Modularity using the Linker

These example files, along with an example Makefile, are available at `~s4264352/public/linkerExample`

In Assignment 3 you need to create several executables. These all have very similar functionality (as players) and only differ in small ways (their playstyle). One way to achieve this is by using the linker.

Let's take a very simple example program:

```c
#include "shared.h"

int main(int argc, char** argv) {
	message();
	printf("Done\n");
	return 0;
}
```

If you try and compile this by itself, it will fail because the linker cannot find a reference to `message()`. Now, say we want one version of this program to print one message, and a different version to print another message.

Let's take a look at `shared.h` first:

```c
#ifndef SHARED_H
#define SHARED_H

#include <stdio.h>

void message(void);

#endif
```

In addition to a simple include guard and including `stdio.h`, it just defines message as a function that takes and returns nothing. Now we get to the modular part!

Inside `hello.c` we'll put:

```c
#include "shared.h"

void message(void) {
	printf("Hello, world!\n");
}
```

Inside `goodbye.c` we'll put:

```c
#include "shared.h"

void message(void) {
	printf("Goodbye, world...\n");
}
```

We now compile `main.c`, `hello.c` and `goodbye.c` with the -c flag in gcc.

`gcc -c main.c -o main.o`.

Repeat this for hello and goodbye. This produces object files - `main.o` has everything needed to run the main program *except* the `message` function. We can now link together the unchanged main with the `hello` version of `message`.

```
gcc main.o hello.o -o hello
```

Do the same for `goodbye`.

```
gcc main.o goodbye.o -o goodbye
```

This leaves us with 2 executables that print different things. The important part here is the main function doesn't have to change - in your assignment, your main component will be far more complex.

NOTE: The `message()` declaration in `shared.h` doesn't actually have to be in a separate file - as long as the definitions match, this example would work fine if the contents of `shared.h` were in the main source file. However, it's good practice to put shared functions in a `.h`