Variables

Dr. Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Objectives

You should be able to ...

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

- Explain the difference between static and dynamic binding.
  - Of value
  - Of types
  - Of location
  - Of scoping (!)
- Give examples of implicit and explicit declaration.
- Give an example of 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?)

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

```
1  const int i = 2;
2
3  int foo(int j) { return i * j; }
4
5  int bar() {
6    int i = 10;
7    i = foo(i);
8    return i;
9  }
```

Static Typing

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

```
1 int sqr(int i) {
2
3   return i * i;
4 }
```

```
1   movi r1, val(i)
2   movi r2, val(i)
3   multi r1,r2,r3
4   pushi r3
```

- The compiler can catch errors: improving programmer reliability.

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

Dynamic Typing

Some languages (e.g., BASIC, PERL most shell languages, 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

- We can have both the advantages of strong typing *and* dynamic typing at the same time!

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

```
1 int length() {
2   int i = 10;
3   String s = new String("hello");
4   return i + length(s);
5 }
```

Weird Language

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

FORTRAN

The Problem

- First released on the IBM 704 in 1957. It had core memory (equivalent to 18,432 *bytes*) and a 12k FLOP processor.
- Can we use a high level language and translate it to machine code?

The Solution: Hard-Code Variable Locations

- This made FORTRAN almost as fast as assembly.
- It is 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.

```
1  int i = 20;
2
3  void inc(int &x) {
4    x = x + 1;
5  }
6  // after this i and x will be the same!
7  ... 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.

```
1  int foo(int i) {
2    int j = 10;
3    return j + 10;
4  }
5
6  int bar(int i) {
7    int j = 20;
8    return foo(j) + foo(i);
9  }
```

Example in C

Consider the following program:

```
1  int i = 2;
2
3  int foo() { return i * i; }
4
5  int bar() {
6    int i = 10;
7    return foo();
8  }
```

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

Example in Emacs Lisp

```
1  (setq i 2)          ;; global variable i = 2
2
3  (defun foo ()
4    (* i i))
5
6  (defun bar ()
7    (let ((i 10))     ;; local variable i = 10
8      (foo)))         ;; call function foo
```

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

Static vs. Dynamic Scoping

- Most languages use *static scoping*.
- The first LISP implementations used *dynamic scoping*.
  - Today it is considered to be a Bad Thing™ by most sentient life-forms.
  - As always, some disagree ...
- It's too easy to modify the behavior of a function.
- Correct use requires knowledge of a function's internals.

Still used by Emacs LISP!