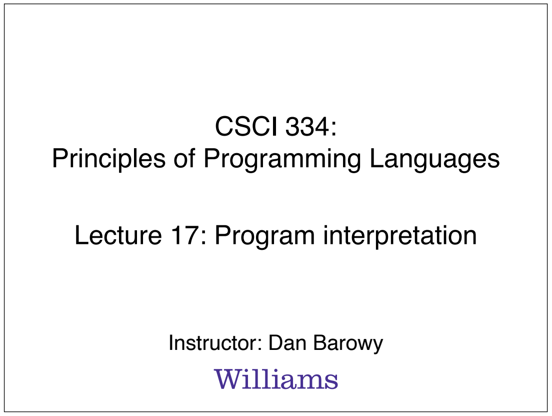
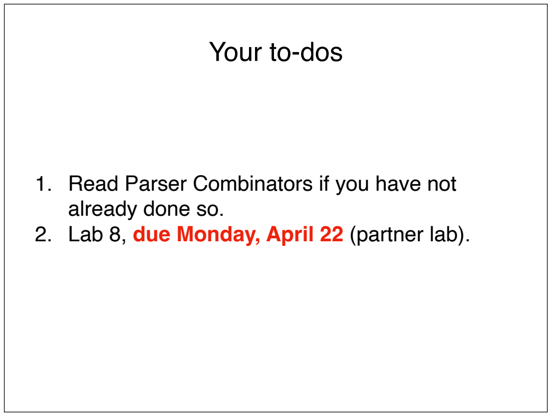


CSCI 334:
Principles of Programming Languages

Lecture 17: Program interpretation

Instructor: Dan Barowy
Williams

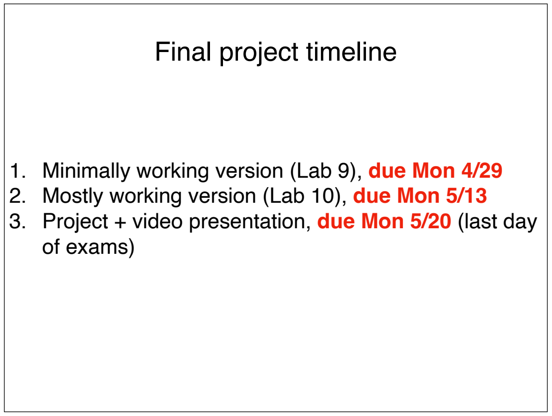
1



Your to-dos

1. Read Parser Combinators if you have not already done so.
2. Lab 8, **due Monday, April 22** (partner lab).

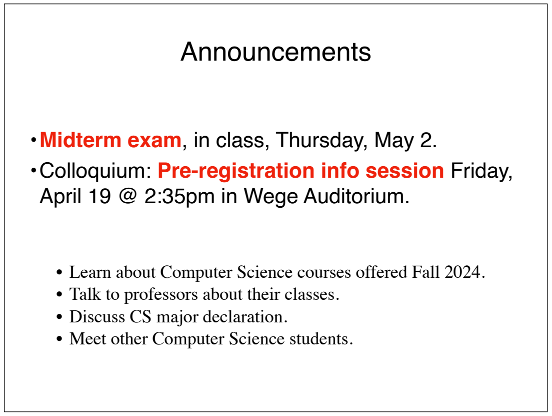
2



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)

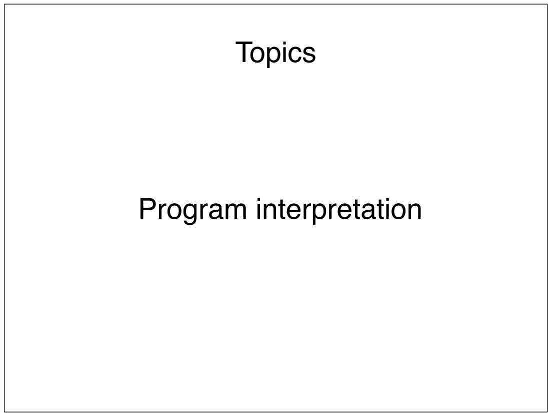
3



Announcements

- **Midterm exam**, in class, Thursday, May 2.
- Colloquium: **Pre-registration info session** Friday, April 19 @ 2:35pm in Wege Auditorium.
    - Learn about Computer Science courses offered Fall 2024.
    - Talk to professors about their classes.
    - Discuss CS major declaration.
    - Meet other Computer Science students.

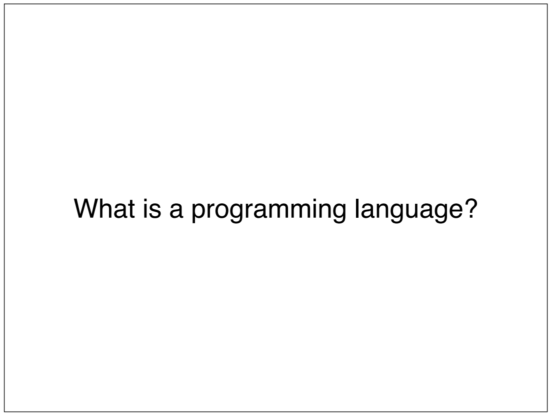
4



Topics

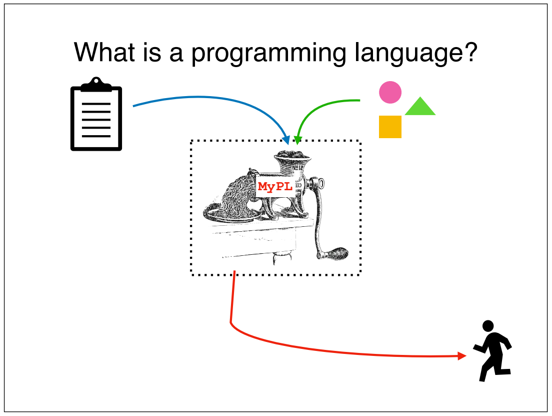
Program interpretation

5



What is a programming language?

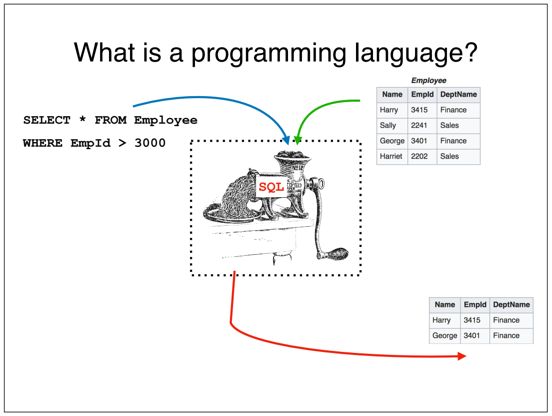
6



What is a programming language?

7

Recall that a programming language is just a function. That function typically takes two inputs: 1) the program itself, as a string, and 2) the inputs to program, in whatever form is necessary. The output can be anything (it depends on the language).



What is a programming language?

```
SELECT * FROM Employee
WHERE EmpId > 3000
```

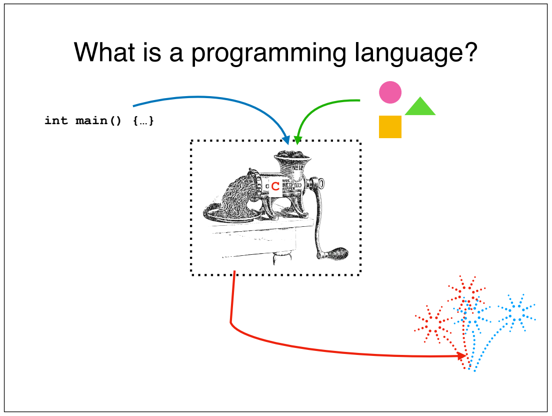
Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Name	EmpId	DeptName
Harry	3415	Finance
George	3401	Finance

8

For example, here is the SQL language. SQL is widely used for data manipulation tasks. The input to SQL is a query and a set of database tables. A database table can be thought of as something like a spreadsheet. The output of a SQL query is a database table.

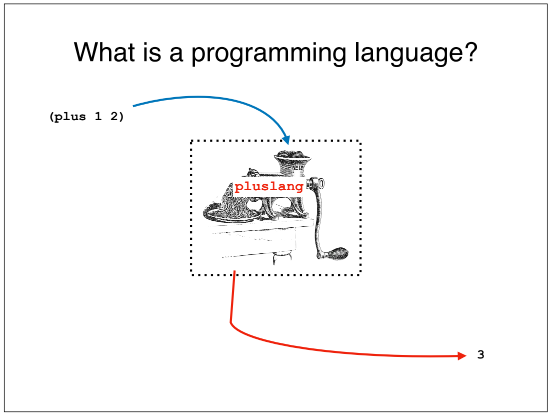


What is a programming language?

```
int main() {…}
```

9

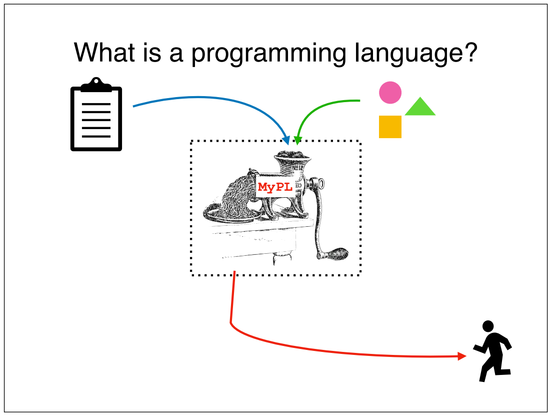
C is also a machine that takes a C program and some input. However, most C implementations are compiled, so the output is usually a sequence machine instructions. Running those instructions produces some effect on the machine.



What is a programming language?

10

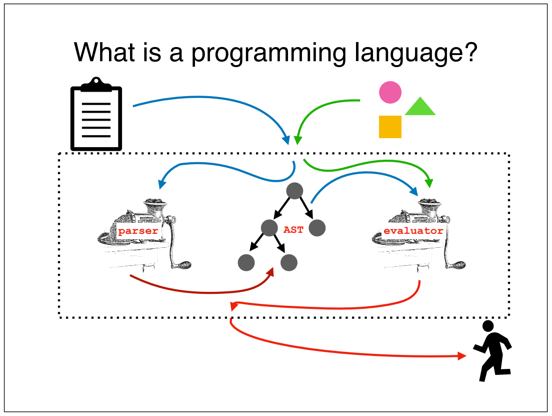
We are going to talk about a simple language today called pluslang. It does not take any input other than the program itself.



What is a programming language?

11

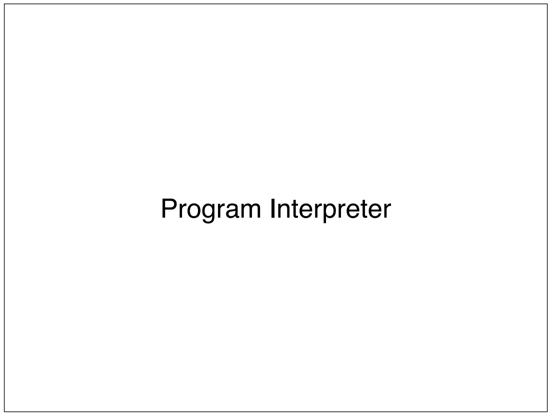
Let’s expand on what’s inside this MyPL box.



What is a programming language?

12

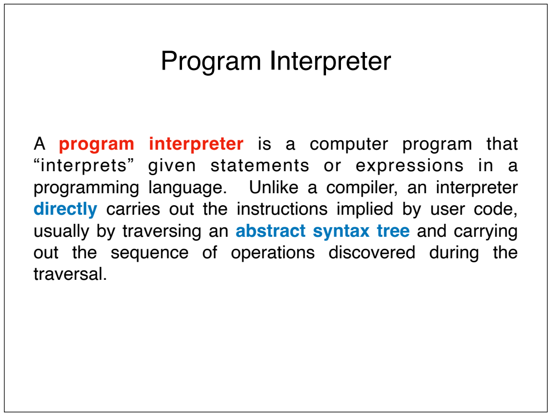
A programming language implementation usually has a frontend that does lexical analysis tasks, like parsing. The result of the frontend is an AST. The AST is then given as input to the backend, along with the input to the program. The backend then evaluates the AST. If the language is interpreted, the result of an evaluation is some effect on the computer. If the language is compiled, the result of evaluation is the program in a different form (e.g., machine code).



Program Interpreter

13

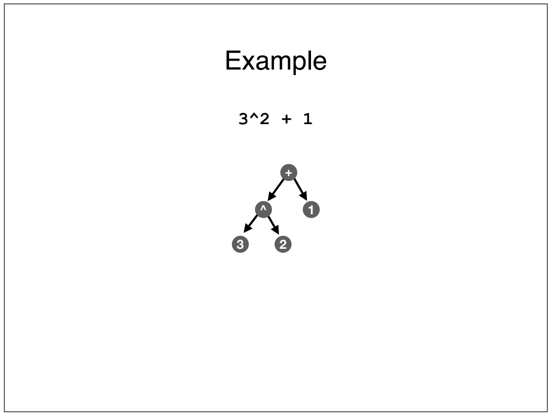
We are going to focus on interpretation.



Program Interpreter

A **program interpreter** is a computer program that "interprets" given statements or expressions in a programming language. Unlike a compiler, an interpreter **directly** carries out the instructions implied by user code, usually by traversing an **abstract syntax tree** and carrying out the sequence of operations discovered during the traversal.

14

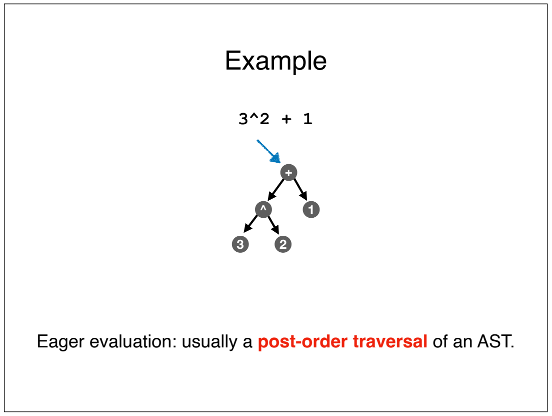


Example

```
3^2 + 1
```

15

Here's an example program and an AST the language's parser might produce. Observe that the operations that need to be carried out first are near the bottom of the tree. Ensuring that the tree has the correct form is the job of the frontend.



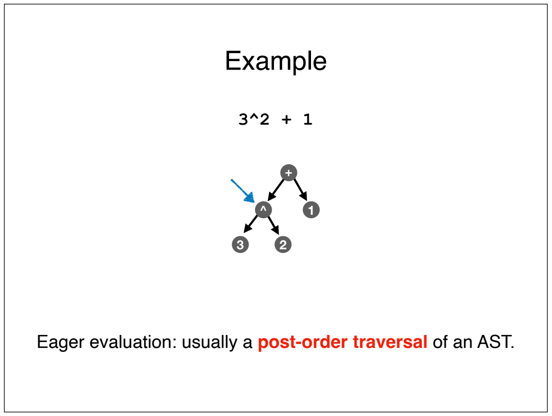
Example

`3^2 + 1`

Eager evaluation: usually a **post-order traversal** of an AST.

16

When we evaluate an AST (using a scheme called “eager evaluation”), the algorithm usually follows something like a post-order traversal of the AST. Let’s walk through this example. We start at the root. Recall in a post-order traversal, we visit the children before we visit the node, and since this node (+) has children, we recursively move on to those. The rationale is that any node with children is an operation, and we need to realize the values of the operands (which can themselves be operations) before we can do the operation.



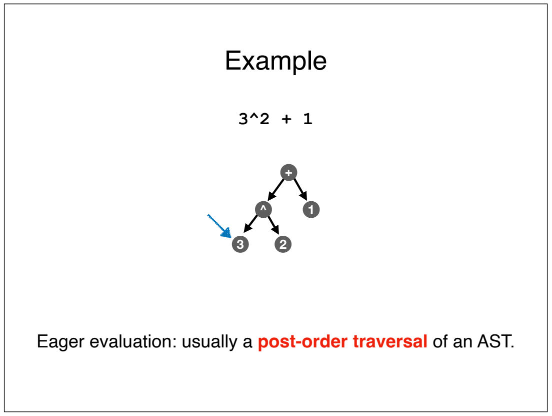
Example

`3^2 + 1`

Eager evaluation: usually a **post-order traversal** of an AST.

17

We visit the first child, which actually is itself an operation. Therefore, we must also visit its children recursively before we can do the operation.



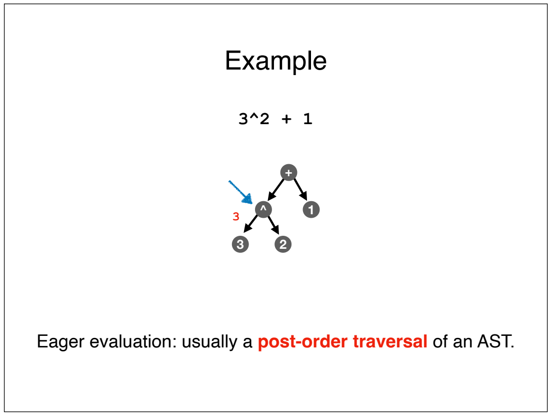
Example

`3^2 + 1`

Eager evaluation: usually a **post-order traversal** of an AST.

18

Finally, we find a node with no children. The evaluation of 3 is just 3, so we return that result.

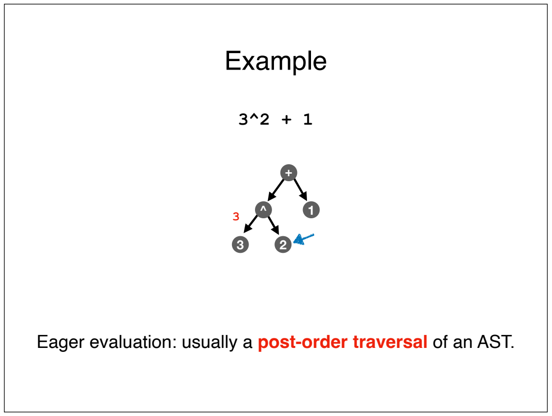


Example

`3^2 + 1`

Eager evaluation: usually a **post-order traversal** of an AST.

19 We must also visit the node’s other child.

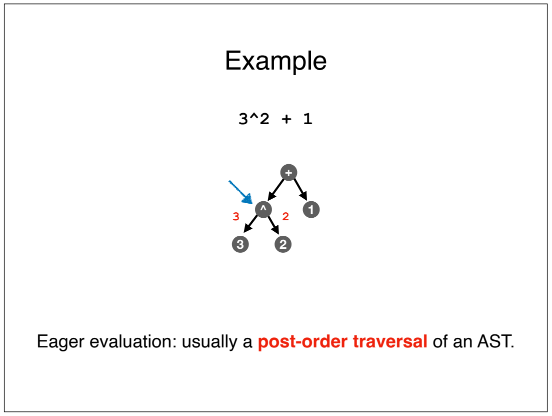


Example

`3^2 + 1`

Eager evaluation: usually a **post-order traversal** of an AST.

20 The result of 2 is 2.

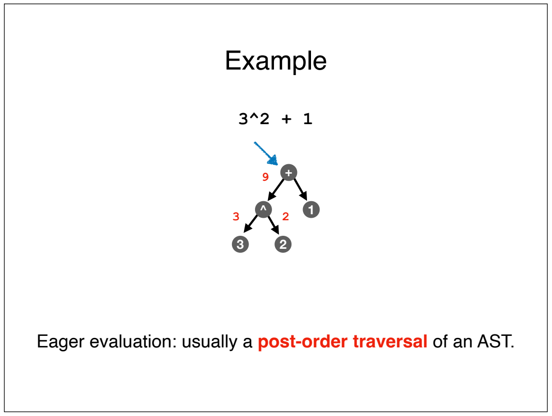


Example

`3^2 + 1`

Eager evaluation: usually a **post-order traversal** of an AST.

21 Now that we have results from both children of ^, we can evaluate 3^2. Note that we can use exponentiation if that is available to us in the language we are writing our programming language in, but if not, we may need to implement it ourselves (e.g., by using the machine’s left shift operation). We return the result of the exponentiation.



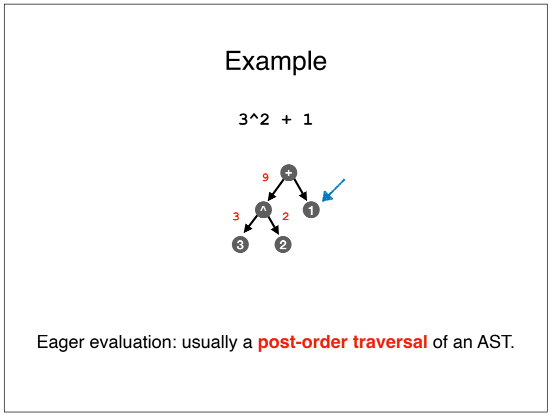
Example

`3^2 + 1`

Eager evaluation: usually a **post-order traversal** of an AST.

22

The + node still has an unevaluated child, so we recursively evaluate the child on the right.



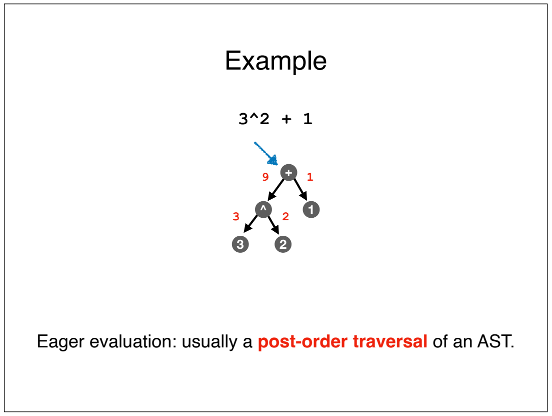
Example

`3^2 + 1`

Eager evaluation: usually a **post-order traversal** of an AST.

23

The result of evaluating 1 is 1.



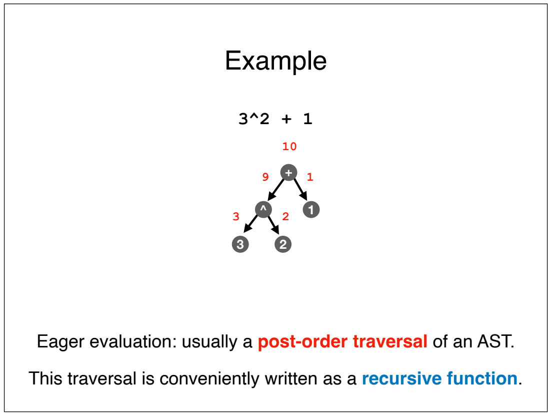
Example

`3^2 + 1`

Eager evaluation: usually a **post-order traversal** of an AST.

24

Finally, we can carry out and return the result of adding 9 and 1.



Example

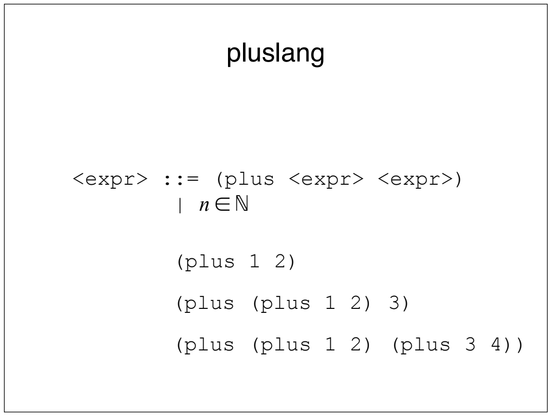
`3^2 + 1`

Eager evaluation: usually a **post-order traversal** of an AST.

This traversal is conveniently written as a **recursive function**.

25

This entire procedure can often be written as a recursive function.

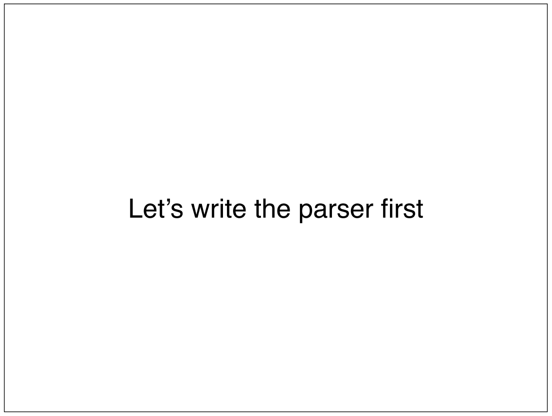


pluslang

```
<expr> ::= (plus <expr> <expr>)
         | n∈ℕ

           (plus 1 2)

           (plus (plus 1 2) 3)

           (plus (plus 1 2) (plus 3 4))
```

26

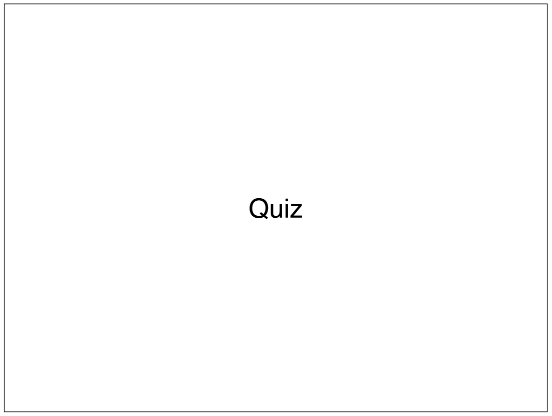
Here is the grammar for the language we are going to implement in class. It is a small language, so we will be able to write the entire thing right now. Here are also some example programs.



Let’s write the parser first

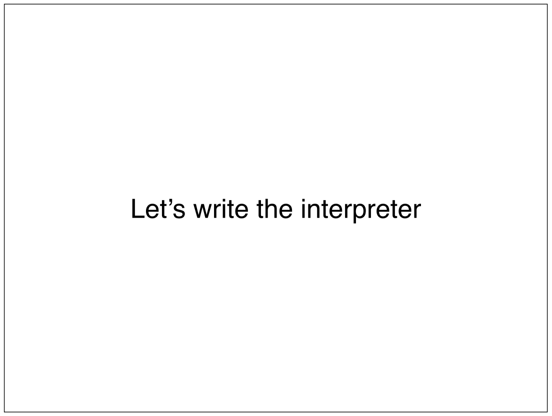
27

Start with the parser. See posted code.



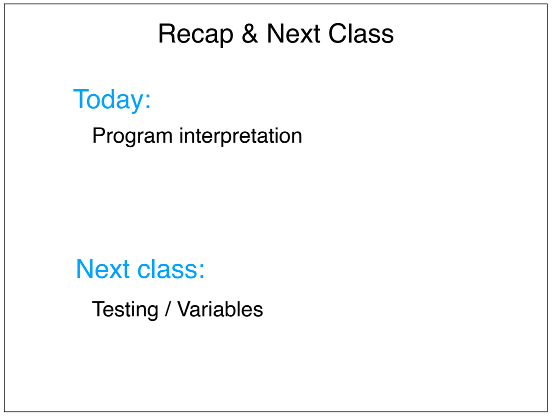
Quiz

28 Hopefully, that was enough of a refresher on parsing that you are ready for a quiz on parsing.



Let’s write the interpreter

29 Let’s now turn to the interpreter. The code is posted.



Recap & Next Class

Today:

Program interpretation

Next class:

Testing / Variables

30