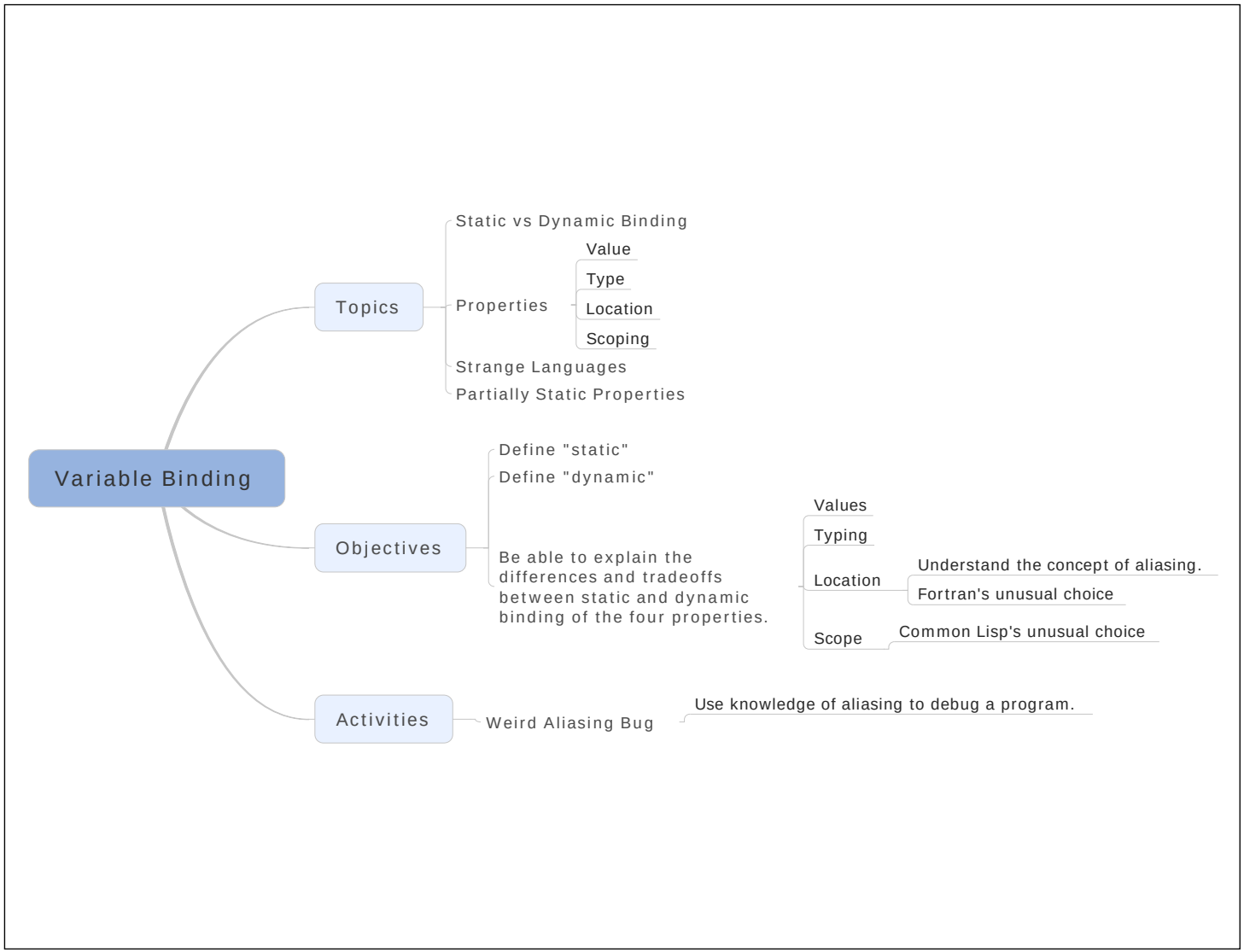
# Variables

Dr. Mattox Beckman

Illinois Institute of Technology
Department of Computer Science

# Outline


Variable Binding
Topics
Static vs Dynamic Binding
Properties
Value
Type
Location
Scoping
Strange Languages
Partially Static Properties
Objectives
Define "static"
Define "dynamic"
Be able to explain the differences and tradeoffs between static and dynamic binding of the four properties.
Values
Typing
Location
Understand the concept of aliasing.
Fortran's unusual choice
Scope
Common Lisp's unusual choice
Activities
Weird Aliasing Bug
Use knowledge of aliasing to debug a program.

# Objectives

## You should be able to...

Variables have many different attributes. These attributes can become *bound* to the variable at different times.

- Know the difference between static and dynamic binding...
  - of value
  - of types
  - of location
  - of scoping (!)
- Know the difference between implicit and explicit declaration.
- Know what aliasing is.

# What is a Variable?

## Mathematically

Variables represent a (possibly unknown) quantity or value. They usually are part of a model (or abstraction) of some concept or system.

$$f(x) = 2^{i\pi} - x$$

## Programming

Variables are implementations of mathematical variables. (Has anyone here read Plato?)

# Variable Attributes

- Q: What is the difference between the mathematical variable $i$ and a C++ variable `i`?

## Variable Attributes

- Location
- Type
- Value
- Scope

# Static vs. Dynamic Binding

## Static Binding

Attribute is bound at compile time.

- Allows the compiler to "hard code" information about the variable into the executable code
- Allows the compiler to perform optimizations based on its knowledge of the variable.

## Dynamic Binding

Attribute is bound at run time.

- A variable's attribute could change during the course of execution, or remain undetermined—very flexible.
- Information about the variable is usually stored with it.
- Sometimes we *don't know* the value of the attribute at compile time.

# Value

- The value attribute of a variable is most likely to be dynamic.
- Sometimes we want the value to be static. (Not to be confused with the `static` keyword in C.)

## Static Value

```
const int i = 2;

int foo(int j) { return i * j; }

int bar() {
  int i = 10;
  i = foo(i);
  return i;
}
```

# Static Typing

- Static Typing: the type of variables are known at compile time.
- This makes many operations very efficient.

```
int sqr(int i) {

  return i * i;
}
```

- The compiler can catch errors: improving programmer reliability.

```
string s = "hi";
bool b = true;
if s then printf("4") else printf("9");
```

# Dynamic Typing

Some languages (e.g., BASIC, Perl, most shell, TCL) use dynamic typing.

```
#!/usr/bin/perl

$i = "The answer is ";
print "$i";

$i = 42;
print "$i\n";
```

Actually, Perl types are partially dynamic. Scalars, arrays, and hashes are represented with different syntax.

# Polymorphism

- Sometimes you want to be able to have both the advantages of strong typing *and* dynamic typing all at the same time.
- Methods include *overloading*, *templates*, and *automatic polymorphism*.

## Overloading

```
int identity(int i)  { return i; }
double identity(double x) { return x; }
```

## Parameterized

```
template <class T>
T ident(T &i) { return i; }
```

## Automatic

```
# let id x = x;;
val id : 'a -> 'a = <fun>
```

# Location

- Heap allocated variables — completely dynamic
- Stack allocated variables — partially static "stack relative" allocation

```
int length() {
  int i = 10;
  String s = new String("hello");
  return i + length(s);
}
```

## Weird Language

There is one language in which *all* variables — even function arguments — are allocated statically!

# Fortran

## The Problem

- Developed during a time when 4k was a lot of memory and processor speeds were measured in kHz.
- Looking up a memory location each time a variable is used is expensive!
- The problem: how do we get scientists to use a high level language rather than machine code?

## The Solution: Hard-code variable locations

- This made Fortran almost as fast as assembly.
- Still the language of choice for numerical computation.
- Downside—you don't get recursion. (Modern Fortran fixes this.)

# Aliasing

It is possible for multiple variables to refer to the *same* location.

```
int i = 20;

void inc(int &x) {
  x = x + 1;
}
// after this i and x will be the same!
... inc(i) ...
```

Use with extreme caution!

# Bad Aliasing

Knowing about aliasing and storage is critical. *Never forget that your variables are representations only.*

Do the Aliasing Bug activity.

# Lifetime

- Variables have a certain *scope* in the program for which they are valid.
- This allows us to have multiple variables with the same name.
- Usually the scope (or *lifetime*) is determined syntactically.

```
int foo(int i) {
  int j = 10;
  return j + 10;
}

int bar(int i) {
  int j = 20;
  return foo(j) + foo(i);
}
```

# Example in C

Consider the following program:

```
int i = 2;

int foo() { return i * i; }

int bar() {
  int i = 10;
  return foo();
}
```

- What value will function `bar` return?
  - 4
  - 100

# Example in Emacs Lisp

- What value will expression `(bar)` return?
  - 4
  - 100

# Static vs. Dynamic Scoping

- Most languages use *static scoping*.
- Common LISP introduced dynamic scoping.
  - "It seemed like a good idea at the time."
  - It is considered to be a Bad Thing™ by most sentient life-forms.
- It's too easy to modify the behavior of a function.
- Correct use requires knowledge of a function's internals.

Still used by Lisp, some Scheme, and Emacs Lisp.

# Problems

- Which of the following is an advantage of dynamic typing that cannot be found with static typing?
  1. You don't have to declare types.
  2. No runtime type errors can occur.
  3. Dynamically typed code will run faster than statically typed code.
  4. None of these are advantages.
- A C++ method can be either static or dynamic. How is this accomplished?

# Answers

- Which of the following is an advantage of dynamic typing that cannot be found with static typing?
  1. You don't have to declare types.
  2. No runtime type errors can occur.
  3. Dynamically typed code will run faster than statically typed code.
  4. **Solution**: None of these are advantages.
- A C++ method can be either static or dynamic. How is this accomplished?
  Syntactically: A method is made dynamic via the `virtual` keyword.
  Implementation: the compiler uses a structure called a *vtable*.