COMP 1406C – Introduction to Computer Science II
Quiz 1, Winter 2015

This is a closed book test. No calculators are allowed. All questions should be answered directly on this sheet in space provided. You have 45 minutes to complete this test. Assume that all C++ code (both given to you and that you write) is prefixed with the lines **`#include`** `<iostream>`, **`#include`** `<string>` and **`using namespace`** `std;`.

		TA Use Only
Name		
Student #	S O L U T I O N S	Grade (/20) TA

1. *(2 points)* When trying to solve a problem, why is it sometimes useful to restate the problem?

1 pt - identifying reason
1 pt - explanation

- view from different angles
- "circle base of hill before climbing"
- confirm understanding

2. *(3 points)* In C++, explain what happens when you say **`char`** `a;`. Compare this to what happens when you say **`new char;`**.

1 pt - explain char a;
1 pt - explain new char;
1 pt - indicate difference (stack/heap or compile/run)

char a; → set aside space on stack at compile time for a character.

new char; → set aside space on heap at run time for a character, return pointer to that space.

For the next two questions, assume that the following **struct** has been defined:

```
struct hydralisk {
    int health;
    int maximum_health;
};
```

3. *(3 points)* What is the output of the following C++ code? Why?

```
void healHydralisk(hydralisk h) {
    h.health += 10;
    if(h.health > h.maximum_health) {
        h.health = h.maximum_health;
    }
}

int main() {
    hydralisk joe = { 10, 45 };
    cout << joe.health << endl;
    healHydralisk(joe);
    cout << joe.health << endl;
    return(0);
}
```

Output: 10 → 1 pt
10

Reason: joe is passed by value and so only the health of the copy is changed.
1 pt: "pass by value"
1 pt: "does not change joe"

4. *(4 points)* The following C++ code compiles without any errors, but there are two problems that occur at run-time. Identify and explain both of these problems.

```
int main() {
    hydralisk army[3];
    for(int i = 0; i < 3; i++) {
       army[i].health = 45;
    }
    for(int i = 0; i < 4; i++) {
        if(army[i].health < army[i].maximum_health) {
            cout << "Hydralisk " << i << " is damaged!" << endl;
        }
    }
}
```

1 pt: identify
1 pt: explain

1. The second for-loop goes to i=3, which is past the end of the array. This could cause garbage to be compared in the if-statement, and so the output is not predictable for the last iteration.

same

2. The first for-loop never initializes the maximum_health member, and so it will contain garbage.

5. *(8 points)* In this question, you will have to write some C++ code. It does not have to be perfect: missing a semicolon (for example) is fine. However, given the code you write here, it should be very straightforward to get it to compile with minimal effort.

1 pt (a) Define a **`struct`** that represents a soda. It should store the brand, the volume (in milliliters), and the *deliciousness* on a scale from 1-10. It is up to you to determine what types to use for each of these.

2 pts (b) Create (on the stack) an array that consists of three sodas **`struct`**s. Initialize each soda values of your choice.

2 pts (c) Write a function that takes an array of sodas and returns the name of soda with maximum volume. The function should not change the array or its contents.

2 pts (d) Write a function that takes a single soda and decreases its deliciousness by one. The function should not return anything.

1 pt (e) Call each of these functions once from `main()`.

(a)
```
struct soda {
    string brand;
    int volume;          // ok if float
    int deliciousness;
};
```

(b)
```
soda sodas[3];
sodas[0].brand = "Coke";
sodas[0].volume = 355;
sodas[0].deliciousness = 8;
(etc.)
```
assume this is in main.

(c)
```
string biggest(soda sodas[], int size){
    int biggest_index = 0;
    for(int i = 1; i < size; i++){
        if(sodas[i].volume > sodas[biggest_index].volume){
            biggest_index = i;
        }
    }
    return sodas[biggest_index].brand;
}
```
-1 if missing

(d)
```
void decrease(soda &s) {
    s.deliciousness--;
}
```
ok to pass pointer, but -1 if passing just a struct by value.

(e)
```
cout << biggest(sodas, 3)
     << endl;
cout << sodas[0].deliciousness
     << endl;
decrease(sodas[0]);
cout << sodas[0].deliciousness
     << endl;
```
Doesn't matter what is done, as long as calls are made.