CS162 Homework 4

HW4 #1
Name - a string of characters used to identify some program entity
Address - memory address of the variable (**l-value**)
Lifetime - the program execution time during which the variable is **bound** to a memory location
Scope - the range of statements that can reference the variable
Type - defines the range of values a variable can have and the operations that can be performed on the values

alias – a variable that refers to the same memory location as a variable of a different name

****
HW4 #2
Chpt5 #5. Implicit heap-dynamic variables acquire types only when assigned values, which must be at runtime. Therefore, these variables are always dynamically bound to types.
****
HW4 #3

Chpt5 #8.

(a) i. `sub1`

ii. `sub1`

iii. `main`

(b) i. `sub1`

ii. `sub1`

iii. `sub1`

Chpt5 #9. Static scoping: `x` = 5.

Dynamic scoping: `x` = 10

****
HW4 #4

Chpt5 #10.

Variable	Where Declared
In `sub1`:	
`a`	`sub1`
`y`	`sub1`
`z`	`sub1`
`x`	`main`
In `sub2:`	
`a`	`sub2`
`b`	`sub2`
`z`	`sub2`
`y`	`sub1`
`x`	`main`
In `sub3`:	
`a`	`sub3`
`x`	`sub3`
`w`	`sub3`
`y`	`main`
`z`	`main`

Chpt5 #12.

Point 1:	`a`	1
	`b`	2
	`c`	2
	`d`	2
Point 2:	`a`	1
	`b`	2
	`c`	3
	`d`	3
	`e`	3

Point 3: same as Point 1

Point 4:	`a`	1
	`b`	1
	`c`	1

****

HW4 #5

Chpt5 #13.

	Variable	Where Declared
(a)	d, e, f	fun3
	c	fun2
	b	fun1
	a	main
(b)	d, e, f	fun3
	b, c	fun1
	a	main
(c)	b, c, d	fun1
	e, f	fun3
	a	main
(d)	b, c, d	fun1
	e, f	fun3
	a	main
(e)	c, d, e	fun2
	f	fun3
	b	fun1
	a	main
(f)	b, c, d	fun1
	e	fun2
	f	fun3
	a	main

Chpt5 #14.

	Variable	Where Declared
(a)	a, x, w	sub3
	b, z	sub2

```
      y          sub1
(b)   a, x, w    sub3
      y, z       sub1
(c)   a, y, z    sub1
      x, w       sub3
      b          sub2
(d)   a, y, z    sub1
      x, w       sub3
(e)   a, b, z    sub2
      x, w       sub3
      y          sub1
(f)   a, y, z    sub1
      b          sub2
      x, w       sub3
```
****

HW4 #6

Chpt6 #1. Boolean variables stored as single bits are very space efficient, but on most computers access to them is slower than if they were stored as bytes.

Chpt 6 #2. Integer values stored in decimal waste storage in binary memory computers, simply as a result of the fact that it takes four binary bits to store a single decimal digit, but those four bits are capable of storing 16 different values. Therefore, the ability to store six out of every 16 possible values is wasted. Numeric values can be stored efficiently on binary memory computers only in number bases that are multiples of 2. If humans had developed fingers that were a power of 2, there'd be no problems…
****

HW4 #7

Chpt6 #13. Require all references to enumeration constants to be qualified with the type name, as in colors.blue.

Chpt6 #14. Let the subscript ranges of the three dimensions be named `min(1)`, `min(2)`, `min(3)`, `max(1)`, `max(2)`, and `max(3)`. Let the sizes of the subscript ranges be `size(1)`, `size(2)`, and `size(3)`. Assume the element size is 1.

Row Major Order:

location(`a[i,j,k]`) = (address of `a[min(1),min(2),min(3)]`)

```
     +((i-min(1))*size(3) + (j-min(2)))*size(2) + (k-
min(3))
```

Column Major Order:

location(`a[i,j,k]`)= (address of `a[min(1),min(2),min(3)]`)

```
     +((k-min(3))*size(1) + (j-min(2)))*size(2) + (i-
min(1))
```

****

HW4 #8

GC- a mechanism for automatic deallocation of heap-allocated objects provided by the runtime

PROs: No dangling references
No memory leaks
Less error prone
Opportunity for optimization of object placement to improve the efficiency of the use of the memory hierarchy

CONs: complicates compiler
Less control by programmer
Complicates runtime system (now needs to keep track of allocation & Deallocation
Introduces runtime overhead

****