Jim Crowell
CS354 – Programming Languages Homework 3
Kennington; W&F:1030-1145

P167 3.1
Indicate the binding time (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.

A) The number of built-in functions (math, type queries, etc.)

B) The variable declaration that corresponds to a particular variable reference (use)

C) The maximum length allowed for a constant (literal) character string

D) The referencing environment for a subroutine that is passed as a parameter

E) The address of a particular library routine

F) The total amount of space occupied by program code and data

C++

A) If I understand this, the basic functions built in to a language, and the queries it allows you to make on objects are all things that are decided at language design time. I think the "built-in" is kind of a giveaway.

B) A specific variable name that one uses to access/reference the variable should be bound at compilation time.

C) Max length could be said to be set at language design time, but I think this is where the differentiation of "language implementation time" the book mentions comes up. I think the implementor may have some flexibility on simple maximums like that.

D) Parameter subroutine references would be set at program writing time, due to C++'s static scope locking the order of ops into the same order that functions and the like are declared or nested.

E) This sounds like inter-module function calling/referencing, which means it must be done at link time, where the linker sets up the layout of modules in reference to each other.

F) Space occupied by program code and data is decided at compile time, because it isn't until the program is compiled and run-able that anyone can know what its "finished state" looks like.

P167 3.4
Give three concrete examples drawn from a programming language with which you are familiar in which a variable is live but not in scope.

Java

Example 1 - Inter-Method Accessibility

```
Class ExampleClass {
   public static void main(String args[]) {
      int i = 12;
      // This works because i is in scope
      System.out.println(i);

      noDice();
   }

   public String noDice() {
      // This doesn't work because i is not in scope
      // This could be fixed if I passed i as an argument above
      System.out.println(i);
   }
}
```

Example 2 - Loop Scope

```
Class ExampleClass {
   for (int i = 0; i < 5; i++) {
      // This works; it's in scope
      System.out.println(i) ;
   }

   // This does not work; i doesn't exist "past" the loop
   System.out.println(i);
}
```

Example 3 - "Private" Accessibility and Instance Variables

```
Class ExampleClass {
   private int i = 15;
   public static void main(String args[]) {

      // If I'm remembering correctly, this doesn't work, in particular
      // due to the fact that it's assigned "private"; if you specified
      // "println(this.i)", I think it would allow it, but here is no good
      System.out.println(i);
   }
}
```

P167 3.5

Consider the following pseudocode.

```
1. procedure main()
2.     a : integer := 1
3.     b : integer := 2
4.     procedure middle()
5.         b : integer := a
6.         procedure inner()
7.             print a, b
8.         a : integer := 3
9.         —— body of middle
10.        inner()
11.        print a, b
12.    —— body of main
13.    middle()
14.    print a, b
```

Suppose this was code for a language with the declaration-order rules of C (but with nested subroutines)—that is, names must be declared before use, and the scope of a name extends from its declaration through the end of the block.
At each print statement, indicate which declarations of a and b are in the referencing environment. What does the program print (or will the compiler identify static semantic errors)?

Repeat the exercise for the declaration-order rules of C# (names must be declared before use, but the scope of a name is the entire block in which it is declared) and of Modula-3 (names can be declared in any order, and their scope is the entire block in which they are declared).

(Hint: each of the three languages will produce different output; one of the languages will produce an error.)

C

A and B are initially set up in the MAIN() body as 1 and 2, respectively. Then, the MIDDLE() function declared/defined.

Here, a new (local) B is declared, and exists in the scope of MIDDLE(). It's value is set to 1. INNER() is defined here, but not run. A new (local) A instantiates, set to 3. Local to MIDDLE(). INNER() executed. INNER() prints 1 and 1 (A and B at that point).
Line 11 prints A and B as 3 and 1, also. Back in MAIN(), A and B print out as 1 and 2.

For concision, in this order: 1,1; 3,1; 1,2

C#

A and B are declared in MAIN() as 1 and 2. MIDDLE() is defined. In MIDDLE(), global B is set to global A, but I think this causes a SEMANTIC error.

Modula-3

A and B are declared in MAIN() as 1 and 2. MIDDLE() defined. B declared as a local int to MIDDLE, and given global A's value (1). INNER() defined (barely worth mentioning; just prints A and B as they are at that time). New local A is declared and given value 3. INNER() executed, A and B print as 3 and 3. Line 11 prints A and B as 3 and 3. Finally, MAIN() prints A and B as 1 and 2.

In this order: 3,3; 3,3; 1,2

(This language is UNPLEASANT)

P169 3.6a

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

```
procedure main()
   g : integer                  Declare G
   procedure B(a : integer)       Take A
      x : integer               Declare X
      procedure A(n : integer)
         g := n                 Take N, give its value to G
      procedure R(m : integer)   Take M
         write integer(x)
         x /:= 2 —— integer division
         if x > 1
            R(m + 1)
         else
            A(m)
      —— body of B
      x := a × a
      R(1)
   —— body of main
   B(3)
   write integer(g)
```

(a) What does this program print?

9 4 2 3

Main starts B, value 3 -> A; A*A=9. R, value 1 -> M; print X, 9/2 Integer = 4, 4>1
R, value 2 -> print 4, x = 2; R, value 3 -> print 2, x = 1; IF fails, value 3 -> N;
G = N -> G = 3

P171 3.14

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?

Static scoping will print 1122
Static scoping always uses its top-level environment. Therefore, even changing the values inside other functions, the global value is unchanged.

Dynamic scoping will print 1121
Dynamic scoping means that a variables value is referenced locally first, and if there's nothing found, the compiler/interpreter will look "up a level" in the stack trying to find it.