CSE 250 Spring 2011
Homework 2
Due Date: Feb 28, Monday, by 2:05pm
Total Points: 25

1. Yes, **foo** does swap the values of the array elements **i** and **j** when they are in bounds. If one or both of them is out of bounds, **foo** will still try to change those locations in memory, which may result in a segmentation fault, or just erroneous values being swapped into the valid region of memory for `a[]`.

2.
   - The line `x[X_SIZE] = 5;` is indexing out of bounds, which may result in a run-time error (Segmentation Fault), but may instead execute and overwrite some other variable leading to strange behavior or data.
   - The line `x[n] = 10/n;` would throw an Arithmetic Exception run-time error if it followed the previous line (`int n = 0;`), because you'd be trying to divide by 0.

3. `Computer c1;` - Declares a Computer object.
   `Computer c2(‘‘Ace’’,‘‘AMD Athlon 2000’’, 512, 60);` - Declares and instantiates a Computer object, using the constructor to set the data fields to the specified parameters.
   `Lap_Top c3(‘‘Ace’’,‘‘AMD Athlon 2000’’, 512, 60);` - This is an invalid statement and will not compile.
   The Lap_Top class only has one constructor, which requires two more arguments than the Computer constructor.
   `Lap_Top c4(‘‘Ace’’,‘‘AMD Athlon 2000’’, 512, 60,15.5, 7.5);` - Declares and instantiates a Lap_Top object.
   `cout << c2.manufacturer << ", " << c2.processor << endl;` - Compilation Error, manufacturer and processor are private variables.
   `cout << c2.get_disk_size() << ", " << c4.get_ram_size() << endl;` - Outputs "60, 512".
   `cout << c2.to_string() << "\n" << c4.to_string() << endl;` - Outputs the following:

```
Manufacturer: Ace
CPU: AMD Athlon 2000
RAM: 512 megabytes
Disk: 60 gigabytes
Manufacturer: Ace
CPU: AMD Athlon 2000
RAM: 512 megabytes
Disk: 60 gigabytes
```

4. In this question, although you could technically get some of the erroneous lines to compile, the idea here is to identify the method signatures that don't make sense with the underlying class.
   `Computer()` - Constructor in class Computer
   `Lap_Top()` - Constructor in class Lap_Top
   `int to_string()` - Not reasonable, to_string should return a string.
   `double get_ram_size()` - Accessor in class Computer, but should return an int
   `string get_ram_size()` - Accessor in class Computer, but should return an int
   `string get_ram_size(string)` - Accessors should not take parameters
   `string get_processor()` - Accessor in class Computer
   `double get_screen_size()` - Accessor in class Lap_Top

5. As in Example 3.1 on pg. 200, our array of pointers is to `Computer` objects, but these pointers could either point to `Computer` objects or `Lap_Top` objects, since `Lap_Top` is a subclass of `Computer`. So when we access the `to_string` member function, the appropriate function to call (either `Computer::to_string` or `Lap_Top::to_string`) will be determined at run-time depending on what type of object each `comp[i]` points to. It's also worth noting that `Lap_Top` has an extra constructor which sets a default manufacturer in case it is not supplied, and that is the constructor used for `comp[1]` (and `comp[2]`, since it points to the same object). The output will be:

```
512
Manufacturer: Ace
CPU: AMD Anthlon 2500
RAM: 512 megabytes
Disk: 60 gigabytes
256
Manufacturer: MyBrand
CPU: Intel P4 2.4
RAM: 256 megabytes
Disk: 40 gigabytes
Screen Size: 15.5
Weight: 7.5
256
Manufacturer: MyBrand
CPU: Intel P4 2.4
RAM: 256 megabytes
Disk: 40 gigabytes
Screen Size: 15.5
Weight: 7.5
```

6. Abstract member functions (pure virtual functions) can only be declared in abstract classes. Declaring a pure virtual function in a class makes that class abstract. Abstract classes cannot be instantiated (created), but pointer variables of abstract class types may be declared. Both abstract classes and actual classes can be subclassed, but the difference here is with abstract classes, you are forced to instantiate one of the subclasses (if it's not abstract) because you can't create an object of the abstract class type.

7.
   - `my_vector.push_back("Pokey")` adds "Pokey" at index 6 of `my_vector`.
     `my_vector.size() == 7`.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
"Bashful"	"Awful"	"Doc"	"Jumpy"	"Happy"	"Dopey"	"Pokey"

   - `my_vector.push_back("Campy");` adds "Campy" at index 7 of `my_vector`.
     `my_vector.size == 8`.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
"Bashful"	"Awful"	"Doc"	"Jumpy"	"Happy"	"Dopey"	"Pokey"	"Campy"

   - `int i = find(my_vector, "Happy");` will return `i == 4`.
     `my_vector.size() == 8`
   - `my_vector[i] = "Bouncy";` replaces `my_vector[4]` with the string "Bouncy".
     `my_vector.size() == 8`

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
"Bashful"	"Awful"	"Doc"	"Jumpy"	"Bouncy"	"Dopey"	"Pokey"	"Campy"

- `my_vector.erase(my_vector.size() - 2);` will remove the second to last entry (`my_vector[6]` == "Pokey") and shift all entries after that to the left by one.
  `my_vector.size() == 7.`

[0]	[1]	[2]	[3]	[4]	[5]	[6]
"Bashful"	"Awful"	"Doc"	"Jumpy"	"Bouncy"	"Dopey"	"Campy"

- `string temp = my_vector[1];` sets the string variable `temp` equal to "Awful".
  `my_vector.size() == 7`
- `my_vector[1] = to_upper(temp);` sets the entry at index 1 to all uppercase.
  `my_vector.size() == 7`

[0]	[1]	[2]	[3]	[4]	[5]	[6]
"Bashful"	"AWFUL"	"Doc"	"Jumpy"	"Bouncy"	"Dopey"	"Campy"

8. The following code will accomplish the desired task:

```
void replace(vector<string>& a_vector, const string& old_item, const string& new_item) {
    for (size_t i=0; i < a_vector.size(); i++) {
        if (a_vector.at(i).compare(old_item) == 0) {
            a_vector.insert(a_vector.begin()+i, new_item);
            a_vector.erase(a_vector.begin()+(i+1));
        }
    }
    return;
}
```

9. The following code will accomplish the desired task:

```
template<typename Item_Type>
void delete_v(vector<Item_Type>& a_vector, const Item_Type& target) {
    for (size_t i=0; i < a_vector.size(); i++) {
        if (a_vector.at(i) == target) {
            a_vector.erase(a_vector.begin()+i);
            break;
        }
    }
    return;
}
```

Note: Unfortunately, this question isn't worded very well, because `delete` is a C++ keyword, so you will not be able to compile code that has a function with this name. However, if you're trying to write this function in code and test it, you can just call it something else like `vector_delete`, and it will compile and run as desired.