Practice Quiz 2

This contains some EXTRA questions as well.

1	An array is a group of contiguous memory locations that all have the same type.	**True** False
2	The index of the first element in an array is 1.	True **False**
3	The following definition int c[12] reserves 12 elements for integer array c, which has indices in the range 1-12.	True **False**

4	What is the value of n[4] after running the following code:
	
	int main(void){
	    int n[5];
	    for (size_t i = 0; i < 5; ++i) {
	    n[i] = 0;
	}
		A. 0
		B. 1
		C. 5
		D. out of range

5	int n[] = {1, 2, 3, 4, 5}; is a legal statement to create an array.	**True** False
6	If int n[] = {1, 2, 3, 4, 5}; , then n[1]++ will	A. add 1 to the index B. add 1 to the value of first element **C. add 1 to the value of second element**
7	intn[10] = {0}; will	**A. initializes entire array to zeros** B. initializes the tenth element of the array to 0 C. illegal statement
8	inty = 5;int *yPtr; yPtr= &y; what is the value of yPtr?	A. the value of y **B. the address of the variable y** C. 5

9	Consider the following code:
	
	#include <stdio.h>
	int main(void)
	{
	int x;
	int y;
	*ptr = 7;
	ptr = &y;
	}
		A. *ptr = 7;
		B. ptr = &y;
		C. Both

	which one is allowed?	
10	If there are fewer initializers in an initializer list than the number of elements in the array, C automatically initializes the remaining elements to the last value in the list of initializers.	True **False** C automatically initializes the remaining elements to zero
11	It's an error if an initializer list contains more initializers than there are array elements.	**True** False
12	An individual array element that's passed to a function as an argument of the form a[i] and modified in the called function will contain the modified value in the calling function.	True **False.** Individual elements of an array are passed by value. If the entire array is passed to a function, then any modifications to the elements will be reflected in the original.
13	A pointer that's declared to be void can be de-referenced	True **False** A pointer to void cannot be dereferenced, because there's no way to know exactly how many bytes of memory to dereference.
14	Pointers of different types may not be assigned to one another without a cast operation.	True **False** Pointers of type void can be assigned pointers of other types, and pointers of type void can be assigned to pointers of other types.
15	A pointer variable contains as its value the ___ of another variable	a. Copy **b. Address** c. Value
16	Value(s) that can be used to initialize a pointer:	a. 0 b. Null c. Address **d. All of the above** e. Two of the above f. None of the above

Q1:

```
#include <stdio.h>
void f(const int *xPtr); // prototype

int main(void)
{
   int y; // define y

   f(&y); // f attempts illegal modification
}

void f(const int *xPtr)
{
   *xPtr = 100; // error: cannot modify a const object
}
```

Options:

1. Error because attempting to modify a constant pointer to non-constant data.
2. Error because attempting to modify a constant pointer to constant data.
3. **Error because attempting to modify data through a non-constant pointer to constant data.**

Q2:

```
#include <stdio.h>

int main(void)
{
   int x; // define x
   int y; // define y

   int * const ptr = &x;

   *ptr = 7;
   ptr = &y;
}
```

Options:

1. **Error because attempting to modify a constant pointer to non-constant data.**
2. Error because attempting to modify a constant pointer to constant data.
3. Error because attempting to modify data through a non-constant pointer to constant data.

Q3:

```
#include <stdio.h>
```

```
int main(void)
{
   int x = 5; // initialize x
   int y; // define y

   const int *const ptr = &x; // initialization is OK

   printf("%d\n", *ptr);
   *ptr = 7;
   ptr = &y;
}
```

Options:

4. Error because attempting to modify a constant pointer to non-constant data.
5. **Error because attempting to modify a constant pointer to constant data.**
6. Error because attempting to modify data through a non-constant pointer to constant data.

Q4:

What is the purpose of the following function:

```
size_t mysteryFunction(const int array[], int key, size_t size)
{
   for (size_t n = 0; n < size; ++n) {
      if (array[n] == key) {
         return n;
      }
   }

   return -1;
}
```

Options:

1. Unique elements in an array
2. Binary search of an array.
3. **Linear search of an array.**
4. Bubble sort.

Q5:

What is the purpose of the following function:

```
size_t SecretFunction(const int b[], int searchKey, size_t low, size_t
high)
{
   while (low <= high) {

      size_t middle = (low + high) / 2;

         printRow(b, low, middle, high);

      if (searchKey == b[middle]) {
         return middle;
      }

      else if (searchKey < b[middle]) {
         high = middle - 1; // search low end of array
      }

      else {
         low = middle + 1; // search high end of array
      }
   } // end while

   return -1; // searchKey not found
}
```

Options:

5. Unique elements in an array
6. **Binary search of an array.**
7. Linear search of an array.
8. Bubble sort.

Q6:

What is the purpose of the following code:

```
for (unsigned int pass = 1; pass < SIZE; ++pass) {

      for (size_t i = 0; i < SIZE - 1; ++i) {

         if (a[i] > a[i + 1]) {
            int hold = a[i];
            a[i] = a[i + 1];
            a[i + 1] = hold;
         }
      }
```

Options:

1. Unique elements in an array
2. Binary search of an array.
3. Linear search of an array.
4. **Bubble sort.**