CSCI 334:
Principles of Programming Languages

Lecture 17: Variables

Instructor: Dan Barowy

Williams

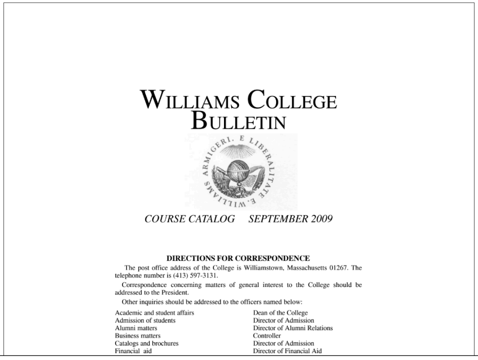
Topics

Variables

Implementing variables

Announcements

- Friday Colloquium: **Pre-registration Info Session**, 2:35pm in Wege Auditorium.



WILLIAMS COLLEGE
BULLETIN

COURSE CATALOG SEPTEMBER 2009

DIRECTIONS FOR CORRESPONDENCE

The post office address of the College is Williamstown, Massachusetts 01267. The telephone number is (413) 597-3131.

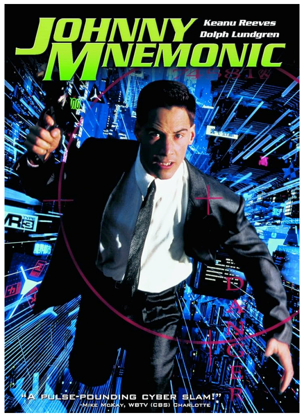
Correspondence concerning matters of general interest to the College should be addressed to the President.

Other inquiries should be addressed to the officers named below:

Academic and student affairs	Dean of the College
Admission of students	Director of Admission
Alumni matters	Director of Alumni Relations
Business matters	Controller
Catalogs and brochures	Director of Admission
Financial aid	Director of Financial Aid

Announcements

- **Johnny Mnemonic**, **TONIGHT!, Apr 24 @ 7pm in Wege Auditorium**



Benefits:

- **Fun!**
- **Snacks!**
- You will finally be able to understand your professor's jokes!
- You will be able to converse fluently with other nerds!
- You *might* learn a little computer science!
- Did I mention snacks?!!
- **Sponsored by Jim Bern**

Your to-dos

1. Read for next week: **Implementing Variables, Implementing Scope**.
2. Next week's quiz will be on **type inference**. Refer to slides and practice problem for study.
3. Lab 10, **Project checkpoint #3**, due **Wednesday, April 30 by midnight**.

Final project timeline

1. ~~Brainstorm (Lab 7), **due Wed 4/9**~~
2. ~~Project Proposal (Lab 9), **due Wed 4/23**~~
3. Minimally working version (Lab 10), **due Wed 4/30**
4. Final project + video presentation (Lab 11), **due Wed 5/14**

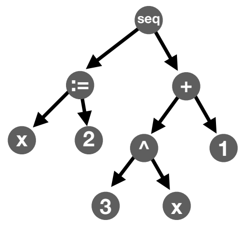
Variables

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.

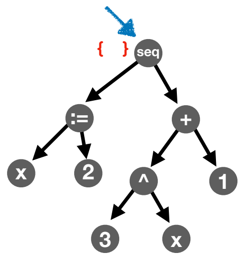
Example

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



Example

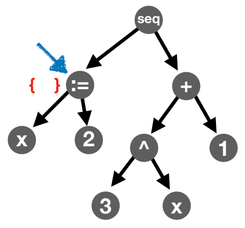
```
x := 2
3^x + 1
```



{ } is an “environment”

Example

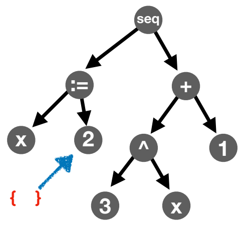
```
x := 2
3^x + 1
```



{ } is an “environment”

Example

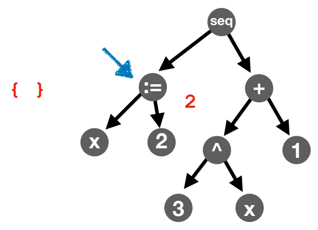
```
x := 2
3^x + 1
```



{ } is an “environment”

Example

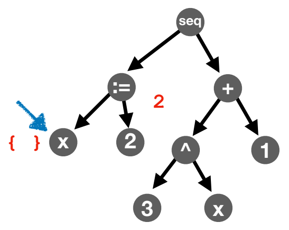
```
x := 2
3^x + 1
```



{ } is an “environment”

Example

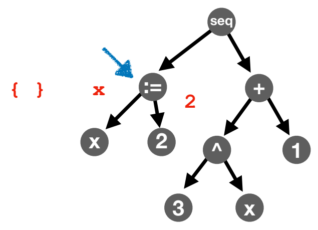
```
x := 2
3^x + 1
```



{ } is an “environment”

Example

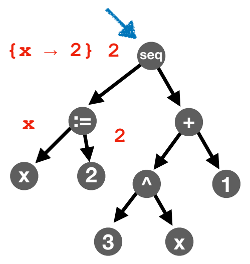
```
x := 2
3^x + 1
```



{ } is an “environment”

Example

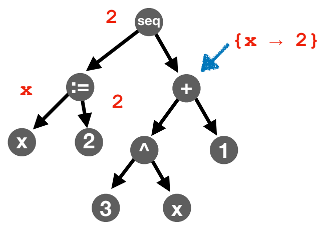
```
x := 2
3^x + 1
```



{ } is an “environment”

Example

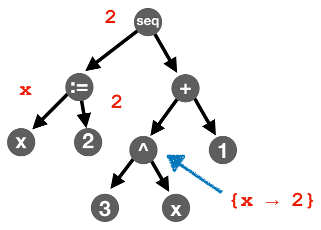
```
x := 2
3^x + 1
```



{ } is an “environment”

Example

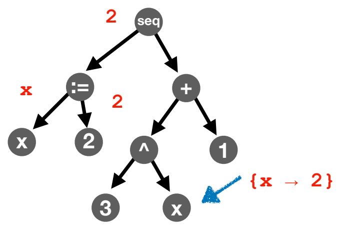
```
x := 2
3^x + 1
```



{ } is an “environment”

Example

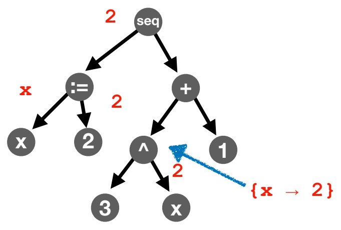
```
x := 2
3^x + 1
```



{ } is an “environment”

Example

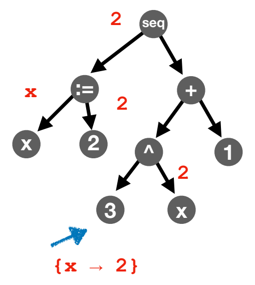
```
x := 2
3^x + 1
```



{ } is an “environment”

Example

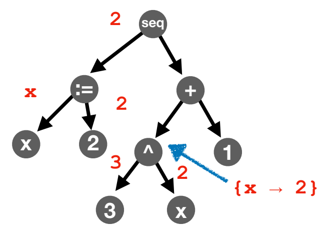
```
x := 2
3^x + 1
```



{ } is an "environment"

Example

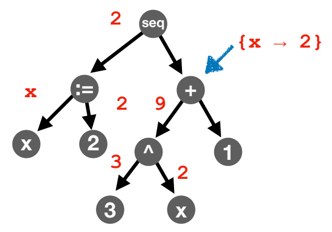
```
x := 2
3^x + 1
```



{ } is an "environment"

Example

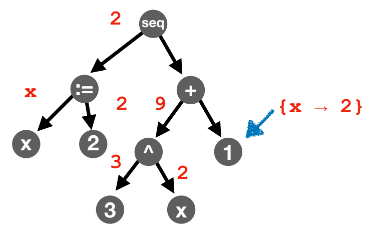
```
x := 2
3^x + 1
```



{ } is an "environment"

Example

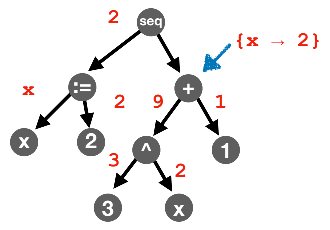
```
x := 2
3^x + 1
```



{ } is an "environment"

Example

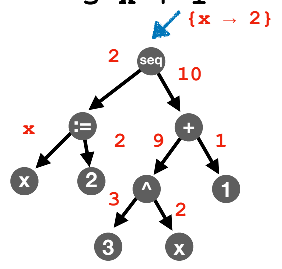
```
x := 2
3^x + 1
```



{ } is an “environment”

Example

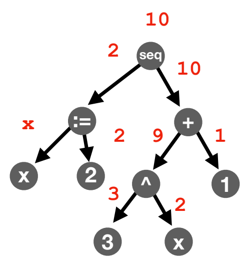
```
x := 2
3^x + 1
```



{ } is an “environment”

Example

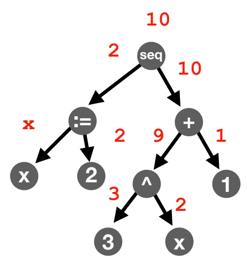
```
x := 2
3^x + 1
```



{ } is an “environment”

Example

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



Cool, huh?

Every CS major should know this.

How does it work?

Blub

```
   <expr> ::= <parens>
           |  <print>
           |  <assign>
           |  <string>
           |  <num>
           |  <var>
           |  <plus>
 <parens> ::= ( <expr> )
  <print> ::= print <expr>
 <assign> ::= <var> := <expr>
 <string> ::= " <char> "
   <char> ::= [any character that is not double quote]
    <num> ::= <digit>+
    <var> ::= <ltr><varchar>+
<varchar> ::= <ltr> | <digit>
    <ltr> ::= A | .. | Z | a | .. | z
  <digit> ::= 0 | .. | 9
  <plus>  ::= <expr> + <expr>
```

(code)

Project Activity

Project Activity

Find a partner **who is not your project partner**.

I will prompt you when to move to the next step in the procedure below.

Each of you will do the following in turn:

1. (~3 minutes) **Explain your project** to your partner. Be sure to discuss at least one primitive and one combining form, and be sure to describe the form of the input and the form of the output.
2. (~3 minutes) Your partner **explains your project back to you**. Take note which concepts they have trouble explaining back to you.
3. **Swap roles** and **go to step 1**.

Recap & Next Class

Today:

Variables

Next class:

Scope/Packages