Names, Scopes, and Bindings exercises

Martin Kozeny
CSCI 4501: Programming Language Structure
Spring 2011 University of New Orleans

March 21, 2011

3 Names, Scopes, and Bindings exercises

3.1

Question Indicate the binding time (e.g., when the language is designed, when the program is linked, when the program begins execution, etc.) for each of the following decisions in your favorite programming language and implementation. Explain any answers you think are open to interpretation

- The number of built-in functions (math, type queries, etc.)
- The variable declaration that corresponds to a particular variable reference (use)
- The maximum length allowed for a constant (literal) character string
- The referencing environment for a subroutine that is passed as a parameter
- The address of a particular library routine
- The total amount of space occupied by program code and data

Answer

3.3

Question Give two examples in which it might make sense to delay the binding of an implementation decision, even though sufficient information exists to bind it early.

Answer Local variables in Fortran 77 may be allocated on the stack, rather than in static memory, in order to minimize memory footprint and to facilitate interoperability with standard tools (e.g., debuggers).

Just-in-time compilation allows a system to minimize code size, obtain the most recent implementations of standard abstractions, and avoid the overhead of compiling functions that aren't used.

3.4

Question Give three concrete examples drawn from programming languages with which you are familiar in which a variable is live but not in scope.

Answer

1. C: global variable `x` is still live but not in the scope if a local variable `x` is declared in the scope and here appears so called hole in the scope.
2. C: when we declared static variable inside function, it is not in scope outside function, but is still alive.
3. Java: private member variables of an object of Java class are live but not in scope when execution is not inside a member function of Java.

3.5

Question Consider the following pseudocode, assuming nested subroutines and static scope.

(a) What does this program print?

(b) Show the frames on the stack when *A* has just been called. For each frame, show the static and dynamic links.

(c) Explain how *A* finds *g*.

Answer Firstly in C after invocation of `main()` are declared and initialized variables `a` and `b`. Then procedure `middle()` is declared and immediately called. Inside procedure `middle()` there is declared variable `b` again, procedure `inner()` and variable `a` also again in that order. After that procedure `inner()` is invoked and inside it is `a` and `b` printed. We know that C uses static scoping and that declaration must be before use. That means that name `b` in procedure `inner()` is bound to declaration in procedure `middle()` and name `a` is bound to declaration in `main()`, because `a` in closest enclosing block (procedure `middle()`) is declared after declaration od `inner()`. So firstly 1, 1 are printed.

After that in procedure `middle()` are bound names of `a` and `b` to declaration in this procedure so 3, 1 is printed.

When printing `a` and `b` in `main()`, also here is declaration in this procedure is used so 1, 2 is printed

In C# we can declare nested functions only using keyword `delegate`, but in this nested function is not possible to declare variable with same name as in enclosing function.

3.14

Question Consider the following pseudocode.

```
x : integer -- global

procedure set_x(n : integer)
   x := n

procedure print_x
   write integer(x)

procedure first
   set_x(1)
   print_x

procedure second
   x : integer
   set_x(2)
   print_x

set_x(0)
first()
print_x
second()
print_x
```

What does this program print if the language uses static scoping? What does it print with dynamic scoping? Why?

Answer In static scoping is variable bound to the closest enclosing block, so in procedure `print_x` is variable `x` always bound to global declaration, so the output will be:

1

1
2
2

In dynamic scoping choose the most recent active binding for `x` at run-time. When calling procedure `second()` and then call inside `print_x`, variable `x` is bound to declaration in procedure `second()` and then printed as 2, but when calling `print_x` outside scope of `second()`, `x` is bound to global declaration and `print_x` prints 1. So the output will be:

1
1
2
1