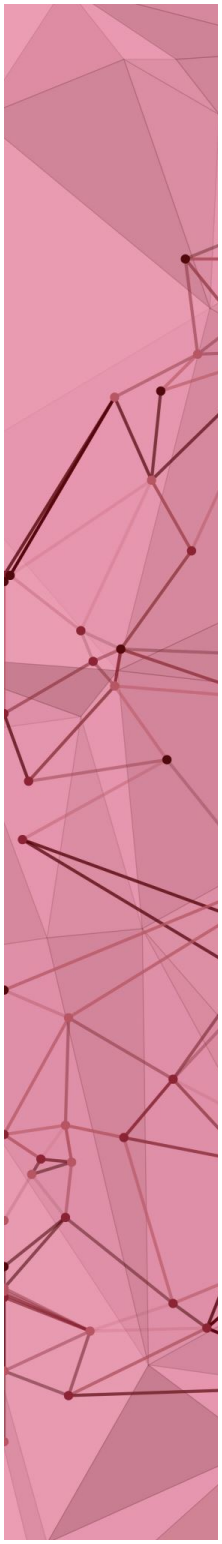
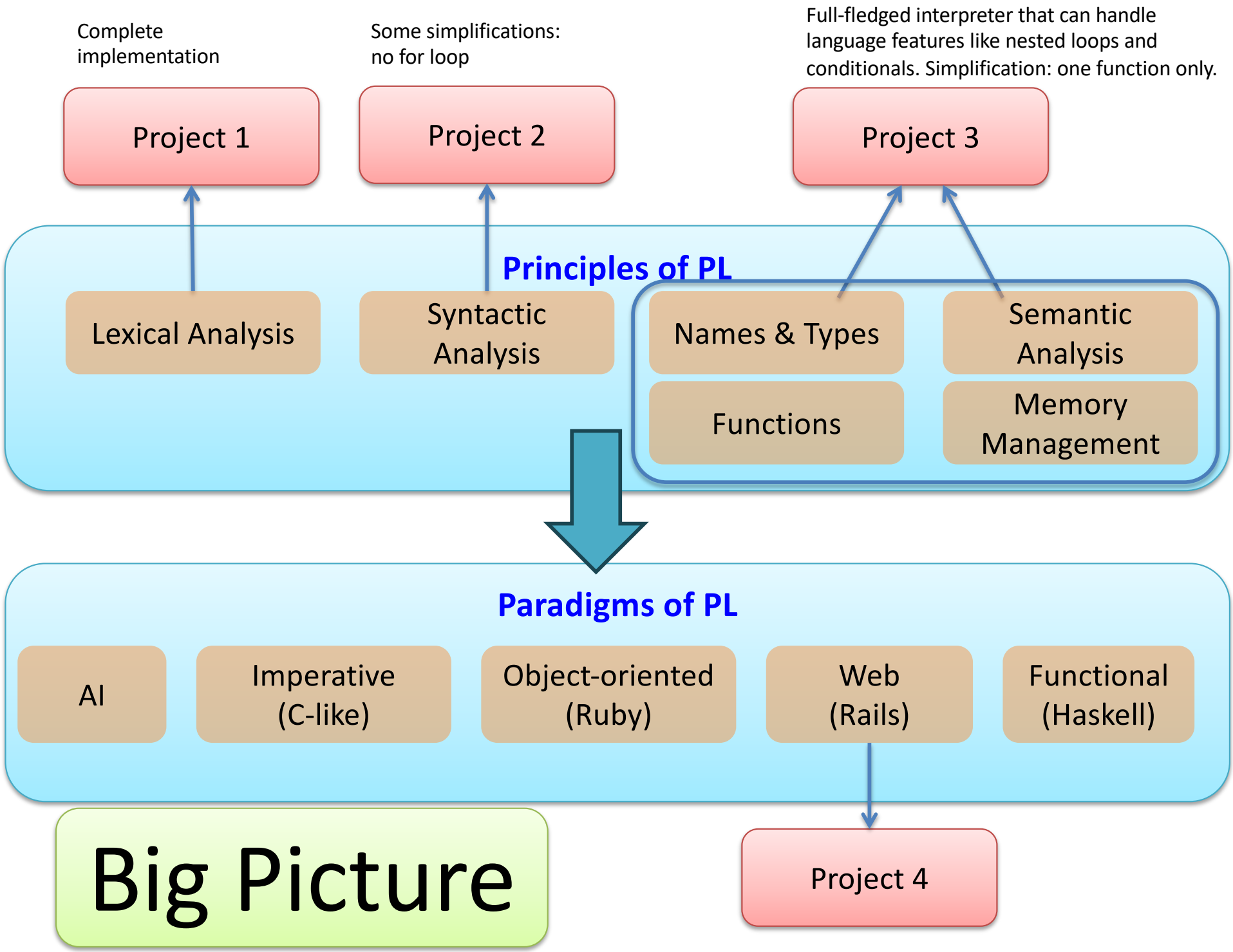
CSCI 2320
Principles of Programming Languages

Types & Type System

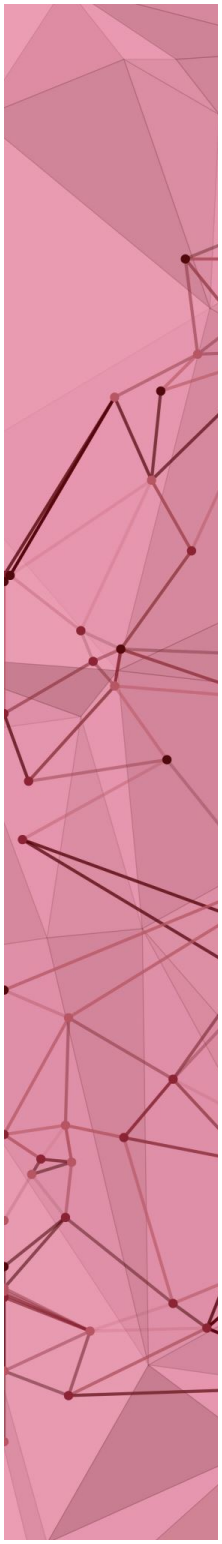
Reading: Ch 5 & 6 (Tucker & Noonan)




Complete implementation
Some simplifications: no for loop
Full-fledged interpreter that can handle language features like nested loops and conditionals. Simplification: one function only.
Project 1
Project 2
Project 3
Principles of PL
Lexical Analysis
Syntactic Analysis
Names & Types
Semantic Analysis
Functions
Memory Management
Paradigms of PL
AI
Imperative (C-like)
Object-oriented (Ruby)
Web (Rails)
Functional (Haskell)
Big Picture
Project 4

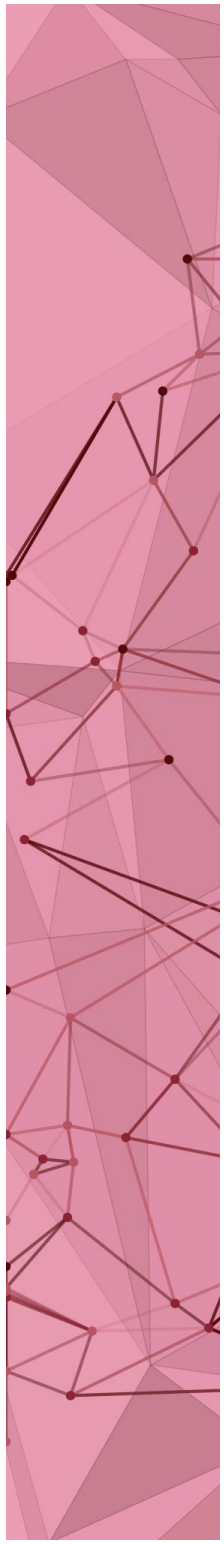
Type

- Definition
- Examples
- Question 1: Will the values be bounded?
- Question 2: Will the types be pre-defined?
- Type errors
- Type systems



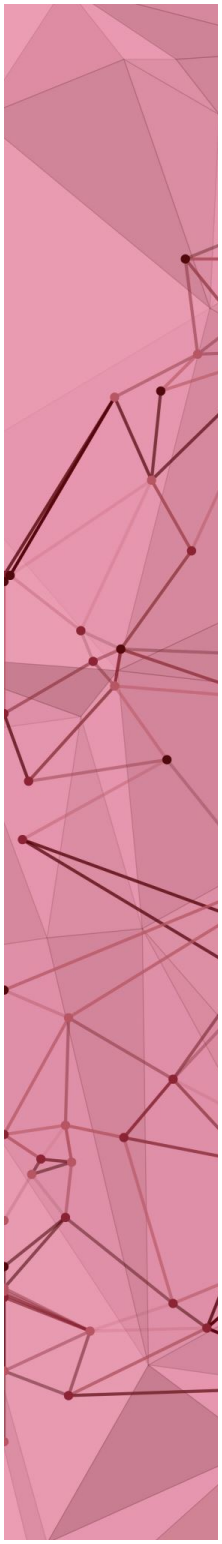
Type

- A *type* is a collection of values and operations on those values.
- Example
  - Integer type has values ..., -2, -1, 0, 1, 2, ... and operations +, -, *, /, <, …
  - Boolean type has values true and false and operations $\wedge$, $\vee$, $\neg$.
- Question 1: Will the values be bounded?
- Question 2: Will the types be pre-defined?



Type errors & Type systems

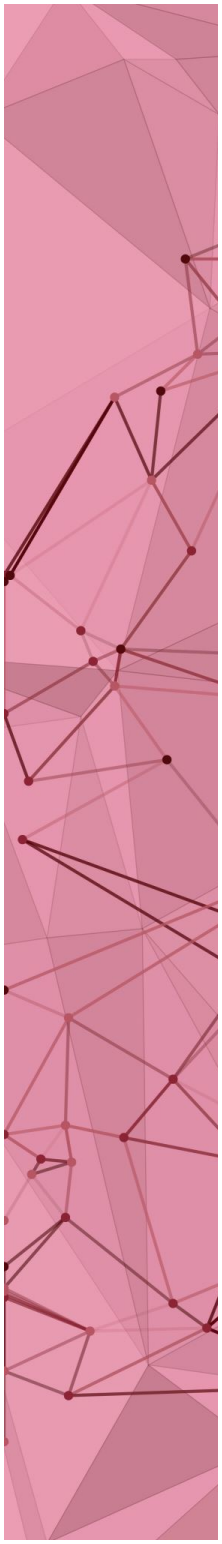
- *Type error:* Incompatibility between data type and operations
- *Type system:* Detects type errors
  - Can’t we detect type errors just by looking at the values?

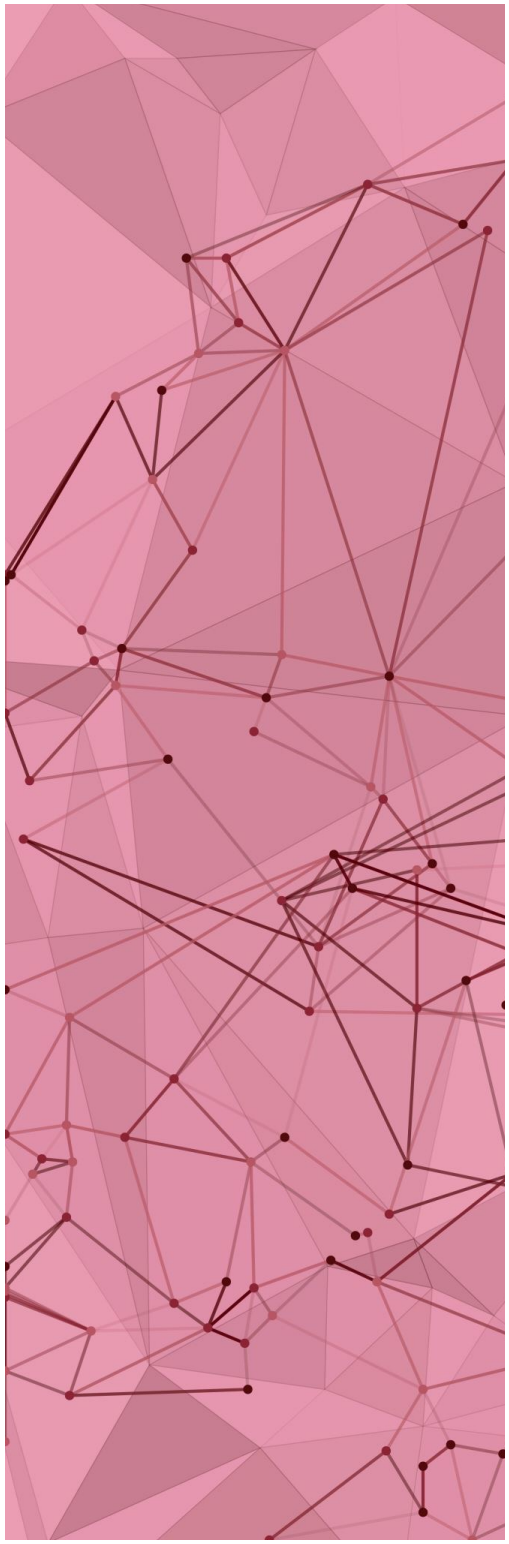


Can’t we detect type errors just by looking at the value?

Machine level:
0100 0000 0101 1000 0000 0000 0000 0000

- The floating point number 3.375
- The 32-bit integer 1,079,508,992
- Two 16-bit integers 16472 and 0
- Four ASCII characters: @ X NUL NUL





Type Systems

Static typing

Dynamic typing

Strongly typed language

Untyped language

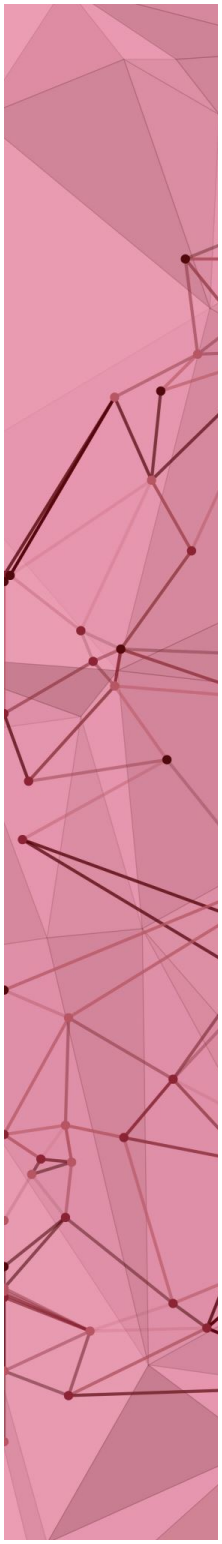
Java: Why dynamic typing for type casting?

```
public class Dyn_Typing
{
    public static Object f()
    {
        int x = Integer.parseInt(System.console().readLine("Enter a number: "));
        if (x%2 == 0) //x is even
        {
            String st = "Even number!";
            return st;
        }
        else
        {
            return new Integer(x);
        }
    }

    public static void main(String[] args)
    {
        String st = (String) f();
        System.out.println(st);
    }
}
```

Confusion around strongly typed languages

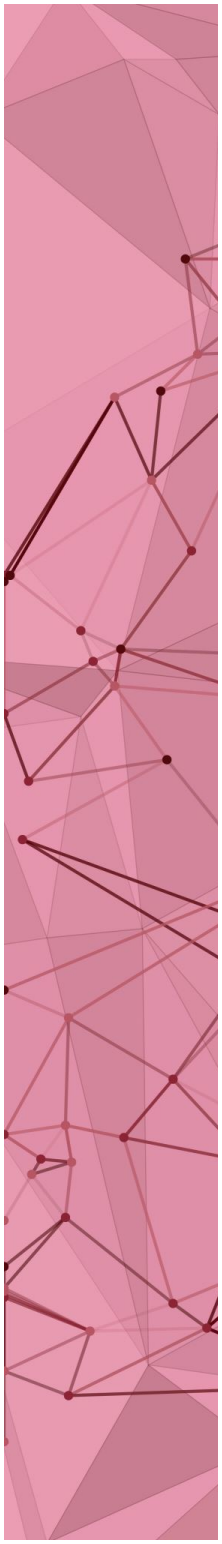
- http://ericlippert.com/2012/10/15/is-c-a-strongly-typed-or-a-weakly-typed-language/
- http://c2.com/cgi/wiki?StronglyTyped



Is C strongly typed?

No, because errors can go undetected.

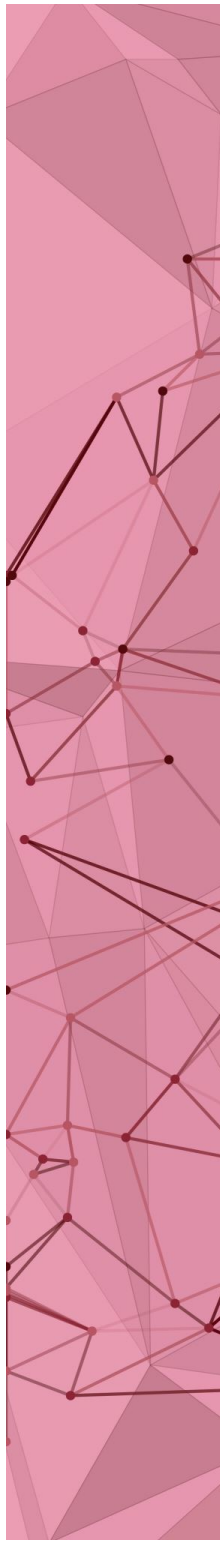
```
#include <stdio.h>
union {int i; float f;} u;
int main()
{
  float x = 0.0;
  u.i = 987654321;
  x = x + u.f;
  printf ("%f\n", x);
  return 0;
}
```

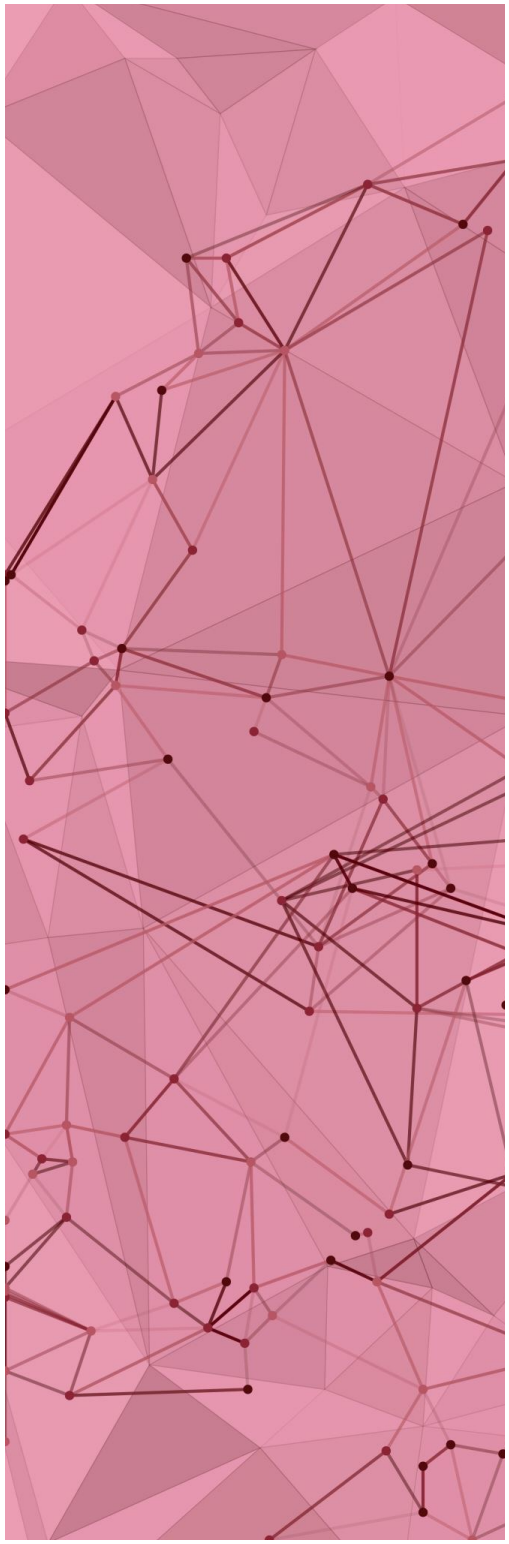


Question

Discuss and write down any potential connections between dynamic/static scoping and dynamic/static typing.

- Think about all possible combinations
- Give an example for each (if there exists one)



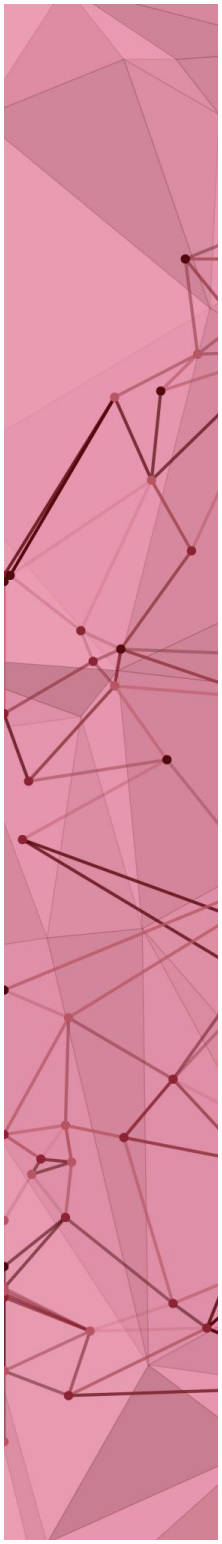


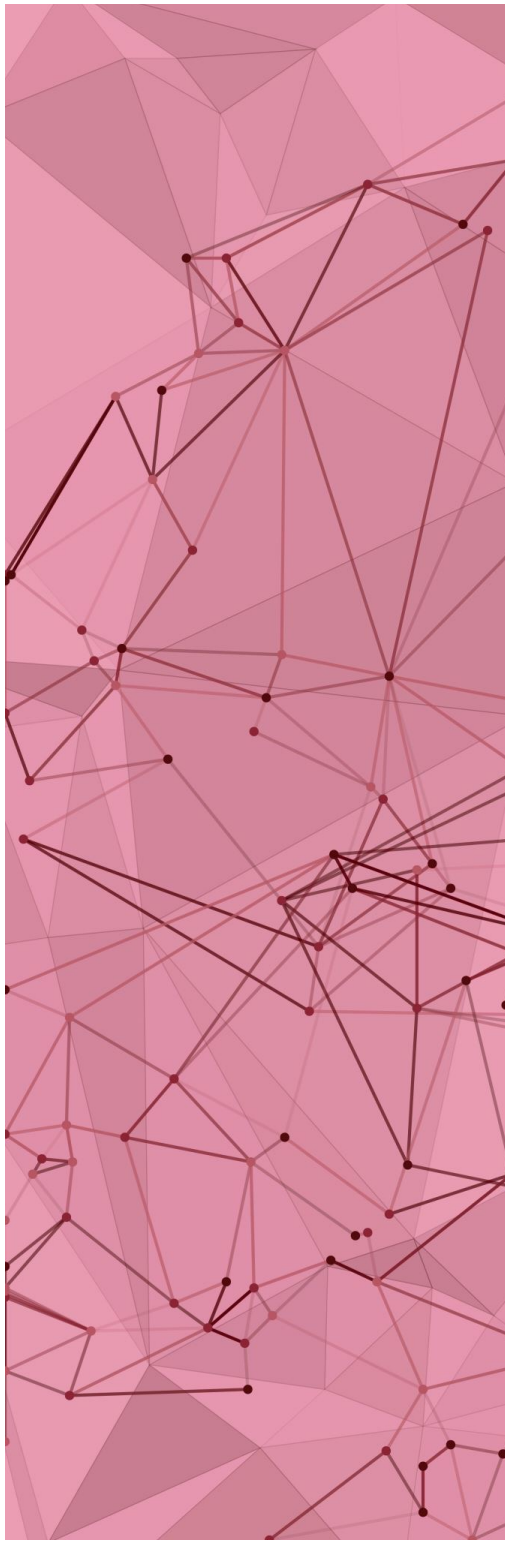
Type Systems

Implementation Issues

Basic vs. non-basic types

- Basic: int, float, bool, char
  - Implicit type conversion: narrowing vs. widening
  - C vs Java vs Python
- Non-basic: pointer, string, array, list, etc. and programmer-defined types
  - Implementation issues





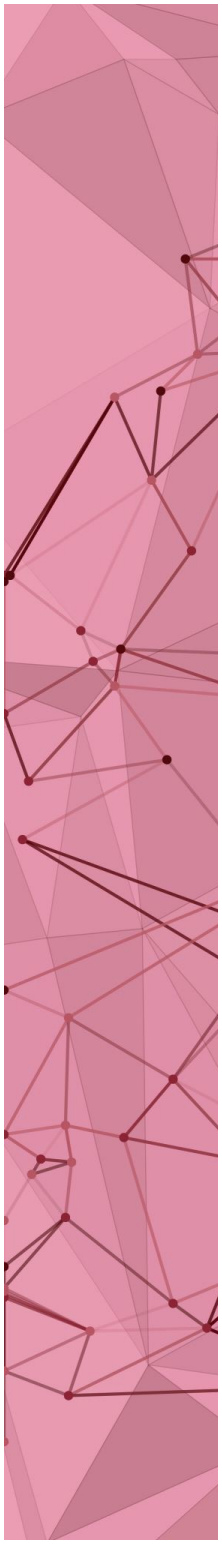
Implementation of non-basic types

Arrays in C and Java

Other non-basic: read the book

Array implementation

“Dope vector” + actual array

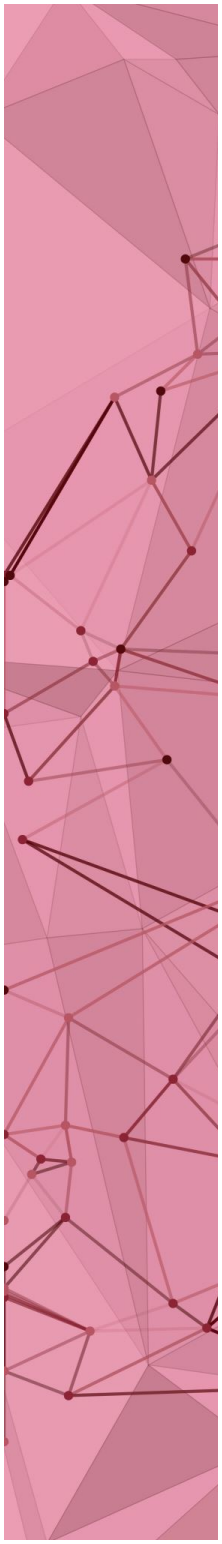


Arrays: C vs. Java

C	Java
Static allocation int a[5];	Dynamic allocation int[] a = new int(5);
Elements must be stored contiguously Why? Example: strcpy	No such requirements
Local arrays not auto-initialized to 0s	Auto-initialized to 0s
Index out-of-bound not checked	Checked

List vs. array

- Size of a list may vary at run-time
- List elements may not be contiguous in memory
- List may contain heterogeneous data (Python)

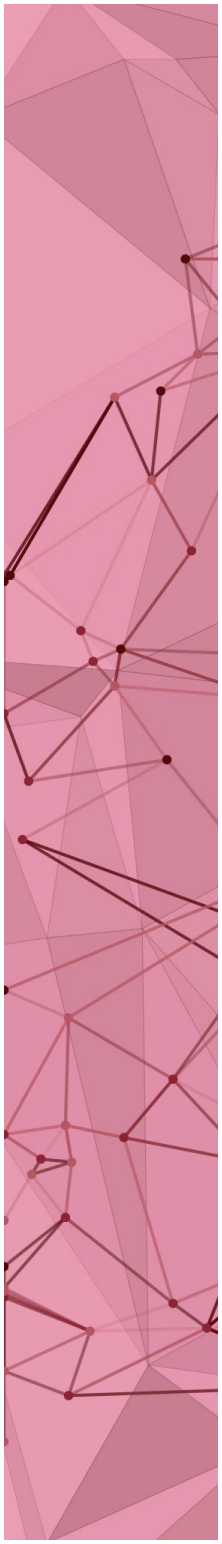


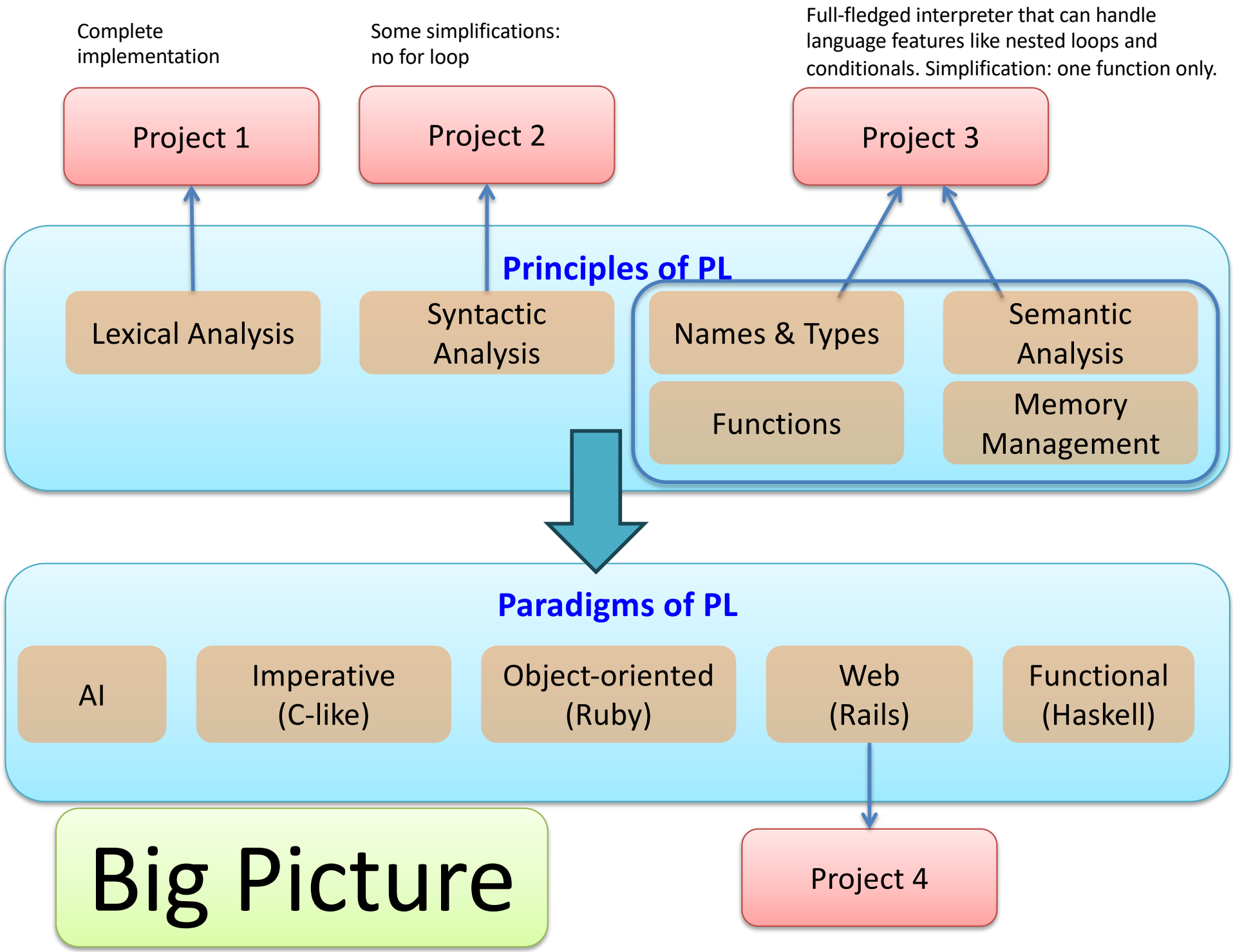
Other issues (read the book)

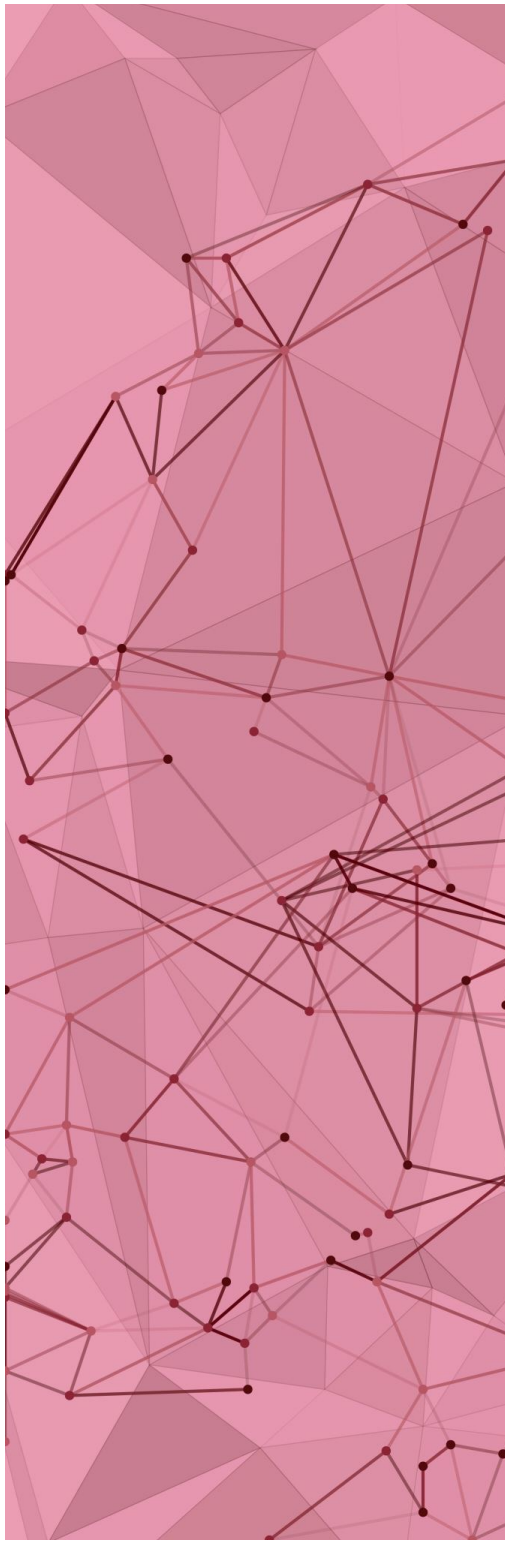
- Functions as types in C
- void qsort(

```
        void *array,
        size_t nitems,
        size_t size,
        int (*compare)(const void *, const void*)
)
```

- Functions as types in Java?
- Sub-types
  - OOP: Subclass
- Polymorphism




Complete implementation
Some simplifications: no for loop
Full-fledged interpreter that can handle language features like nested loops and conditionals. Simplification: one function only.
Project 1
Project 2
Project 3
Principles of PL
Lexical Analysis
Syntactic Analysis
Names & Types
Semantic Analysis
Functions
Memory Management
Paradigms of PL
AI
Imperative (C-like)
Object-oriented (Ruby)
Web (Rails)
Functional (Haskell)
Big Picture
Project 4

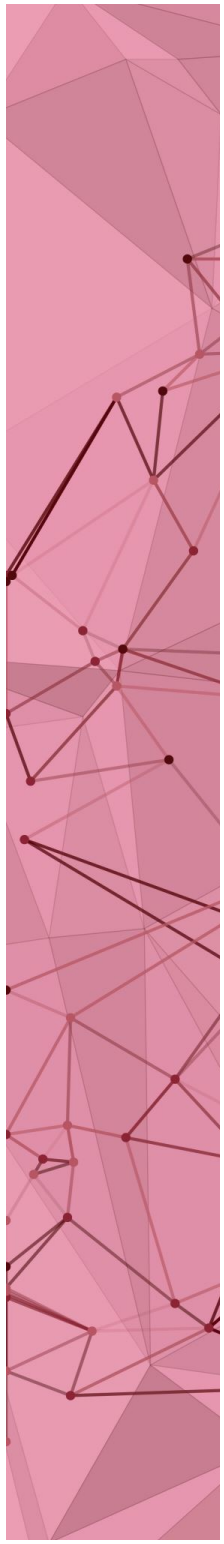


Review: Names

Static and dynamic scoping

- Static scoping: a name is bound to a collection of statements according to its position in the source program.
- Dynamic scoping: binding depends on the control flow of the program.

- Most modern languages use static (or *lexical*) scoping.



Symbol table data structure:
Stack of dictionaries

Each dictionary:

{<name, binding>, <name, binding>, …}

Symbol table data structure: Stack of dictionaries

Algorithm:

- { : push a new dictionary
- } : pop the top dictionary
- Var declaration: insert its binding into the top dictionary
- Name reference (not var dec): search stack top to bottom

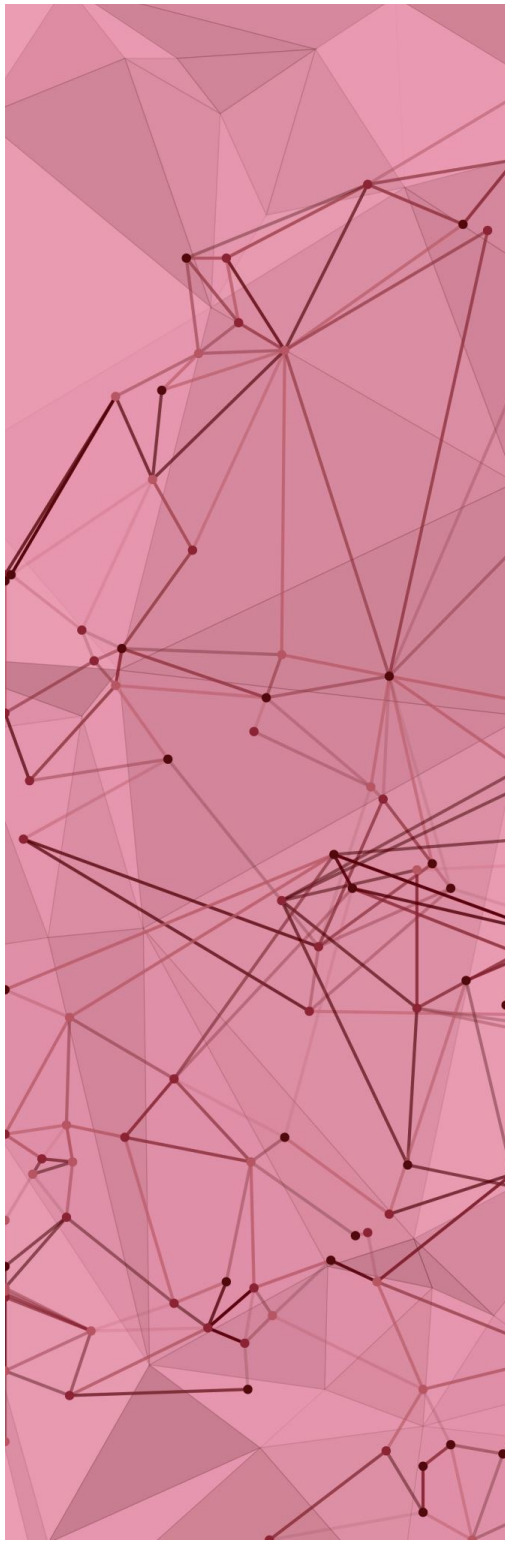
Static Scoping:

Each function has its own symbol table

```
1 int h, i;
2 void B(int w) {
3     int j, k;
4     i = 2*w;
5     w = w+1;
6     ...
7 }
8 void A (int x, int y) {
9     float i, j;
10    B(h);
11    i = 3;
12    ...
13 }
14 void main() {
15    int a, b;
16    h = 5; a = 3; b = 2;
17    A(a, b);
18    B(h);
19    ...
20 }
```

i (line 4) vs. i (line 11)

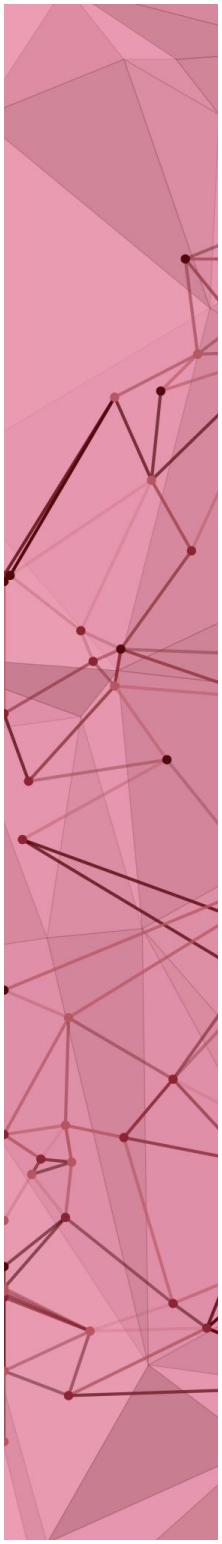
1. Outer scope: <h, 1> <i, 1> <B, 2> <A, 8> <main, 14>
2. Function B: <w, 2> <j, 3> <k, 4>
   <h, 1> <i, 1> <B, 2> <A, 8> <main, 14>
3. Function A: <x, 8> <y, 8> <i, 9> <j, 9>
   <h, 1> <i, 1> <B, 2> <A, 8> <main, 14>
4. Function main: <a, 15> <b, 15>
   <h, 1> <i, 1> <B, 2> <A, 8> <main, 14>



Review: Types

Static vs. dynamic type systems

- A language is *statically typed* if the types of all variables are fixed when they are declared at compile time.
  - Example: C
- A language is *dynamically typed* if the type of a variable can vary at run time depending on the value assigned.
  - Example: Python
- Both static and dynamic typing?
  - Example: Java
- Which one is “better”—static or dynamic?

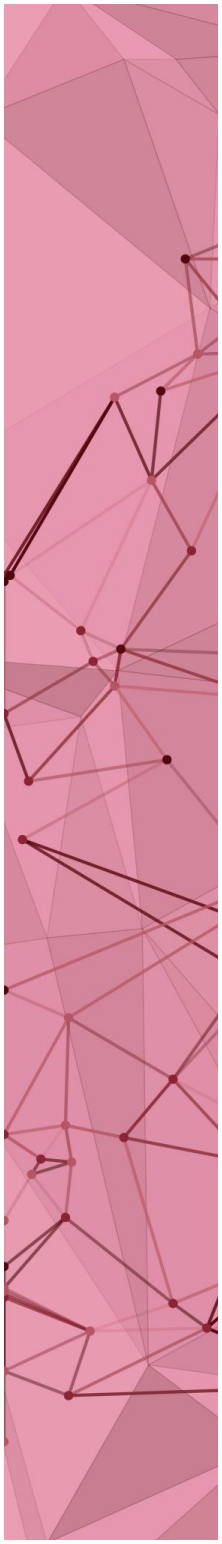


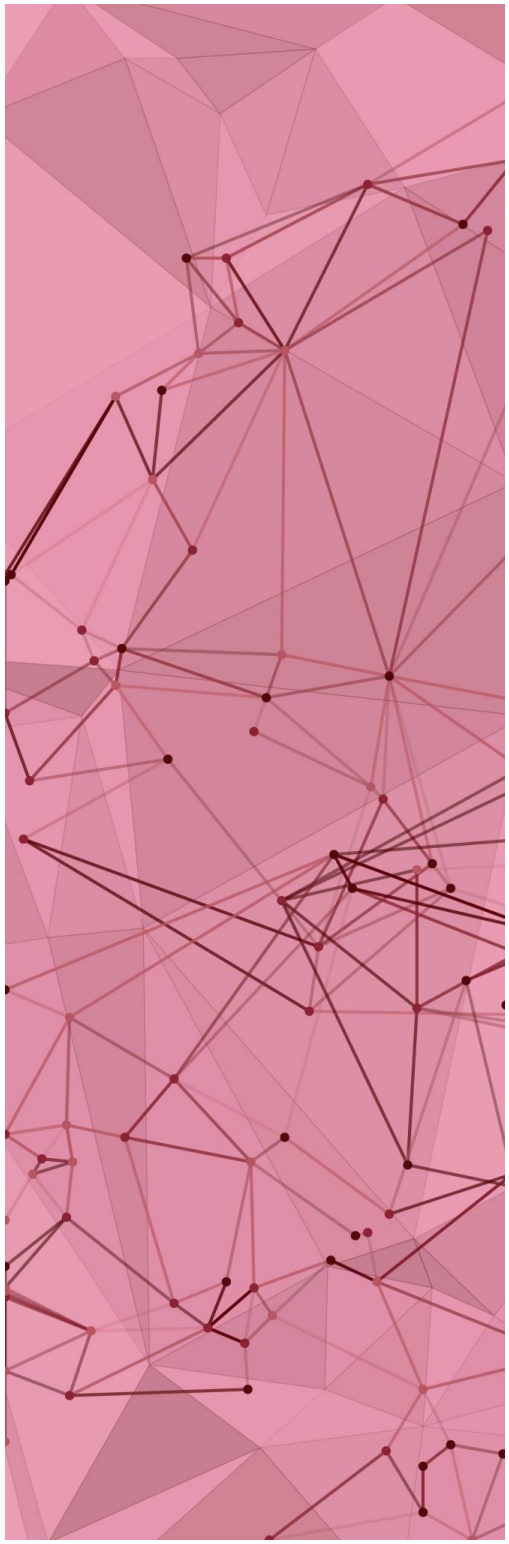
Implementation of array type

"Dope vector" + actual array

Review arrays in C vs. Java

Review arrays vs. lists in Python





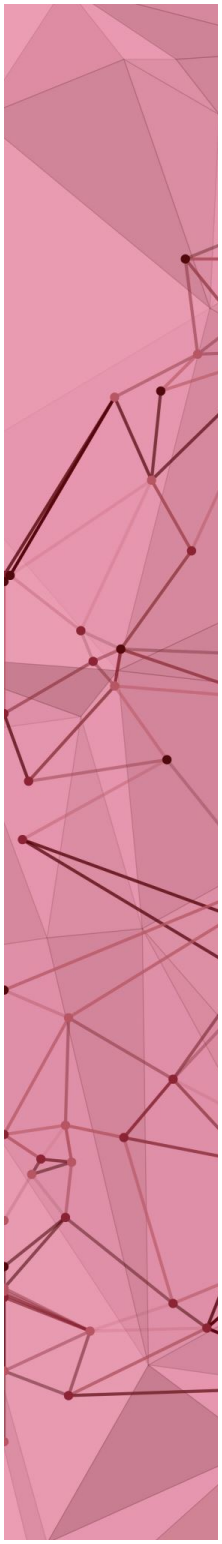
Type System: Detect type errors

Ch. 6

Type System for CLite

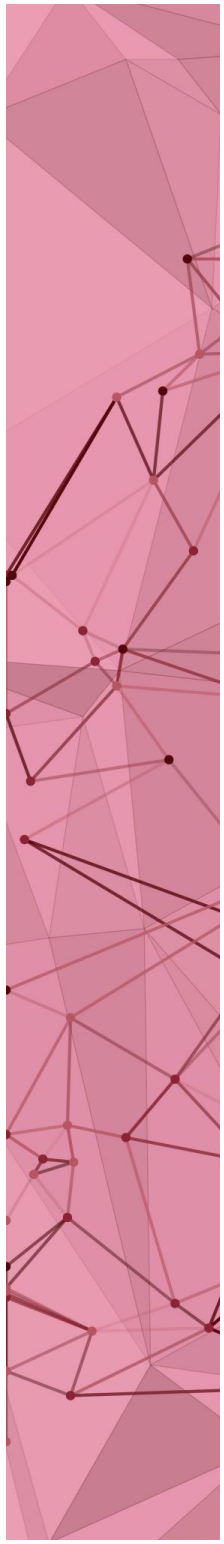
- Static typing and type checking at compile time
- Single function: main
- No global variables

- Next: Type rules



Example *Clite* Program

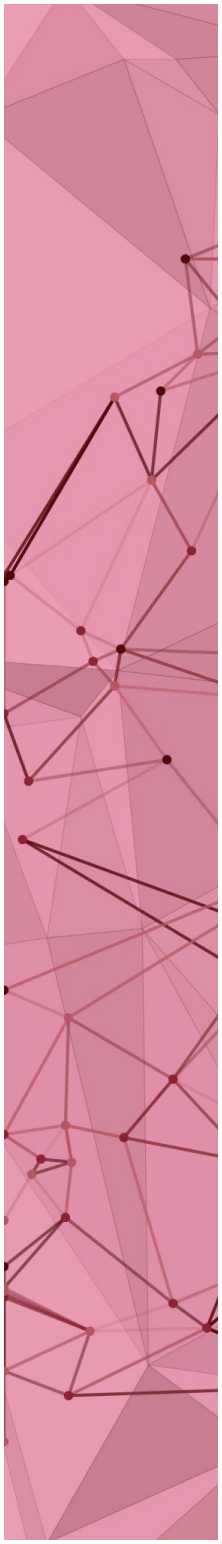
```
void main ( ) {
    int n;
    int i;
    int result;
    n = 8;
    i = 1;
    result = 1;
    while (i < n) {
            i = i + 1;
            result = result * i;
    }
    print result;
}
```



Type Rule 1

- All referenced variables must be declared.
- Type map is a set of ordered pairs

  E.g., {<n, int>, <i, int>, <result, int>}
  - Can implement as a hash table

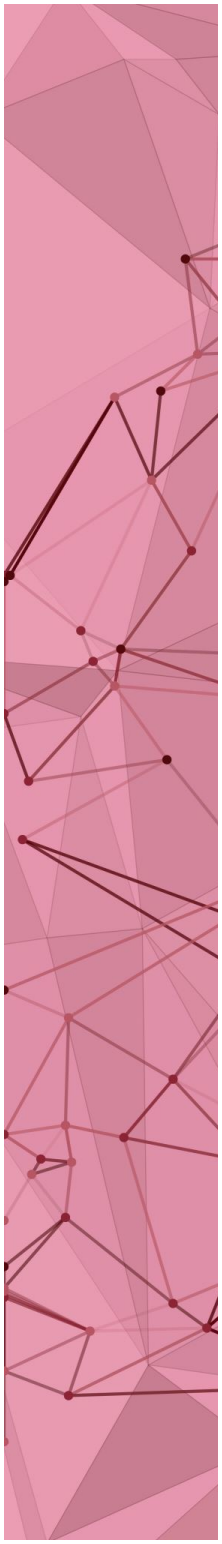


Type Rule 2

- All declared variables must have unique names.
  - How to implement?

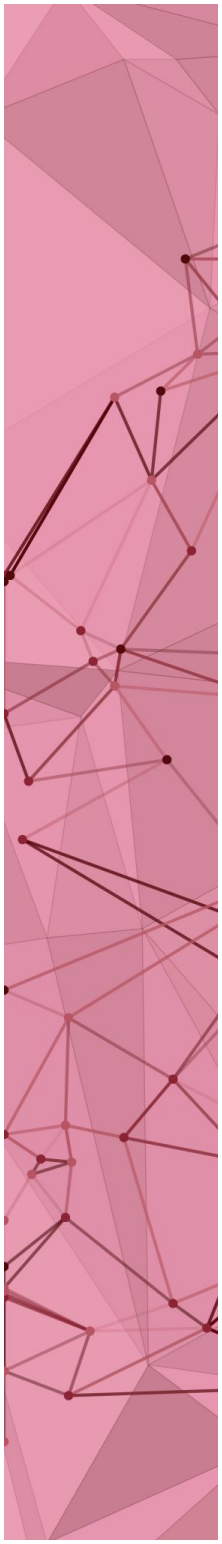
```
void main ( ) {
    int n;
    int i;
    int result;
    n = 8;
    i = 1;
    result = 1;
    while (i < n) {
            i = i + 1;
            result = result * i;
    }
    print result;
}
```

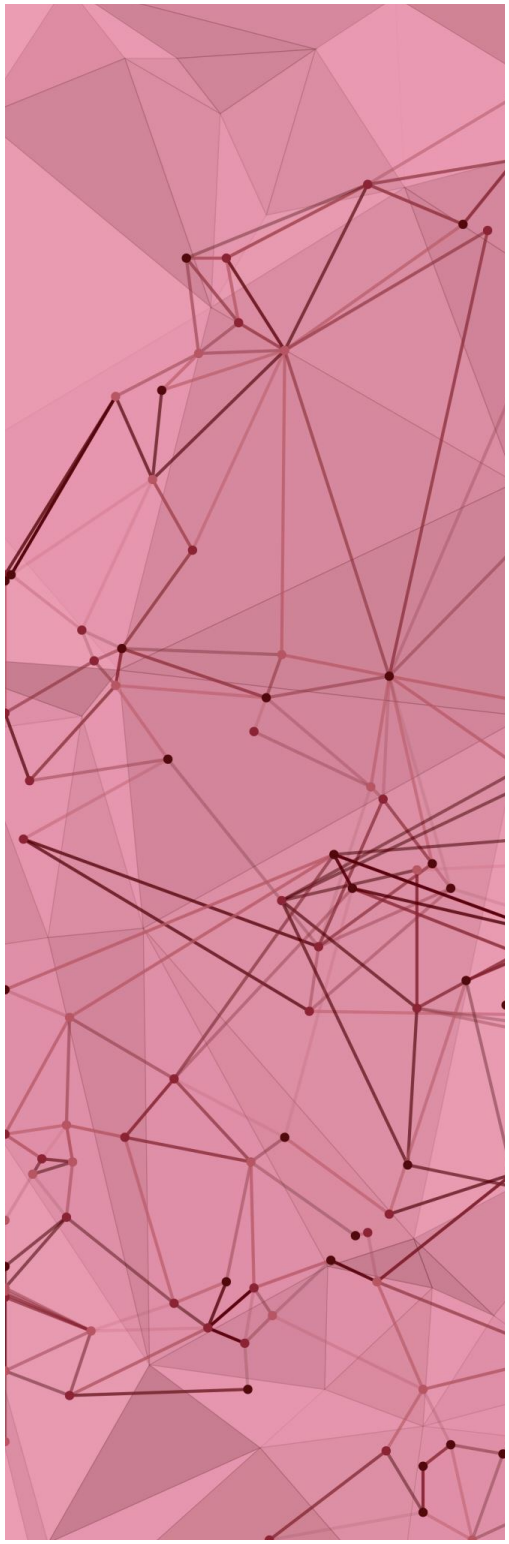
These must all be unique



Type Rule 3

- Identifier must not be a keyword
- How to implement?
- Can enforce it even before semantic analysis! (Project 1)



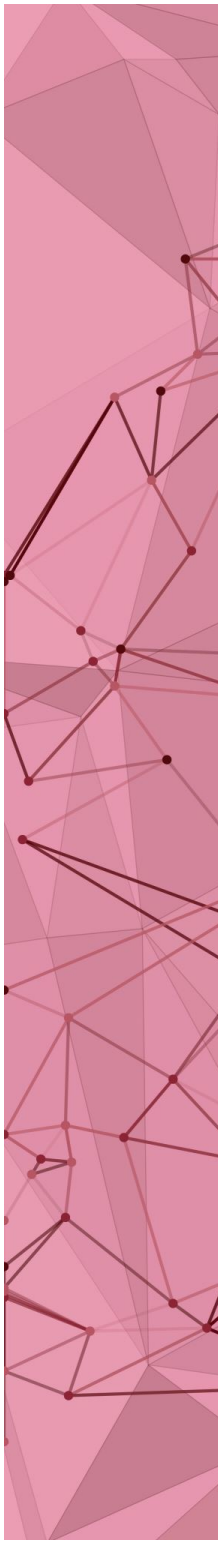


Type Checking

Type checking

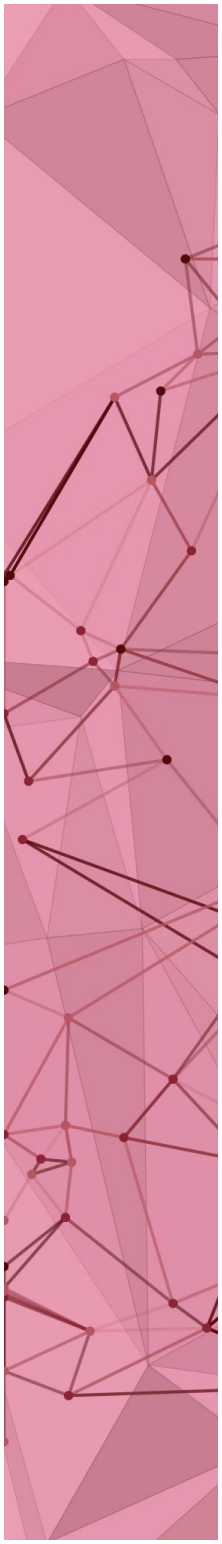
- *A program is valid if*
  - Declarations are valid and
  - Rest is valid wrt Declarations

```
void main ( ) {
    int n;
    int i;
    int result;
    n = 8;
    i = 1;
    result = 1;
    while (i < n) {
            i = i + 1;
            result = result * i;
    }
    print result;
}
```



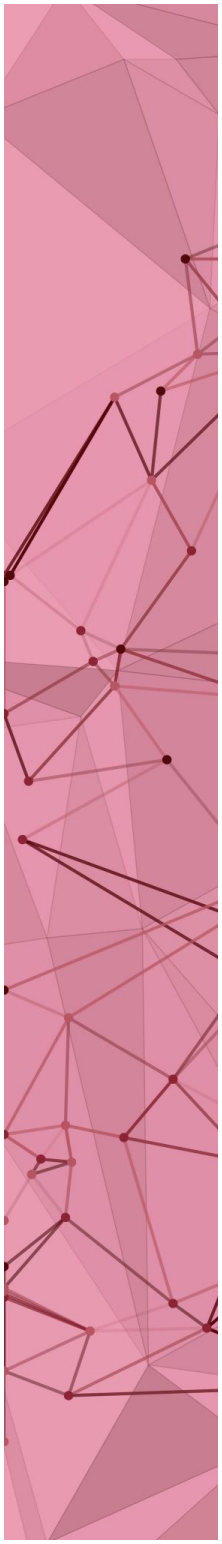
Type checking

- *Validity of a Statement:*
- Assignment statement is valid if
  - Its target *Variable* is declared
  - Its source *Expression* is valid
  - If the target *Variable* is float, then the type of the source *Expression* must be either float or int
  - Otherwise if the target *Variable* is int, then the type of the source *Expression* must be either int or char
  - Otherwise, the target *Variable* must have the same type (e.g., int, bool, etc.) as the source *Expression*



Type checking

- A conditional stmt (or if stmt) is valid if:
  - Its test *Expression* is valid and has type bool
  - Its body is valid
- A while loop is valid if:
  - Its test *Expression* is valid and has type bool
  - Its body is valid



Design question: Validity of an expression

- Define the validity and type of any expression for int, bool, char, and float types
- That is, define the compatibility between values and operations in any expression
- Cases:
  - Single term
  - More complex
    - Unary operations
    - **Binary operations:** think about all possible *kinds* of binary operations w.r.t. type
      - Arithmetic
      - Relational