

Programming Languages and Compilers (CS 421)

Elsa L Gunter
2112 SC, UIUC

<http://www.cs.uiuc.edu/class/fa06/cs421/>

Based in part on slides by Mattox Beckman, as updated by Vikram Adve and Gul Agha

Data Objects

- Structured storage location for data
- Specification
 - Attributes
 - Values
 - Operations
- Implementation
 - Storage location and layout
 - Access
 - Operations

Elsa L. Gunter

Data Object Attributes

- Location:
 - Virtual machine may allow programmer to view as fixed, but may be moved on actual hardware by storage management of virtual machine

Elsa L. Gunter

Variables and Data Objects

- Attributes of a variable:
 - Name
 - Scope
 - Lifetime
 - Data Object (abstract memory cell)
- Each variable is associated with a unique data object
- Data object may be associated with more than one variable (aliasing)

Elsa L. Gunter

Data Object Attributes

- Data type:
 - Set of values that may be stored in data object, and set of basic operation which may be performed on it

Elsa L. Gunter

Data Object Attributes

- Value
- Name(s)
- Components:
 - Binding of one or more data objects as part of the given one (eg. pairs)
- Lifetime
 - Allocation to garbage collection

Elsa L. Gunter

Data Object Versus Data Value

- Data object:
 - A location in memory to hold data accessed by the variable name
- Data value: 110110
 - Binary representation of the number 54
- Data value stored in data object:

00110110

Elsa L. Gunter

Binding of Data Objects

- A variable is a binding of a name to a data object:
 - Data object is storage for values
 - Contents of the object may change
- Different binding means different variable

Elsa L. Gunter

Bindings of Program Elements

- A *binding* of a program element is the selection of a particular value for it from a set of possible values
- The *binding time* is when that selection is made
- Binding time can depend on whether we use a compiler or an interpreter

Elsa L. Gunter

Binding Times

- Language definition time: language syntax and semantics
- Language implementation time: interpreter versus compiler, aspects left flexible in definition, set of available libraries

Elsa L. Gunter

Binding Times

- Compile time: some initial data layout, internal data structures
- Link time (load time): binding of values to identifiers across program modules
- Run time (execution time): actual values assigned to non-constant identifiers

Elsa L. Gunter

To class

- In expression **int x = y + 5** what bindings can you identify, and when are they most likely made?

Elsa L. Gunter

Binding of Data Objects and Variables

- Attributes of data objects and variables have different binding times
- If a binding is made before run time and remains fixed through execution, it is called *static*
- If the binding first occurs or can change during execution, it is called *dynamic*

Elsa L. Gunter

Constants and Variables

- A named data object which may have its data value changed is called a *variable*
 - Usually dynamically allocated
- A named data object with a fixed data value is called a *constant*
 - Usually statically allocated
- Some constants have names that we think of as their values (eg 2 or “hi”). These are called *literals*

Elsa L. Gunter

Declarations

- User defined variable and associated data object, created at compile time
- Declaration tells for a variable its:
 - Name
 - Type
 - Scope - which may also indicate its lifetime
 - Can be implicit as part of an initial assignment

Elsa L. Gunter

Uses for Declarations

- Determine storage representations
- Help with storage management
- Resolve overloaded constants
- Help type checking

Elsa L. Gunter

Static vs Dynamic Type Checking

- If types of all variables and signatures of all procedures are known at compile time, type checking can be done *statically*
 - Runtime code more efficient since no runtime type checks
- If variable types only known at runtime, must use *dynamic* type checking
 - Variables may be used at multiple types

Elsa L. Gunter

Assignment

- Data object (and variable) has two values associated with it:
 - Location and value
- In assignment $X := Y$ we are saying the location of X gets the value of Y
- Location called *l-value* of object (value used when on left of assignment)
- Value called *r-value* (value used on right of assignment)

Elsa L. Gunter

Initialization

- When a variable is declared, a data object is created for it
- Before it is initialized it contains random data (bit strings)

Elsa L. Gunter

Initialization

- If used before initialized, program will probably not recognize the random string is junk and will use it
- Hard to find such bugs
- Good language design forces initialization as part of declaration

Elsa L. Gunter

Scope of Variable

- Range of program that can reference that variable (ie access the corresponding data object by the variable's name)
- Variable is *local* to program or block if it is declared there
- Variable is *nonlocal* to program unit if it is visible there but not declared there

Elsa L. Gunter

Lifetime

- Lifetime of a variable is the duration of the program execution during which the variable - binding of the name to data object - exists
 - Should contain the variable's scope
 - May exceed the scope through hiding
- Lifetime of data object is the period from its allocation to its garbage collection
 - Needs to contain lifetime of all variables associated with it
 - Failure of this leads to dangling pointers

Elsa L. Gunter

Static Scoping

- Scope computed at compile time, based on program text
- To determine the variable of a name used must find statement declaring variable

Elsa L. Gunter

Static Scoping

- Subprograms and blocks generate hierarchy of scopes
 - Subprogram or block that declares current subprogram or contains current block is its *static parent*

Elsa L. Gunter

Static Scoping

- General procedure to find declaration:
 - First see if variable is local; if yes, done
 - If nonlocal to current subprogram or block recursively search static parent until declaration is found
 - If no declaration is found this way, undeclared variable error detected

Elsa L. Gunter

Example

```

program main;      begin { main }
  var x : integer;  ... x ...
  procedure sub1;  end; { main }
  var x : integer;
  begin { sub1 }
    ... x ...
  end; { sub1 }
  
```

Elsa L. Gunter

Dynamic Scope

- Now generally thought to have been a mistake
- Main example of use: original versions of LISP
 - Common LISP uses static scope
 - Perl allows variables to be declared to have dynamic scope

Elsa L. Gunter

Dynamic Scope

- Determined by the calling sequence of program units, not static layout
- Name bound to corresponding variable most recently declared among still active subprograms and blocks

Elsa L. Gunter

Example

```

program main;      procedure sub2;
  var x : integer;  var x : integer;
  procedure sub1;  begin { sub2 }
  begin { sub1 }    ... call sub1 ...
  ... x ...        end; { sub2 }
  end; { sub1 }    begin { main }
                  ... call sub2 ...
                  ... call sub1 ...
                  end; { main }
  
```

Elsa L. Gunter

Example: Static Scope

```

program main;      procedure sub2;
  var x : integer;  var x : integer;
  procedure sub1;  begin { sub2 }
  begin { sub1 }    ... call sub1 ...
  ... x ...        end; { sub2 }
  end; { sub1 }    begin { main }
                  ... call sub2 ...
                  ... call sub1 ...
                  end; { main }
  
```

Elsa L. Gunter

Example: Dynamic Scope

```

program main;
var x : integer;
procedure sub1;
begin { sub1 }
  ... x ...
end; { sub1 }

procedure sub2;
var x : integer;
begin { sub2 }
  ... call sub1 ...
end; { sub2 }

begin { main }
  ... call sub2 ...
  ... call sub1 ...
end; { main }

```

Elsa L. Gunter

Example: Dynamic Scope

```

program main;
var x : integer;
procedure sub1;
begin { sub1 }
  ... x ...
end; { sub1 }

procedure sub2;
var x : integer;
begin { sub2 }
  ... call sub1 ...
end; { sub2 }

begin { main }
  ... call sub2 ...
  ... call sub1 ...
end; { main }

```

Elsa L. Gunter

Referencing Environment

- The *referencing environment* of a program point is the set of all variables (names bound to data objects) visible at that program point
- In a statically scoped language referencing environment is all local variables (declared so far) together with all declared variables of all static ancestors minus those that have been hidden

Elsa L. Gunter

Example (Static Scope)

```

program main;
var x, y : integer;
procedure sub1;
var z : integer
begin { sub1 }
  ... point1 ...
end; { sub1 }

procedure sub2;
var w, x : integer;
begin { sub2 }
  ... point2 ...
end; { sub2 }

...point3 ...
end; { main }

```

Elsa L. Gunter

Example

- Referencing Environments:
- Point1: main.x main.y sub1.z
- Point2: main.y sub2.x sub2.w
- Point3: main.x main.y

Elsa L. Gunter