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.
Binding Time

- Language design time
- Language implementation time
- Programming time
- Compilation time
- Linking time
- Load time
- Runtime
**Object Lifetime**

- **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;
    ```
Object Allocation

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

---

Dr. Nguyen Hua Phung  phung@cse.hcmut.edu.vn
CuuDuongThanCong.com  https://fb.com/tailieudientucntt
Definition

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

Example,

```pascal
procedure foo ()
var x: integer;
begin
 x := 1;
end;
```

```c
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 surrounding
  - Global scope
  - Local static scope

- **Dynamic scope** is determined at runtime:
  - Current binding - the most recently executed 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

```plaintext
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.
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>A:real</td>
<td>Main</td>
</tr>
<tr>
<td>B:real</td>
<td>Main, Sub1, Sub2</td>
</tr>
<tr>
<td>C:real</td>
<td>Main, Sub1</td>
</tr>
<tr>
<td>A:real</td>
<td>Sub1, Sub2</td>
</tr>
</tbody>
</table>

Dr. Nguyen Hua Phung  
phung@cse.hcmut.edu.vn

https://fb.com/tailieudientucntt
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 → Sub1 → Sub2
X ⇒ X:Integer in Sub1

Calling chain:
Big → Sub2
X ⇒ 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.
```

<table>
<thead>
<tr>
<th>Function</th>
<th>Referencing Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td>A, B, C, Sub1</td>
</tr>
<tr>
<td>Sub1</td>
<td>A, B, C, D, Sub1, Sub2</td>
</tr>
<tr>
<td>Sub2</td>
<td>A, B, C, D, Sub1, Sub2</td>
</tr>
</tbody>
</table>
Example on Dynamic-scoped Language

```c
void sub1() {
    int a, 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

<table>
<thead>
<tr>
<th>Frame</th>
<th>Referencing Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>c → o1, d → o2</td>
</tr>
<tr>
<td>sub2</td>
<td>b → o3, c → o4, d → o2</td>
</tr>
<tr>
<td>sub2</td>
<td>b → o5, c → o6, d → o2</td>
</tr>
<tr>
<td>sub1</td>
<td>a → o7, b → o8, c → o6, d → o2</td>
</tr>
</tbody>
</table>
Name
Binding
Scope
Referencing Environment
References