Variables and their 6 Attributes

1. **Name**

   = identifier

   length, legal chars, case-sensitivity, special words

   can be one-one, one-many, or one-none mapping to memory

2. **Address**

   location in memory

   may vary dynamically

3. **Type**

   range of values + legal operations

4. **Value**

   contents of the address

   - l-value (address)
   - r-value (value)

5. **Scope**

   Range of statements over which the variable is visible.

   Static/dynamic

6. **Lifetime**

   Time during which the variable is bound to a storage location.

Binding

- How and when are attributes bound to variables?
  - → *Static*

    occurs before runtime

    constant through program execution
  - → *Dynamic*

    occurs or can change during runtime
- In many ways, the various binding times determin the flavor of a language.
- As binding time gets earlier:
  - → efficiency goes up
  - → safety goes up
  - → flexibility goes down

Type Binding

When is type bound to variable?

How is binding specified?

- ***Static typing*** **(before runtime)**
  - → **explicit declaration**

    var x: integer
  - → **implicit declaration**

    e.g., Fortran:

    i := 5 OK

    i := 1.3 not OK

- **Advantages:**
  - → **cheaper**
  - → **safer**
- **Disadvantage:**
  - → **less flexible**

Type Binding

- *Dynamic Binding* (after compile time)
  - → Variable gets type of value assigned to it.

```
x := 5
...
x := "foo"
```

- Advantage:
  - → flexibility
- Disadvantages:
  - → runtime overhead
  - → poor error detection
- More about types later. . .

Scope

- *Static (lexical) scope*
  - → Scope of a variable is determined by the textual layout of the program.
  - → In block structured languages, scope of a variable is the unit in which it is defined, plus all units nested inside that unit, excluding those in which the variable is redeclared.
  - → To find the declaration of a variable, look through the statically enclosing units until a declaration is found.
- *Dynamic Scope*
  - → Scope of a variable depends on program execution, and therefore changes dynamically.
  - → To find declaration, look up through the call chain.

Example (evaluate both ways)

```
program foo;
var x: integer;

   procedure f;
   begin
       print(x);
   end;

   procedure g;
   var x: integer;
   begin
       x := 2;
       f;
   end

begin
   x := 1;
   g;
end.
```

Lifetime (= extent)

- The lifetime of a variable is the interval of time during which it is bound to a specific memory location.

- Static variables
  - → bound to memory cells before execution
  - → retain same binding throughout execution
  - → efficient, inflexible
  - → allow history-sensitivity
  - → used in FORTRAN
  - → do not support recursion

- Semidynamic variables
  - → storage allocated when unit is called
  - → storage deallocated when unit returns
  - → allows recursion

Lifetime (continued)

- Explicit Dynamic Variables
  - → storage allocated and deallocated by programmer
  - → *new*, *dispose* in Pascal
  - → flexible and efficient, but dangerous
- Implicit Dynamic Variables
  - → automatically bound to storage as needed
  - → storage automatically reclaimed when no longer needed
  - → flexible, safe, less efficient
  - → lists in Lisp, Prolog

Scope ≠ Lifetime

- **lifetime > scope: storage that can't be accessed through that variable.**

```
var p: ^integer;
begin
  ...
  new(p)
  ...
end
```

here, storage is still allocated but p is not defined,
Lifetime > Scope

- **scope > lifetime: variable without storage.**

```
var p: ^integer;
begin
  ...
  new (p)
  ...
  dispose(p)
```

here, p is defined but has no value,
Scope > Lifetime

```
end
```

Scope ≠ Lifetime (continued)

- **Also, scope has "holes" during execution, but lifetime does not.**

```
procedure f;
begin
   ...
end

procedure g;
var x: integer;
begin
   ...
   f
   ...
end
```

out of x's scope during execution of f (assuming static scope), but x's lifetime persists.

Variable Initialization

- static or dynamic
- once for static variables, at each allocation for dynamic variables
- many possible methods:

```
sum: integer := 0;          (Ada)
int first := 10             (Algol 68 initialization)
int first = 10              (Algol 68 constant declaration)
```

- unavailable in Pascal
- default initializations

Aliasing

- **Two variables are aliases if they share the same storage.**

```
var x,y:  ^integer;
begin
   new (x);
   x^ := 5;
   y := x;
   x^ := 10;
   writeln (y^);
end
```

```
var x,y:  integer;
begin
   x := 5;
   y := x;
   x := 10;
   writeln (y);
end
```

- **Also results from *var* parameters:**

```
procedure p (var x,y: integer);
....
p (a,b);
p (x1,x1);
```