Wasfi Momen

1.

a. The difference between value and reference types in C# is memory allocation and referencing the memory addresses. Value types take up one memory space and are usually types like primitive types like int, float, boolean, and char. Reference types have one memory space for the actual value (say an object) and one memory space for the reference.

b. string types in c# are reference types.

c. The operations of C# strings behave more like Java String Operations since C# uses a collection of characters to build a string and are immutable. Concatenating strings shows that C# string act more like Java with C#'s StringBuilder class.

2.

a.
b. 1160

3.

a. ((A B) (F)), creates a cons with the first value of L1 and everything except the first value of L2.
b. (F A B), creates a cons with everything except the first value of L2 and the first value of L1
c. (H), G is nil so car returns nothing, H is nil, but the reference is still there.
d. ((A B) ((E) F)) G), creates a list with a nested list. Nested list contains the first value of L1 and all of L2. G is in the outer list.

*Here is the code, I have learned Lisp is not for me.

```
(defparameter E (list 1))
(defparameter F (list 2))

(defparameter L1 (list '(a b) 'c 'd) ) ;; list((A B) C D)

(defparameter L2 (list '(E) 'F )) ;;  list ((E) F)

(print (cons (car L1) (list (cdr L2)))) ;;  (CONS (CAR L1) (LIST (CDR L2)))
(print (cons (car (cdr L2)) (car L1) )) ;; (CONS (CAR (CDR L2)) (CAR L1))
(print (cdr (cons (car (cons 'G L1)) '(H))) );; (CDR (CONS (CAR (CONS 'G L1)) '(H)))
(print (list (list (car L1) L2) 'G)) ;; (LIST (LIST (CAR L1) L2) 'G)
```

4. 0

5. a. 5 bytes
b. 65 bytes
c. 20 bytes

d. 16 bytes
e. 21 bytes
f. 20 bytes
g. 21 bytes

6. a. address of the index value 1 in array a.
b. addresses of the beginning of where the array b resides.
c. two memory spaces ahead of the beginning of where array b resides
d. ILLEGAL
e. offset of the memory locations of r and q.
f. address pointing to the array a plus a memory space
g. address of q[3]
h. 4

7.
a. .5
b. −64.125

8.
a. chart to short is narrowing, since char is by definition an integer type in C. Going from an integer to short is narrowing.

b. unsigned ints can hold all the positive range of int and more, so this would be considered a narrowing of type.

9.
a. $(((a*b)^{1}- 1)^{2} + c)^{3}$
b. $(((a * (b – 1)^{1})^{2} / c)^{3} \text{ mod } d)^{4}$
c. $(((a – b)^{1} / c)^{2} \& ((((d * e)^{3} / a)^{4} - 3)^{5})^{6}$
d. $((-a)^{1} \text{ or } ((c = d)^{2} \text{ and } e)^{3})^{4}$
e. $(((a > b)^{1} \text{ xor } c)^{3} \text{ or } (d <= 17)^{2})^{4}$
f. $(- (a+b)^{1})^{2}$

10. The type of the function parameters cannot be a built-in type. You cannot change how an int will add together with another int.

11. The differences between all of these function prototypes lies in the ability of the compiler's linkage. All of these functions have a keyword void, which means they do not return a value. The difference is that the keywords of extern and static mean that the function has explicit linkage. Extern means that the function can be used in other compilation units, while static specifies that the function can only be used within the specific unit it is defined aka the file that it's in. Void is the same as extern since it implcitly has external linkage.