Header Files

In general, it is good practice to keep your code modular and organized to maintain readability. One caveat of doing so, however, is that you may run into the case where source files quickly become very long and going through it to find exactly what you want may be a nightmare. To better organize our source code, we can split functions up into separate source files.

Let's pretend we're building a game and our source file looks somewhat like this:

game.c

```
#include <stdio.h>

int score = 0;    // global variable

void update_score(int amt) {
    ...
}

void render_score() {
    ...
}

void render_board() {
    ...
}

void create_board(char *config) {
    ...
}

char *get_winner() {
    ...
}

void check_if_done() {
    ...
}
```

game.c (continued)

```
int add_user(char *name) {
    ...
}

int remove_user(char *name) {
    ...
}

char *move_user(char *name) {
    ...
}

void end_game() {
    ...
}

void start_game() {
    ...
}

void reset_game() {
    ...
}

int change_level(int level_id) {
    ...
}
```

(This example isn't very lengthy, but use your imagination to scale it up.)

The functions above could be regrouped into separate files, with each file containing a subset of functions dealing with a particular aspect of the game. There are many ways to divide the functions, and it depends on the programmer. One *possible* division may be the following:

- Rendering functions (i.e. visual appearance)
- Functions related to score-keeping
- Functions which affect the game's state
- Functions for maintaining user status

Here's a table showing what the new source files may look like:

render.c	score.c	state.c	users.c
render_score() render_board()	update_score() check_if_done() get_winner()	start_game() end_game() reset_game() change_level()	add_user() remove_user() move_user()

Note:
As mentioned above, the grouping of functions is ultimately up to the developer. You may have had something else in mind for how to divide the functions, and that's completely fine as long as it makes sense.

Now that everything's been split up, our source files will look much cleaner, but they may no longer work properly. What if a function in one file wants to call a function in another file? For example, when you update the score (`score.c > update_score`), you may want to modify the visual rendering of the score as well (`render.c > render_score`).

To allow our program to make use of functions across various files, we need to add a **header file** to make our compiler aware of functions which exist in other files, and to link the files together.

Header files look like regular C files. They may contain macro definitions, variable names, as well as function prototypes, but they do **not** contain function bodies. **Do not define a function within a header file!**

Given the functions above, here's what the header file could look like:

game.h

```
void update_score(int);
void render_score();
void render_board();
void create_board(char *);
char *get_winner();
void check_if_done();
int add_user(char *);
int remove_user(char *);
char *move_user(char *);
void end_game();
void start_game();
void reset_game();
int change_level(int);
```

Note the following:

- The file extension for header files is ".h", not ".c".
- You must specify the return type and parameter types for each function.
- You do not need to include the parameter names, but you're free to do so.

So where do we define the function bodies? Within our C files.

To simplify, for each of the functions in the header file, the header file tells the compiler: "Hey, there's this function named `<name>`, whose return type is `<thing>`, and takes parameters `<params>`. It is implemented in one of the C files which include this header, and I want to make this function available to every other C file which includes this header."

To include the header in a C file, you would write the following in the C file:
```
#include "game.h"
```

This looks very similar to how we would include a standard library (i.e. `stdio.h`), but instead of brackets, we use quotation marks.

Now, when you compile the program and execute it, the compiler will search all the C files which include the header to find where each function is implemented, and then make that function available to the rest of the C source files which make use of the same header.

When compiling the program, you must include the name of every C file which implements the functions in the header file(s), like so:
```
gcc -Wall -std=c99 render.c score.c state.c users.c
```

Note that you will run into errors in the following cases:

- If a function in a header file is defined more than once across the various C files use #ifndef
- If a function in a header file is called, but not defined in any of the C files

Finally, note that one of the C files must have a "main" function so that the program can be executed (and knows where to begin execution).