

Name, Binding and Scope

Dr. Nguyen Hua Phung

phung@cse.hcmut.edu.vn

HCMC University of Technology, Viet Nam

09, 2013

- Name - character string used to represent something else.
 - identifiers,
 - operators (+, &, *).
- Use **symbol** instead of **address** to refer an entity.
- Abstraction

Definition

- Binding - the operation of associating two things.
- Binding time - the moment when the binding is performed.

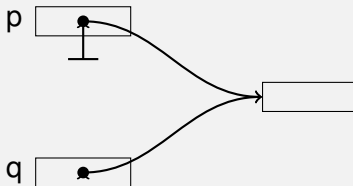
Some issues

- Early binding vs. Late binding
- Static binding vs. Dynamic binding
- Polymorphism - A name is bound to more than one entity.
- Alias - Many names are bound to one entity.

- Language design time
- Language implementation time
- Programming time
- Compilation time
- Linking time
- Load time
- Runtime

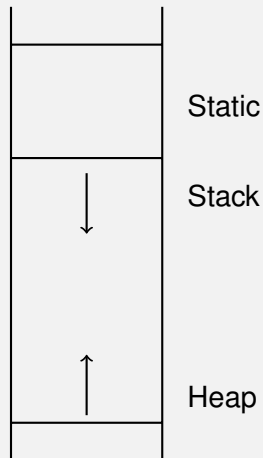
- *Object* - any entity in the program.
- *Object lifetime* - the period between the object creation and destruction.
- *Binding lifetime*
- *Dangling reference*

```
p = new int;  
q = p;  
delete p;  
*q;
```



- *Leak memory - Garbage*
p = new int;
p = null;

- Static
- Stack Dynamic
- Explicit Heap Dynamic
- Implicit Heap Dynamic



Definition

A block is a textual region, which can contain declarations to that region

Example,

```
procedure foo()  
var x:integer;  
begin  
    x := 1;  
end;  
  
    {  
        int x;  
        x = 1;  
    }
```

Definition

Scope of a binding is the textual region of the program in which the binding is effective.

Static vs. Dynamic

- Static scope, or lexical scope, is determined during compilation
 - Current binding - in the block most closely surround
 - Global scope
 - Local static scope
- Dynamic scope is determined at runtime.
 - Current binding - the most recently execution but not destroyed

- A reference to an identifier is always bound to **its most local declaration**
- A declaration is **invisible** outside the block in which it appears
- Declarations in enclosing blocks are **visible** in **inner blocks**, unless they have been **re-declared**
- Blocks may be named and its name declaration is considered as a **local declaration of outer** block.

Example on Static scope

```
var A, B, C: real; //1
procedure Sub1 (A: real); //2
    var D: real;
    procedure Sub2 (C: real);
        var D: real;
        begin
            ... C:= C+B; ...
        end;
    begin
        ... Sub2(B); ...
    end;
begin
    ... Sub1(A); ...
end.
```

Variable	Scope
A:real //1	Main
B:real //1	Main, Sub1, Sub2
C:real //1	Main, Sub1
A:real //2	Sub1, Sub2
...	

Example on Dynamic Scope

```
procedure Big is
  X : Real;
  procedure Sub1 is
    X : Integer;
    begin -- of Sub1
      ...
    end; -- of Sub1
  procedure Sub2 is
    begin -- of Sub2
      ...X...
    end; -- of Sub2
begin -- of Big
  ...
end; -- of Big
```

X in Sub2 ?

Calling chain:

Big \rightarrow Sub1 \rightarrow Sub2

X \Rightarrow X:Integer in Sub1

Calling chain:

Big \rightarrow Sub2

X \Rightarrow X:Real in Big

- The **referencing environment** of a **statement** is the collection of all names that are visible to the statement
- In a **static-scoped** language, it is the local names plus all of the visible names in all of the enclosing scopes
- In a **dynamic-scoped** language, the referencing environment is the local bindings plus all visible bindings in all active subprograms

Example on Static-scoped Language

```
var A, B, C: real; //1
procedure Sub1 (A: real); //2
    var D: real;
    procedure Sub2 (C: real);
        var D: real;
        begin
            ... C:= C+B; ...
        end;
    begin
        ... Sub2(B); ...
    end;
begin
    ... Sub1(A); ...
end.
```

Function	Referencing Environment
Main	A, B, C, Sub1
Sub1	A, B, C, D, Sub1, Sub2
Sub2	A, B, C, D, Sub1, Sub2

Example on Dynamic-scoped Language

	main	→ sub2	→ sub2	→ sub1
void sub1() {	c	b	b	a
int a, b;	d	c	c	b
...				
} /* end of sub1 */				
void sub2() {				
int b, c;				
...				
sub1;				
} /* end of sub2 */				
void main() {				
int c, d;				
...				
sub2();				
} /* end of main */				

Frame	Referencing Environment
main	c → o1, d → o2
sub2	b → o3, c → o4, d → o2
sub2	b → o5, c → o6, d → o2
sub1	a → o7, b → o8, c → o6, d → o2

- Name
- Binding
- Scope
- Referencing Environment



, Maurizio Gabbrielli and Simone Martini, Programming Languages: Principles and Paradigms, Chapter 4, Springer, 2010.