- **Example: Lazy evaluation**
  - Give `if (false && foo()) . . .`, is `foo()` called?
    - Show them `lazyEval.c`, and ask for the output. Why `foo()` is not executed?
  - The answer depends on the language. Most languages now use **lazy evaluation** (short circuit evaluation). When evaluating `&&` expressions, do it from left to right and stop as soon as you know the value of the expression. Therefore, in the example above, we don’t need to call `foo()`. [C, Java, Python uses lazy evaluation]
  - Similarly, `if (true || foo())` does not require a call to `foo()`. This is important if `foo()` has side effects.
  - **Why is this important?** [show `lazyEval.py`, ask for output. `if (y/x > 3 && x != 0)`]
    - Consider the example `if (x != 0 && y/x > 3) . . .`
    - If `x` is equal to 0, the lazy evaluation prevents the execution of the second expression and so avoid the DivideByZero error.
  - Therefore, the order of expressions in a lazy evaluation matters when the expressions have side effects.
  - Why cares? [make the program more efficient]

---

What are semantics? How do we define semantics?

- The **semantics** of a programming language is a *precise definition of the meaning* of any program that is *syntactically and type-wise correct*
- There are at least three ways to define the semantics
  - **Operational semantics:** the *meaning* of a program is the output produced by a given architecture/compiler pair (pre- and post- increment in C)
  - **Axiomatize semantics:** the *meaning* of a program can be rigorous proven by using a systematic logical argument (formal specification)
  - **Denotational semantics:** the *meaning* of a program can be described as a *collection of meaning functions* operating on the **program state** (this course)

---

What is the program state?

- The **state** of a program is the *binding of all variables to their current values*.
- A variable and its value can be modeled as an *ordered pair*, e.g. `<i, 5>`
- A state is a *set of ordered pairs* (dictionary)

\[
state = \{ <var1,val1>, <var2,val2>, \ldots <var_m,val_m> \}
\]

- A program can have more than one states.
• A state is a snapshot of the memory of the variables used by a program. After each statement execution, the program will have a new snapshot.

- Example
  • The program calculates factorial. Show the states of this program.

<table>
<thead>
<tr>
<th>Snapshot</th>
<th>Before Statement</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 int main () {</td>
<td>4</td>
<td>{&lt;a, undef&gt;, &lt;b, undef&gt;}</td>
</tr>
<tr>
<td>2 int a, b;</td>
<td>5</td>
<td>{&lt;a, 1&gt;, &lt;b, undef&gt;}</td>
</tr>
<tr>
<td>4 a = 1;</td>
<td>7</td>
<td>{&lt;a, 1&gt;, &lt;b, 2&gt;}</td>
</tr>
<tr>
<td>5 b = 2;</td>
<td>9</td>
<td>{&lt;a, 1&gt;, &lt;b, 3&gt;}</td>
</tr>
<tr>
<td>6 b = a + b;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 return 0;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 }</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Trace Table

- Consider an assignment statement. What do assignment statements do?
  • Modify the “state” of the program or the contents of the computer’s memory.
  • How do we indicate that mathematically?
old state $\rightarrow$ new state

- This sounds like a function with input and output. An assignment statement is just a function that takes a state as input and outputs a state.

- An if statement changes the state as well. All Clite statement are just functions from $state_1 \rightarrow state_2$

- We will combine all these functions into one function that takes two parameters.

$$M : \text{Statement} \times \text{State} \rightarrow \text{State}$$

- So

$$M(\text{assignment, oldstate}) = \text{newstate}$$
$$M(\text{ifstatement, oldstate}) = \text{newstate}$$
$$M(\text{blockstatement, oldstate}) = \text{newstate}$$

- In each case, the new state depends both on the old state and the statement being executed.

- BTW, $M$ stands for “meaning”, as in “the meaning of this assignment statement in this state is this new state”.

- In addition to this function, we need two more:

$$M : \text{Program} \rightarrow \text{State}$$
$$M : \text{Expression} \times \text{State} \rightarrow \text{Value}$$

- A value could be any kind of r-value such as an integer, float, or any other mathematically precise object.

- Let’s start implementing $M$ for Clite.

- Note that, since we are thinking of expressions and statements in terms of the AST, and since the meanings of expressions and statements depend on the meaning of their subtrees, guess what we will have when we implement these functions? [Lots of recursion]

---

Program Meaning Function

- The meaning function of $\text{Program}$ can be implemented into this.

$$M : \text{Program} \rightarrow \text{State}$$

$$M(\text{Program } p) = M(p.\text{body}, InitialState(p.\text{decpart}))$$
- If we have a Program \( p \), the implemented function should take two parameters, the body of \( p \) and the initial state of \( p \)'s decpart.

- In this implementation, the InitialState is an auxiliary function that creates a state containing all the variables declared in \( p \)'s decpart paired with their default values.

- For example, if \( \text{int } x = 0, y = 0; \) is \( p \)'s decpart in C, then \( \text{initState}(p\text{.decpart}) = \{<x,0>, <y,0>\} \).

- Note:
  - The "meaning" of a Program is the final state of the computer's memory when the program halts.
  - The \( p \).body part uses recursion. Depends on the statement type in \( p \).body, it will be implemented differently. For example, expression will have a different semantics than assignment.

---

**InitialState in Programming Languages**

- The InitialState function must be clearly defined for a language in order for programs to have meaning. [show initialState.c, initialState.java, initialState2.java]

- C does not initialize the memory space. If not explicit initialization, the initial value of a variable is the random value stored in that memory space. (So, initialState.c gives different results every time you run it)

- Java requires the variables being initialized before using. So initialState.java gives an error for no initialization. But if the variables are fields in a class, even without a constructor, initialState2.java won't complain, since a default constructor will be called. Default constructor will initialize the variables to the default values of their types, e.g. int is 0, float is 0.0 and string is "".

```c
/**
 * initialState.c
 * Ying Li
 * 09/29/2020
 */
#include <stdio.h>
int main () {
    int sum;
    for (int i = 0; i < 10; i++) {
        sum += 5;
    }
    printf("sum = %d \n", sum);
    return 0;
}
```
public class InitialState {
    public static void main (String[] args) {
        int sum;
        for (int i = 0; i < 10; i++) {
            sum += 5;
        }
        System.out.printf("sum = %d \n", sum);
    }
}

public class InitialState2 {
    private int sum;
    public int calSum () {
        int sum = 0;
        for (int i = 0; i < 10; i++) {
            sum += 5;
        }
        return sum;
    }
    public static void main (String[] args) {
        InitialState2 s = new InitialState2();
        System.out.printf("sum = %d \n", s.calSum());
    }
}