Hello

Next

Listing Files

Hello, world! At right, in the *text editor*, is the very first program we wrote in C, in a file called `hello.c`.

Click the folder icon, and you'll see that `hello.c` is the only file that's present at the moment. Per the mention of `/root/sandbox` below that icon, `hello.c` happens to be in a folder (otherwise known as a *directory*) called `sandbox`, which itself is in another folder called `root`. Click the folder icon again to hide all that.

Next, in the *terminal window* at right, immediately to the right of the dollar sign (`$`), otherwise known as a *prompt*, type precisely the below (in lowercase), then hit Enter:

```
ls
```

You should see just `hello.c`? That's because you've just listed the files in that same folder, this time using a command-line interface (CLI), using just your keyboard, rather than the graphical user interface (GUI) represented by that folder icon. In particular, you *executed* (i.e., ran) a command called `ls`, which is shorthand for "list." (It's such a frequently used command that its authors called it just `ls` to save keystrokes.) Make sense?

Here on out, to execute (i.e., run) a command means to type it into a terminal window and then hit Enter. Commands are "case-sensitive," so be sure not to type in uppercase when you mean lowercase or vice versa.

Next

Compiling Programs

Now, before we can execute the program at right, recall that we must *compile* it with a *compiler* (e.g., `clang`), translating it from *source code* into *machine code* (i.e., zeroes and ones). Execute the command below to do just that:

```
clang hello.c
```

And then execute this one again:

```
ls
```

This time, you should see not only `hello.c` but `a.out` listed as well? (You can see the same graphically if you click that folder icon again.) That's because `clang` has translated the source code in `hello.c` into machine code in `a.out`, which happens to stand for "assembler output," but more on that another time.

Now run the program by executing the below.

```
./a.out
```

Hello, world, indeed!

Next

Naming Programs

Now, `a.out` isn't the most user-friendly name for a program. Let's compile `hello.c` again, this time saving the machine code in a file called, more aptly, `hello`. Execute the below.

```
clang -o hello hello.c
```

Take care not to overlook any of those spaces therein! Then execute this one again:

```
ls
```

You should now see not only `hello.c` (and `a.out` from before) but also `hello` listed as well? That's because `-o` is a *command-line argument*, sometimes known as a *flag* or a *switch*, that tells `clang` to output (hence the `o`) a file called `hello`. Execute the below to try out the newly named program.

```
./hello
```

Hello there again!

Next

Making Things Easier

Recall that we can automate the process of executing `clang`, letting `make` figure out how to do so for us, thereby saving us some keystrokes. Execute the below to compile this program one last time.

```
make hello
```

You should see that `make` executes `clang` with even more command-line arguments for you? More on those, too, another time!

Now execute the program itself one last time by executing the below.

```
./hello
```

Phew!

Getting User Input

Suffice it to say, no matter how you compile or execute this program, it only ever prints `hello, world`. Let's personalize it a bit, just as we did in class.
Modify this program in such a way that it first prompts the user for their name and then prints `hello, so-and-so`, where `so-and-so` is their actual name.

As before, be sure to compile your program with:

```
make hello
```

And be sure to execute your program, testing it a few times with different inputs, with:

```
./hello
```

Staff's Solution

To try out the staff's implementation of this problem, execute

```
./hello
```

within this sandbox.

Hints

Don't recall how to prompt the user for their name?

Recall that you can use `get_string` as follows, storing its *return value* in a variable called `name` of type `string`.

```
string name = get_string("What is your name?\n");
```

Don't recall how to format a string?

Don’t recall how to join (i.e., concatenate) the user’s name with a greeting? Recall that you can use `printf` not only to print but to format a string (hence, the `f` in `printf`), a la the below, wherein `name` is a `string`.

```
printf("hello, %s\n", name);
```

Use of undeclared identifier?

Seeing the below, perhaps atop other errors?

```
error: use of undeclared identifier 'string'; did you mean 'stdin'?
```

Recall that, to use `get_string`, you need to include `cs50.h` (in which `get_string` is *declared*) atop a file, as with:

```
#include <cs50.h>
```

Next

How to Submit

For this and all subsequent problems, you’ll be using a built-in tool called `submit50` to submit your work. No need to download your code and manually upload to GitHub as you did in Problem Set 0!

Instead, execute the below in the terminal window, logging in with your GitHub username and password when prompted. For security, you’ll see asterisks (`*`) instead of the actual characters in your password.

```
submit50 cs50/problems/2019/x/hello
```

You can then go to https://cs50.me/cs50x to view your current scores!