Scoping

Presented by
Deepti Verma
Computer Science Department
New York University
dv697@cs.nyu.edu

Introduction

- What is Scoping?
- Static Scoping
- Dynamic Scoping
- Static Scoping vs Dynamic Scoping Examples

What is Scoping?

- Rules that determine the visibility of names in a program.
- The name is a mnemonic character string used to represent some entity example identifiers.
- Scope: The portion of the program where a particular name is visible.

Example:

```
class Example1{
private int globalVariable;

    public void foo(int arg1){
    int localVariable;
    ..
    }
    }
```

Static Scoping

- In a language with static scoping, the bindings between names and objects can be determined at compile time by examining the text of the program, without consideration of the flow of control at run time.
- All modern languages use Static Scoping.

Static Scoping

- The determination of scopes for entities can be made by the compiler (early binding)
- All bindings for identifiers can be resolved by examining the program
- Typically, we choose the most recent, active binding made at compile time

Static Scoping Example:

```
program A;
var I:integer; K:char;
  procedure B;
  var K:real; L:integer;
    procedure C;
    var M:real;
    begin
       (*scope A+B+C*)
    end;
    procedure D;
    var N:real;
    begin
       (*scope A+B+D*)
    end;
  (*scope A+B*)
  end;
  (*scope A*)
end;
```

Dynamic Scoping

- In a language with dynamic scoping, the bindings between names and objects depend on the flow of control at run time, and in particular on the order in which subroutines are called.
- Very few languages implement dynamic scoping Lisp, postscript.
- Because the flow of control cannot in general be predicted in advance, the bindings between names and objects in a language with dynamic scoping cannot in general be determined by a compiler.
- The dynamic resolution can only be determined at run time (late binding)

Dynamic Scoping

Advantages:

- Easier to implement in interpreters.
- Programming convenience

Disadvantages:

- Readability: Hard to analyze routines with non-local variables that may be redefined by any caller of the routine
- Can lead to a lot of errors.
- Binding errors may not be detected until run time.

Static Scoping vs Dynamic Scoping Example 1

```
int x;
int main() {
   x = 14;
   f();
   g();
}

void f() {
   int x = 13;
   h();
}

void g() {
   int x = 12;
   h();
}
void h() {
printf("%d\n",x);
}
```

Solution:
Static Scoping:
14
14

Dynamic Scoping:
13
12

Static Scoping vs Dynamic Scoping Example 2

```
const int b = 5;
int foo(){
    int a = b + 5;
    return a;
}
 int bar(){
    int b = 2;
    return foo();
}
int main(){
    foo();
    bar();
    return 0;
}
```

Solution:
Static Scoping:
foo and bar return 10
and 10 respectively

Dynamic Scoping:
foo returns 10
bar returns 7

Static Scoping vs Dynamic Scoping Example 3

```
x=1;
function g () {
echo $x ;
x=2 ;
}
function f () {
local x=3;
g;
}
f;
echo $x;
```

Solution:
Static Scoping:
1
2
Dynamic Scoping:
3
1

Static Scoping vs Dynamic Scoping Example 4

```
n:integer
procedure first
  n:=1

procedure second
  n:integer
  first()

n:=2
if read_integer() > 0
      second();
else
      first();
write_integer(n)
```

Solution:
Static Scoping:
1
Dynamic Scoping:
It depends on the value given for read_integer(). If the input is positive then it is 2 otherwise 1

Static Scoping vs Dynamic Scoping Example 4

Explanation:

- Static scoping requires that the reference resolve to the closest lexically enclosed declaration. Procedure first changes n to 1 and write_integer prints the value.
- Dynamic scoping require that we choose the most recent binding for n at run time.

We create a new binding for n when we enter the main program. Another binding for n is created in the procedure second. When we execute the assignment statement “n:=1”, the n to which we are referring will depend on whether we entered first through second or directly from the main program.

If we entered procedure first through procedure second, value 1 will be assigned to second’s local n.

If we entered procedure first through main program, we will assign the value 1 to the global n.

In either case the line write_integer(n) will refer to the global variable n, since second’s n will be destroyed along with its binding, when the control returns to the main program.