Chapter 3

1. Briefly explain the following terms/concepts.
    a. Activation record
    b. Aliasing
    c. Dangling reference
    d. Garbage
    e. Lifetime
    f. Name binding
    g. Overloading
    h. Scope
    i. Referencing Environment

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

3. 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).

4. 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?

5. C# has unusually sophisticated support for first-class subroutines. Among other things, it allows delegates to be instantiated from anonymous nested methods, and gives local variables and parameters unlimited extent when they may be needed by such a delegate. Consider the implications of these features in the following C# program.

```
using System;
public delegate int UnaryOp(int n);
   // type declaration: UnaryOp is a function from ints to ints

public class Foo {
    static int a = 2;
    static UnaryOp b(int c) {
        int d = a + c;
        Console.WriteLine(d);
        return delegate(int n) { return c + n; };
    }
    public static void Main(string[] args) {
        Console.WriteLine(b(3)(4));
    }
}
```

What does this program print? Which of a, b, c, and d, if any, is likely to be statically allocated? Which could be allocated on the stack? Which would need to be allocated in the heap? Explain.

6. Consider the following pseudocode:

```
procedure P(A, B : real)
    X : real

procedure Q(B, C : real)
    Y : real
    . . .

procedure R(A, C : real)
    Z : real
    . . .                          -- (*)
```

Assuming static scope, what is the referencing environment at the location marked by (*)?

7. 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