewhere in the program (e.g., FindObstacle in an earlier BL example)
The mathematical model of an AST for a BL statement is therefore a tree of `STATEMENT_LABEL`.

- The *kind* of statement (e.g., `BLOCK`, `WHILE`)
- The *test* condition (e.g., `NEXT_IS_EMPTY`, `TRUE`)
- The *call* of an instruction (e.g., `infect`, `move`), realizing that this may be an instruction defined elsewhere in the program (e.g., `FindObstacle` in an earlier BL example)
Resources

• Wikipedia: Abstract Syntax Tree