C vs. Java

For many operations, C is similar to Java. However, there are a number of key differences between the languages:

Java	C
Object oriented	No objects, inheritance, or polymorphism
"Methods"	"Functions"
Class libraries of data structures	Lower level libraries
Automatic memory management	Manual memory management
Strongly typed	Weakly typed (can typecast all types to any other type)
Compiles to machine independent bytecode	Compiles to machine *dependent* assembly
Initializes variables to default (e.g. 0)	Variables start as "garbage"
	Exposes addresses in the form of *pointers*

Pointers

A pointer is a variable that contains the memory address of a variable.

- To declare a pointer, you write:
  `<type> *x;` or `<type>* x;` (declares x as a pointer to `<type>`)
- The dereference operator (`*`) is used to obtain the *value* of a pointer:
  `y = *x;` (assigns the value of x to y).
- The address operator (`&`) is used to obtain the *address* of a pointer:
  `z = &x;` (assigns the address of x to z).

1. What is terrible about the following function?

```
int *blasphemy(void) {
    int x; return &x;
}
```

This function returns a pointer to memory that was allocated on the stack, but deallocated after the function returns. There's no guarantee that this memory will contain or hold any particular value, or that things won't crash when you try to access it.

2. What is wrong about the following function that swaps the values of two `int` variables? What changes need to be made for `swappy` to function correctly?

```
/* Swaps the values of A and B. */
void swappy(int *a, int *b) {
    int tmp = *a;
    *a = *b;
```

```
    *b = tmp;
}
```

Arrays

An array in C is essentially a consecutive chunk of memory.

- To declare an array, you write:
  `<type> arr[n];` (declares arr as a n-element array of <type>)
  `<type> arr[] = {<v1>, <v2>};` (initializes arr as a 2-element array of <type>).
- The name of an array is the pointer to the first (0^{th}) element of the array:
  `*arr = arr[0];`
- You can also do pointer arithmetic and access other elements of the array:
  `*(arr+i)` is equivalent to `arr[i]`, which gives the i^{th} element of arr.

Strings

A string in C is stored as a character array (`char[]`) that ends with the null terminating character, '`\0`' (ASCII=0; this is different from the character zero, '`0`', whose ASCII=48).

3. Implement the `strcpy` function. Assume `SRC` and `DST` are properly declared.

```
/* Copies the string SRC to DST. No return value required. */
void strcpy(char* dst, const char* src) {
    // order: assign -> increment -> check while
    while (*dst++ = *src++);
}
```

4. (MT1 Sp12 - Graph Representation) A graph is represented as an array of `edge_t` pointers terminated by a null pointer. Given the following implementation, draw a possible pointer diagram starting from `G`, assuming the graph it points to has exactly two edges and three vertices.

```
typedef struct edge {
    vertex_t *first;
    vertex_t *second;
} edge_t;
```

```
/* The following pointer to a
   graph is declared: */
   edge_t** G;
```