Code Explanation

By Soenke Van Lohn and Tjeerd Bakker
Design choices are explained as comments in the code
The page is a little over 1 page, but there's some code examples that take up space.

Why did you have to include `iostream` in your code?

IOStream is a part of the standard library that handles IO to the terminal. Since the user needs to input information into the terminal, and the program prints output to the terminal, we need IOStream to interface with the terminal.

What is the difference between private and public access specifiers in a C++ class?

Private methods and variables **are not** accesible from outside the class, while Public methods and variables **are** accesible from outside the class.

```
class Testclass {
private:
    int private_int;     // This is private
public:
    int public_int;      // This is public
};

int main(){
    Testclass test;
    test.public_int = 1;     // This is fine
    test.private_int = 1;    // This is not possible
    return 0;
};
```

Explain the purpose of data hiding.

Data hiding is an important concept in OOP programming. When you hide object data you can make sure that you don't accidentally (or intentionally) change properties of an object. This results in easier debugging and cleaner, more consistent code.

How can a program use the standard library class string without using a using namespace directive? What is problematic about putting a using namespace directive at the start of your code?

```
#include <iostream>
using namespace std;
cout << "This is bad" << cin;
```

```
#include <iostream>
std::cout << "This is better" << std::cin;
```

When using `using namespace`, **any** name of a class, function or variable in the specified namespace is now reserved for that library. If you were to create your own function that accidentally had the same name as a function in the namespace the compiler would throw an error (or way worse, use the wrong function).

Give the output of the following code and explain

This question covers the concept of scopes. A variable defined globally is accesible everywhere (`int a = 20;`), but a variable defined locally is only accesible from the local scope (`int a = 10;`). Local scope takes presedence over global scope.

The output will first print `20` and `10`

```
#include <iostream>
void function();        // Declare the function
int a = 20;             // Globally declare and define 'a'
int main() {
    int a = 10;         // Locally declare and define 'a'
    function();         // Print 'a' (20) from global scope
    std::cout << a;     // Print 'a' (10) from local scope
    return 0;
}
void function() {
    std::cout << a << std::endl; // Print 'a'. This will be the global 'a' (20)
}
```