Library Types in Interfaces

If the prototype includes any types from the standard library (such as `string` or `vector`), then you must `#include` the correct header, and **fully qualify** the name of the type. Here's an example:

```
// Header file
std::string zipZap(const std::string& str);
// Implementation file
string zipZap(const string& str) { . . . }
```

Header files **should never** use identifiers from the standard library without explicitly including the `std::` qualifier. In the implementation file, you may use the name as is, because your implementation file will contain a `using` declaration or directive.

Here are **three rules to remember**.

- **Never** add `using namespace std` to a header file. Header files are `#included` in other files; doing so changes the environment of that file.
- **Always** add `std::` in front of **every** library type, such as `std::string`, but **never** in front of primitive types like `double`.
- For all library types, `#include` the appropriate header file inside of your header file. If you use the `std::string` type, you must `#include <string>` Note that when including standard libraries, you enlose them in angle brackets (`<>`), while your header files use double quotes when included.

Linker Errors

Once you have finished prototyping all three functions, build your project again by by typing `make client`. When you do, you **won't** see any compiler error messages; the client program compiles. However, you will see some **linker error messages**. Your function was **declared** correctly, but the **definition** could not be found at linking time.

```
$ make client
client.cpp:(.text+0x32): undefined reference to
`firstDigit(int)'
```

*If you still see **undeclared** (instead of **undefined**), make sure you have added the line `#include "digit.h"` to the top of the client program.*

Two words to note in your compiler's error messages: **undefined** and **undeclared**. Recognizing these will help you locate and fix the problem.

- An **undeclared** error message is a **compiler** syntax error. It means you are missing a prototype or you are calling a function incorrectly.
- An **undefined** error message comes from the **linker** and means that you are missing the definition for a function.