

Problem 5.4

4. Dynamic type binding is closely related to implicit heap-dynamic variables. Explain this relationship.

Implicit heap-dynamic variables are bound and allocated at run time and this is performed as-needed by code generated by the compiler (as opposed to in response to commands issued by the programmer). This is the type of binding and allocation that is needed when dynamic type binding is performed.

Problem 5.5

5. Describe a situation when a history-sensitive variable in a subprogram is useful.

Anytime that state information needs to be maintained from one invocation of the program to the next. A simple example would be if a function was keeping track of how many times was invoked. This might be useful for profiling a recursive function.

Problem 5.6

6. Consider the following JavaScript skeletal program:

```
// The main program
var x;
function sub1() {
  var x;
  function sub2() {
    ...
  }
}
function sub3() {
  ...
}
```

Assume that the execution of this program is in the following unit order:

main calls sub1

sub1 calls sub2

sub2 calls sub3

- a. Assuming static scoping, in the following, which declaration of `x` is the correct one for a reference to `x`?
  - i. `sub1`
  - ii. `sub2`
  - iii. `sub3`
- b. Repeat part a, but assume dynamic scoping.

(a)(i) – The 'x' declared in sub1()

(a)(ii) – The 'x' declared in sub1()

(a)(iii) – The 'x' declared in main()

(b)(i) – The 'x' declared in sub1()

(b)(ii) – The 'x' declared in sub1()

(b)(iii) – The 'x' declared in sub1()

Problem 5.7

7. Assume the following JavaScript program was interpreted using static-scoping rules. What value of  $x$  is displayed in function `sub1`? Under dynamic-scoping rules, what value of  $x$  is displayed in function `sub1`?

```
var x;
function sub1() {
  document.write("x = " + x + "<br />");
}
function sub2() {
  var x;
  x = 10;
  sub1();
}
x = 5;
sub2();
```

Under static scoping:  $x = 5$

Under dynamic scoping:  $x = 10$

## Problem 5.8

8. Consider the following JavaScript program:

```

var x, y, z;
function sub1() {
  var a, y, z;
  function sub2() {
    var a, b, z;
    ...
  }
  ...
}
function sub3() {
  var a, x, w;
  ...
}

```

List all the variables, along with the program units where they are declared, that are visible in the bodies of `sub1`, `sub2`, and `sub3`, assuming static scoping is used.

Variable	Declared in	Visible in		
		sub1()	sub2()	sub(3)
x	main()	X	X	
y	main()			X
z	main()			X
a	sub1()	X		
y	sub1()	X	X	
z	sub1()	X		
a	sub2()		X	
b	sub2()		X	
z	sub2()		X	
a	sub3()			X
x	sub3()			X
w	sub3()			X

---

 Problem 5.9

9. Consider the following Python program:

```

x = 1;
y = 3;
z = 5;
def sub1():
    a = 7;
    y = 9;
    z = 11;
    ...
def sub2():
    global x;
    a = 13;
    x = 15;
    w = 17;
    ...
def sub3():
    nonlocal a;
    a = 19;
    b = 21;
    z = 23;
    ...
...

```

List all the variables, along with the program units where they are declared, that are visible in the bodies of `sub1`, `sub2`, and `sub3`, assuming static scoping is used.

Variable	Visible in		
	sub1()	sub2()	sub3()
<b>a</b>	sub1()	sub2()	sub2()
<b>b</b>	--	--	sub3()
<b>w</b>	--	sub2()	sub2() - readonly
<b>x</b>	Main() - readonly	Main()	Main() - readonly
<b>y</b>	sub1()	Main() - readonly	Main() - readonly
<b>z</b>	sub(1)	Main() - readonly	sub3()

In Python, if a variable is not local and it is not declared as `global` or `nonlocal`, then if it is visible in the next higher scope it can be read, but not written, in the local scope. If it is written to, then it becomes a local variable and prior reads become errors (reading a local variable before declaration). This is a subtlety of the language, so if the `readonly` entries are marked as not visible, that is acceptable.

## Problem 5.10

10. Consider the following C program:

```

void fun(void) {
    int a, b, c; /* definition 1 */
    ...
    while (...) {
        int b, c, d; /*definition 2 */
        ... ←———— 1
        while (...) {
            int c, d, e; /* definition 3 */
            ... ←———— 2
        }
        ... ←———— 3
    }
    ... ←———— 4
}

```

For each of the four marked points in this function, list each visible variable, along with the number of the definition statement that defines it.

Variable	Visible in			
	Location 1	Location 2	Location 3	Location 4
a	def 1	def 1	def 1	def 1
b	def 2	def 2	def 2	def 1
c	def 2	def 3	def 3	def 1
d	def 2	def 3	def 3	--
e	--	def 3	def 3	--



## Problem 5.12

12. Consider the following program, written in JavaScript-like syntax:

```
// main program
var x, y, z;

function sub1() {
  var a, y, z;
  ...
}

function sub2() {
  var a, b, z;
  ...
}

function sub3() {
  var a, x, w;
  ...
}
```

Given the following calling sequences and assuming that dynamic scoping is used, what variables are visible during execution of the last subprogram activated? Include with each visible variable the name of the unit where it is declared.

- main calls sub1; sub1 calls sub2; sub2 calls sub3.
- main calls sub1; sub1 calls sub3.
- main calls sub2; sub2 calls sub3; sub3 calls sub1.
- main calls sub3; sub3 calls sub1.
- main calls sub1; sub1 calls sub3; sub3 calls sub2.
- main calls sub3; sub3 calls sub2; sub2 calls sub1.

	Problem Part					
Variable	(a)	(b)	(c)	(d)	(e)	(f)
<b>a</b>	sub3	sub3	sub1	sub1	sub2	sub1
<b>b</b>	sub2	--	sub2	--	sub2	sub2
<b>w</b>	sub3	sub3	sub3	sub3	sub3	sub3
<b>x</b>	sub3	sub3	sub3	sub3	sub3	sub3
<b>y</b>	sub1	sub1	sub1	sub1	sub1	sub1
<b>z</b>	sub2	sub1	sub1	sub1	sub2	sub1