CSCI 334:
Principles of Programming Languages

Lecture 19: Variables

Instructor: Dan Barowy
Williams

1

Topics

Variables

Implementing variables

2

Your to-dos

1. Read *Variables* and *Worse is Better* **before the last week of class**.
2. Lab 9 (project checkpoint #1), **due Monday, April 29** (group project).

3

Final project timeline

1. Minimally working version (Lab 9), **due Mon 4/29**
2. Mostly working version (Lab 10), **due Mon 5/13**
3. Project + video presentation, **due Mon 5/20** (last day of exams)

Ward Prize nomination deadline: **May 6**

4

If you want to be considered for the Ward Prize, please let me know and try to get me a version of your project to look at before May 6. There is a cash prize for the Ward Prize!

Variables

5

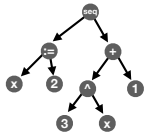
Variables

A **variable** is a named placeholder for a value in an expression. At runtime, when a value is **assigned** to a variable, that **variable name is bound to the value** within the variable's scope. When a variable is **used** in an expression, the bound value is **substituted** into the expression when the expression is evaluated.

6

Example

```
x := 2
3^x + 1
```

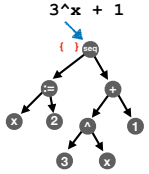


7

Here's a small program in a fantasy language. Observe that this small program's AST has some obvious candidates represented in its AST (numbers, math operations) but that it also contains a couple new things, namely variables, the assignment operator, and something called "seq." This last operation is "sequential composition" and many languages explicitly model it in their ASTs, even though it may not be something a programmer explicitly thinks about. In fact, we should be trying to think deeply about what operations on a computer mean. We *should* explicitly think about the fact that we "execute one line and then the next." One reason is that there are alternative ways of interpreting the meaning of a multi-line program: imagine a language where all of the lines of code are executed in parallel!

Example

```
x := 2
3^x + 1
```



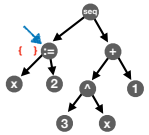
{ } is an "environment"

8

I have claimed many times that we should think of program evaluation as a kind of tree traversal. We will use the same scheme here: a post-order traversal of the tree. So we start at the top. But note that when we introduce variables, we also need to introduce some additional bookkeeping: a data structure called an "environment." For a language with only global variables, it is suffices to think of an environment as a kind of dictionary or map.

Example

```
x := 2
3^x + 1
```



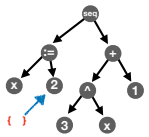
{ } is an “environment”

9

Since seq needs to evaluate its children before it can be evaluated itself, we start by evaluating the first expression.

Example

```
x := 2
3^x + 1
```



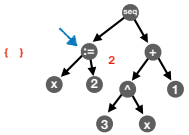
{ } is an “environment”

10

For assignment, we must evaluate the right hand side first.

Example

```
x := 2
3^x + 1
```

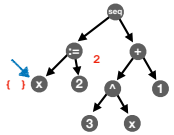


{ } is an “environment”

11

Example

```
x := 2
3^x + 1
```



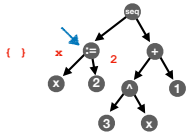
{ } is an “environment”

12

Now we evaluate the left hand side. Specifically, we are checking that the left hand side is actually a variable.

Example

```
x := 2
3^x + 1
```



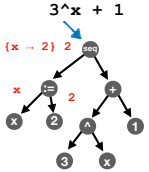
{ } is an “environment”

13

Once we have the evaluated right hand side and we know what variable the left hand side refers to, we can bind the value 2 to the variable x. We create a binding by adding it to our environment.

Example

```
x := 2
3^x + 1
```



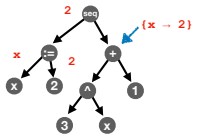
{ } is an “environment”

14

Observe that we pass both the result of evaluating := and the environment back up the tree. This is accomplished by returning both values (as a tuple) in our evaluator. Moreover, observe that here we define := so that it returns a value. Assignment actually does work this way in many languages, most notably C.

Example

```
x := 2
3^x + 1
```



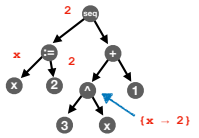
{ } is an “environment”

15

Now that we’ve evaluated the first expression in the seq, we turn to the second. Observe that the environment containing the binding for x follows us down the tree.

Example

```
x := 2
3^x + 1
```



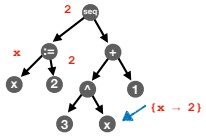
{ } is an “environment”

16

Keep going to get the operands for exponentiation.

Example

```
x := 2
3^x + 1
```



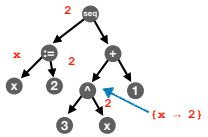
{ } is an “environment”

17

Here, we need to know what value x has. This corresponds to a lookup in our environment. If x were not bound in our environment, this would be a runtime program error. You’ve probably seen this error before in Python. It looks something like “undefined variable.” Note that Java does not have this issue, because the compiler can prove that every variable is either defined or not; if a variable is undefined, then the compiler refuses to compile the program.

Example

```
x := 2
3^x + 1
```



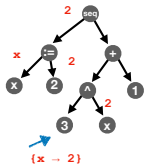
{ } is an “environment”

18

Now turn to the left hand side. Note that I chose to evaluate the exponent before the base as per our rules of arithmetic, but you could make another choice if you wanted to (but perhaps not while preserving the expected result using classic arithmetic).

Example

```
x := 2
3^x + 1
```

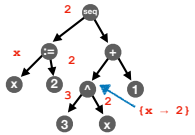


{ } is an “environment”

19

Example

```
x := 2
3^x + 1
```



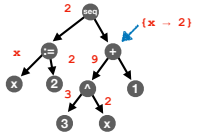
{ } is an “environment”

20

Finally we can evaluate the exponentiation.

Example

```
x := 2
3^x + 1
```



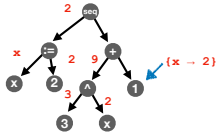
{ } is an “environment”

21

Evaluate LHS of plus.

Example

```
x := 2
3^x + 1
```

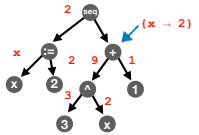


{ } is an “environment”

22

Example

```
x := 2
3^x + 1
```



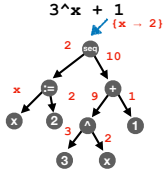
{ } is an “environment”

23

Compute plus now.

Example

```
x := 2
3^x + 1
```



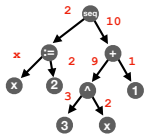
{ } is an “environment”

24

Example

```
x := 2
3^x + 1
```

10



{ } is an “environment”

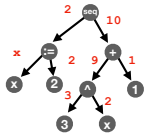
25

And finally, seq itself must return something. If we follow the rule that F#, Scala, and many other functional programming languages use, we return the last expression evaluated, in this case the result of the addition.

Example

```
x := 2
3^x + 1
```

10



Cool, huh?

Every CS major should know this.

26

Knowing that this is how a program is evaluated is a serious superpower when trying to debug. Keep in mind that the precise rules vary from language to language! But if you want to be a killer hacker, look those rules up. Every programming language worth using is very clear about the meaning (i.e., “semantics”) of each datum and operation.

How does it work?

27

So how can we make something like this happen in a real language? See the “blub” language posted on the course website for a complete example. You may borrow any of the code you see there for your own projects.

Recap & Next Class

Today:

Variables

Next class:

Scope / Midterm review

28