PIC 10A 1C Week 5b Problems. TA: Eric. [Solutions]

1. The Declaration of Function-pendence

Based off of the following code snippets, infer what the function's declaration would be. The first is done for you.

Code	**Function Declaration**

`int a = 1; double b = 3.14;`
`int result = myfn(a,b);`
	int myfn(int arg1, double arg2);

`string name = "Nathan";`
`char c = fn2(name, name);`
	char fn2(string arg1, string arg2);

`fn3(string("hi").substr(1,1), 42.0);`	void fn3(string, double); Note: return type could be anything, ambiguous.

2. Classy Interfaces

Louis Reasoner is working on some classes. Given the class interface and usage, point out any possible errors. Assume that no libraries/namespaces have been included/set yet.

```
class Minion {
private:
  int myhealth;

  void perish();

  string sing();
public:
  int myid;

  Minion(int health);

  void attack(Minion victim);

  void take_damage(int
damage);

  int get_health();
};
```

```
int main() {
  Minion bob = Minion(10);

  Minion joe;

  attack(bob);

  bob.take_damage(2);

  cout << 'Bob health: '
    << bob.get_health() << endl;

  cout << "Bob attacks back! "
    << bob.attack(joe);

  joe.take_damage();

  cout << "Joe health: " << joe.myhealth;

  cout << "Bob ID: " << bob.myid;
  return 0;
}
```

[Solution]
#include <iostream> /* Don't forget to include these! */
#include <string>
using namespace std; /* Alternately, do std::cout, std::endl, std::string, etc. */

```
int main() {
  Minion bob = Minion(10); // This is fine, since a one-argument public
constructor Minion::Minion(int health) is declared in the class interface.

  Minion joe; // ERROR: Constructor Minion() does not exist! Either have to
declare a no-argument constructor in class interface, or add an integer
here, ie: Minion joe(10);

  attack(bob); // ERROR: attack() is not a function, but is a member
function of the Minion class. Should do something like: joe.attack(bob).

  bob.take_damage(2);

  cout << 'Bob health: ' // ERROR: Use double quotes, not single quotes!
String literals use double-quotes, char's use single-quotes.
    << bob.get_health() << endl;

  cout << "Bob attacks back! "
    << bob.attack(joe); // ERROR: Minion::attack returns nothing (void),
can't output nothing!

  joe.take_damage(); // ERROR: Minion::take_damage expects an int argument!

  cout << "Joe health: " << joe.myhealth; // ERROR: Minion::myhealth is
declared as private, can't access it outside of class. Should instead use
Minion::get_health(), which is public!

  cout << "Bob ID: " << bob.myid; // This is fine, since Minion::myid is
declared public.
  return 0;
}
```

3. Inferring the Interfaces

Given the following code snippet, infer what the class interface for the Goblin class is.

```
Goblin gunth("Gunth");
Hero arthur("Arthur");
int dmg = gunth.attack(arthur);
```

```
cout << gunth.get_name() << " attacked " << arthur.get_name() << "!" <<
endl;
cout << "Damage: " << dmg << endl;
cout << "The goblin sings the song of his people: " << gunth.sing();
gunth.restore_health();
cout << "The goblin restored health!\n");
return 0;
```

```
class Goblin {
// FILL ME IN!
// Note: There is some flexibility here, but here's one answer:
private:
  string myname; // could be public or private, but conventionally this
                 // would be private.
public:
  Goblin(string name);
  string get_name();
  int attack(Hero victim); // Note: returning double would work too
  string sing(); // Return type can be anything, but not void (ambiguous)
  void restore_health(); // Return type can be anything (ambiguous)
}
```

4. Fizzbuzz Lite

Write a program that asks the user for an integer. If the integer is divisible by 3, display "Fizz". If the integer is divisible by 5, display "Buzz". If it's divisible by both 3 and 5, then print "Fizzbuzz". Otherwise, simply display the number.
Hint: To check if an integer a is divisible by an integer b, use the mod operator. "`(a % b) == 0`" is true when a is divisible by b.

YOUR CODE HERE

```
#include <iostream>
using namespace std;
int main() {
    int n;
    cout << "Please input an integer: ";
    cin >> n;
    if (((n % 3) == 0) && ((n % 5) == 0)) {
        cout << "Fizzbuzz";
    } else if ((n % 3) == 0) {
        cout << "Fizz";
    } else if ((n % 5) == 0) {
        cout << "Buzz";
    } else {
        cout << n;
    }
    return 0;
}
```

5. Don't Lose Your Head!

Recall that we use header guards (#ifndef, #define, #endif) to avoid accidentally defining variables/classes multiple times. Consider the following header files:

File: myheader1.h	**File: myheader2.h**

`#ifndef _MYHEADER1_`
`#define _MYHEADER1_`
`int MY_X = 100;`
`#endif`
	`int MY_X = -2;`

```
#include <iostream>
#include "myheader2.h"
int main() {
    std::cout << "MY_X is: " << MY_X;    return 0;
}
```

What is the output of the program? If it crashes, describe the crash.

MY_X is: -2
Explanation: Even though two header files exist (myheader1.h, myheader2.h), we only include myheader2.h, so myheader1.h doesn't cause any problems with multiply-defined MY_X.

Next, consider the following header files:

File: h1.h	**File: h2.h**

`#include "h2.h"`
`#ifndef _H_`
`#define _H_`
`int MY_X = 100;`
`#endif`
	`#ifndef _H_`
	`#define _H_`
	`int MY_X = -50;`
	`#endif`

```
#include <iostream>
#include "h1.h"
int main() {
    std::cout << "MY_X is: " << MY_X;    return 0;
}
```

What is the output of the program? If it crashes, describe the crash.

MY_X is: -50
Explanation: When we include "h1.h", the first thing h1.h does is include "h2.h". Notably, "h2.h" defines the _H_ preprocessor variable, then defines MY_X.
When "h1.h" finishes including "h2.h", it sees that _H_ is already defined (due to h2.h), and does *not* process the code within its own header guard (ie "int MY_X = 100;"). Thus, by using header guards, we avoid a multiply-defined variable error/

6. Static Cling

For the following, insert **at most one** `static_cast` to make the expression evaluate to the desired value. Note that `static_cast` may not be necessary!

Code:	**Desired Output:**

`cout << (3.0 / 2) + 1;`
`cout << static_cast<int>(3.0 / 2) + 1;`
	2

`cout << (6 / 3) + 1;`
`cout << static_cast<double>((6 / 3) + 1);`
Note: static_cast can really go anywhere.
	3.0

`cout << (2/4) + 0.5 + (5/4);` No need to add static_cast.	1.5

7. Storage Wars

Recall that, in a computer, there are three places to store data: primary storage (ie RAM), registers, and secondary storage (ie hard disk). Compare and contrast these three components. In particular, what are the strengths/weaknesses of each component?

[Solution]: In order of size/capacity, from smallest to largest: registers, primary storage (RAM), secondary storage (hard disk). In 2016, we can store about a few dozen int's in registers. A laptop will typically have more than 8 GB of RAM, with a hard drive space of around 500 GB. However, the larger the storage is, the slower it is. Accessing a register is many times faster than accessing RAM, which is many many times faster than accessing the hard disk.
Also, secondary storage is for long-term (non-volatile) storage - data is not lost when power is shut down. However, primary storage and registers are volatile - data is lost when power is shut down.