

TASK 1

Let's learn some C!

History

A little history lesson first. Back in the 1970s, computation was really taking off. The Von Neumann architecture was generally complete, the Apollo missions successfully landed on the moon, a little device called a “microprocessor” was offering untold potential in future computing environments, and the power of the computer was quickly becoming evident in applications society-wide. With more demand for computation, many companies began to invest concerted efforts into the creation more robust and generalized operating systems.

At the time, a small little company called Bell Labs, owner of Bell Telephone, the biggest phone provider in the world (Yes, I was being sarcastic) decided they needed to create an operating system that would effectively handle their telecommunications switching and operations. This effort led to the creation of a Kernel Operating system called UNIX. Because of the demands required to create such a modern OS, a new programming language was needed. To do this, developers at Bell created the C programming language. After creating the syntactical requirements, the compilers, and the overall lingual architecture, they proceeded to use C to build UNIX from the ground up.

Picking up the torch from Bell, two more companies, Apple and Microsoft, decided to opt in with C as well. Today *all three* primary operating systems in use in most computers use the C

programming language. It is this overwhelmingly *dominant* use of C that necessitates that all OS students learn and understand C at a fundamental level.

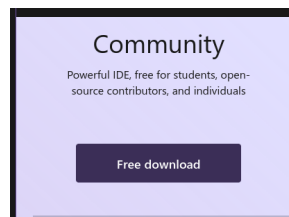
Interestingly enough, the very same UNIX core implemented by Bell in the ‘70s is still at the heart of both Mac and Linux operating systems. DOS diverged significantly in architecture, though it still uses the C programming language. Because both Mac and Linux still offer UNIX kernel access, writing, compiling and executing C code on both Mac and Linux is effortless. Windows can’t say the same. Writing C code on a Windows machine is difficult and can be frustrating at times.

So, assuming you use a Windows Machine, let’s get you setup.

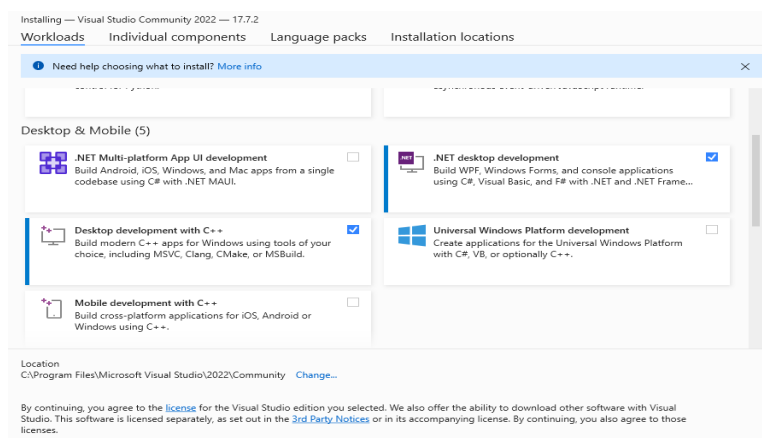
Visual Studio

Let’s start by getting Visual Studio setup. Visual Studio is the easiest Windows solution for C programming on Windows Machines.

1. To install Visual Studio, visit this link: <https://visualstudio.microsoft.com/downloads/>
2. Click the “Free Download” button under Community and wait for the download to finish.

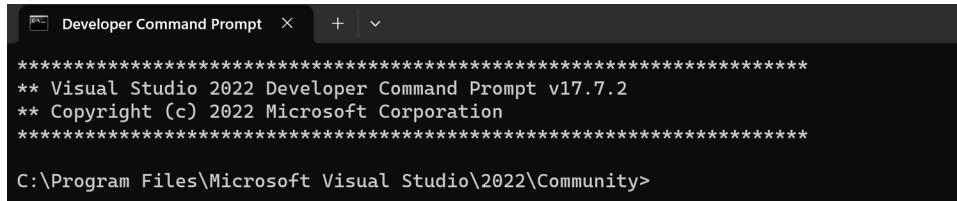


3. Run the installer and follow the wizard prompts.
4. When prompted to choose Workloads, check the **Desktop development with C++** option



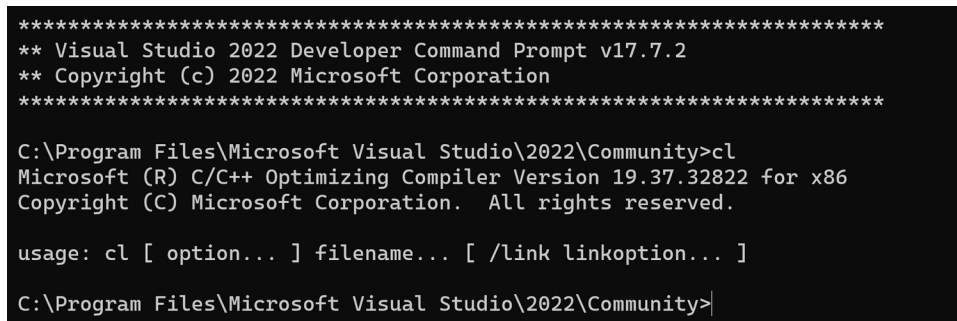
5. Click Install
6. When Installation Completes, there’s no need to open Visual Studio. Close all VS windows.

7. At the start menu, search **Developer Command Prompt** and open the program.



```
Developer Command Prompt x + v
*****
** Visual Studio 2022 Developer Command Prompt v17.7.2
** Copyright (c) 2022 Microsoft Corporation
*****
C:\Program Files\Microsoft Visual Studio\2022\Community>
```

8. When the program opens, type **cl** and hit Enter (C as in Craig, L as in Larry). You should see the output below.



```
*****
** Visual Studio 2022 Developer Command Prompt v17.7.2
** Copyright (c) 2022 Microsoft Corporation
*****
C:\Program Files\Microsoft Visual Studio\2022\Community>cl
Microsoft (R) C/C++ Optimizing Compiler Version 19.37.32822 for x86
Copyright (C) Microsoft Corporation. All rights reserved.

usage: cl [ option... ] filename... [ /link linkoption... ]

C:\Program Files\Microsoft Visual Studio\2022\Community>
```

TASK 2

Next, we need to setup a directory into which we can place all of our C source files.

Before we move on, let's be sure we understand some basic Windows CLI commands.

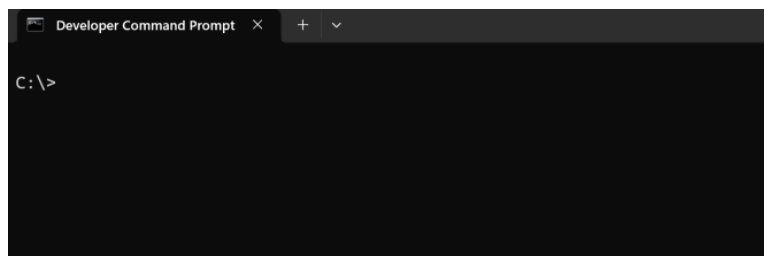
cd : Stands for Change Directory. It is used to navigate around the file system. Use cd {Directory Path} to move a directory, cd .. to move up one level, cd ../.. to move up two levels, etc.

dir : Stands for Directory. When executed, the command will show the contents of the current working directory.

mkdir : Stands for Make Directory. Will create a directory when called with a name.

Great! Let's do this.

1. First, in the **Developer Command Prompt**, let's jump into the root directory of your machine by typing: **cd C:** You should see a screen that looks like this.



```
Developer Command Prompt x + v
C:\>
```

- Next, type **dir** to see the contents of your root folder and look at the command results.

```
Developer Command Prompt x + v
C:\>dir
Volume in drive C is Windows
Volume Serial Number is 8AF8-184C

Directory of C:\

05/12/2023  04:28 AM          112,080 appverifUI.dll
08/22/2023  01:32 PM        <DIR>      Intel
05/07/2022  12:24 AM        <DIR>      PerfLogs
08/23/2023  10:21 AM        <DIR>      Program Files
08/23/2023  10:21 AM        <DIR>      Program Files (x86)
08/15/2023  02:34 PM        <DIR>      Users
05/12/2023  04:29 AM          66,160 vfcompat.dll
08/23/2023  08:23 AM        <DIR>      Windows
                2 File(s)      178,240 bytes
                6 Dir(s)    930,384,674,816 bytes free

C:\>
```

- To continue, we're going to navigate to the Desktop. The Desktop path should be: **C:\Users\{username}\Desktop**. To get there type: **cd Users\{username}\Desktop**.

```
C:\>cd Users\spanieram\Desktop
C:\Users\spanieram\Desktop>
```

- Once at the Desktop, let's make a lab folder. Type: **mkdir 401_Labs**

```
C:\>cd Users\spanieram\Desktop
C:\Users\spanieram\Desktop>mkdir 401_Labs
```

- Just to be sure everything worked, let's type **dir**. You should see a new directory.

```
C:\Users\spanieram\Desktop>dir
Volume in drive C is Windows
Volume Serial Number is 8AF8-184C

Directory of C:\Users\spanieram\Desktop

08/23/2023  11:21 AM        <DIR>      .
08/23/2023  11:20 AM        <DIR>      ..
08/22/2023  08:09 PM          <DIR>      150
08/22/2023  08:11 PM          <DIR>      158
08/18/2023  08:25 AM          <DIR>      235
08/16/2023  05:28 PM          <DIR>      401
08/23/2023  11:23 AM          <DIR>      401_Labs
08/23/2023  11:07 AM          2,208 Atom.lnk
08/22/2023  01:28 PM          1,252 PyCharm Community Edition 2023.2.lnk
                2 File(s)      3,460 bytes
                7 Dir(s)    930,396,827,648 bytes free

C:\Users\spanieram\Desktop>
```

- Now that we have a place to put our work, let's jump in. Type: **cd 401_Labs**

```
C:\Users\spanieram\Desktop>cd 401_Labs
C:\Users\spanieram\Desktop\401_Labs>
```

TASK 2.2

This one is for all my awesome Mac users out there. Let's start by getting a gcc installer onto your machine.

1. Open a new Terminal Window: Cmd + Space --> Type Terminal --> Hit Return
2. Let's get the system ready to install command line tools. Run: **xcode-select --install**
3. Run the following command to install Homebrew:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

4. Run: **brew install gcc**
5. To check your GCC install, run: **gcc --version**

TASK 3

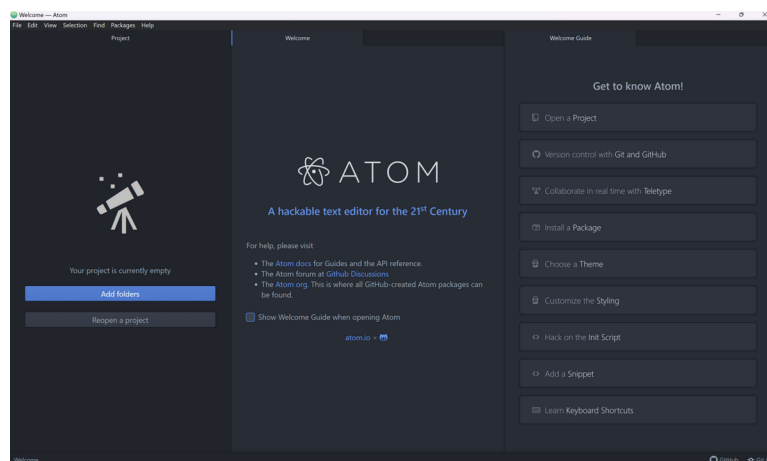
Alrighty, we now have a Command Line Interface that can actually compile C. Fantastic! Next up, let's install an IDE that can help us write our C source code. To do this, we'll use Atom. Atom is an open source IDE that can be modified for individual needs. It offers text coloring, highlighting, and other useful functions when coding in multiple coding languages.

1. Navigate to <https://github.com/atom/atom/releases/tag/v1.60.0>
2. FOR PC:
 - Scroll down and *left* click "AtomSetup-x64.exe"
 - After the download is complete, run the setup. (Choose Dark Mode. It's the best).

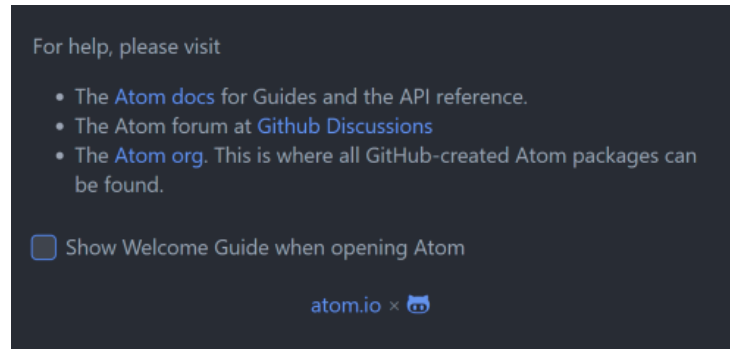
FOR MAC (I haven't tested this... cuz I don't have a Mac. You'll be my Guinea Pigs):

- Download "atom-mac.zip"
- Run: **sudo Atom.app/Contents/MacOS/Atom**
- Install shell commands to run globally: With Atom running, go to **Atom** at upper left next to the Apple --> Click *Install Shell Commands*
- Run Atom by typing **Atom** at the command line

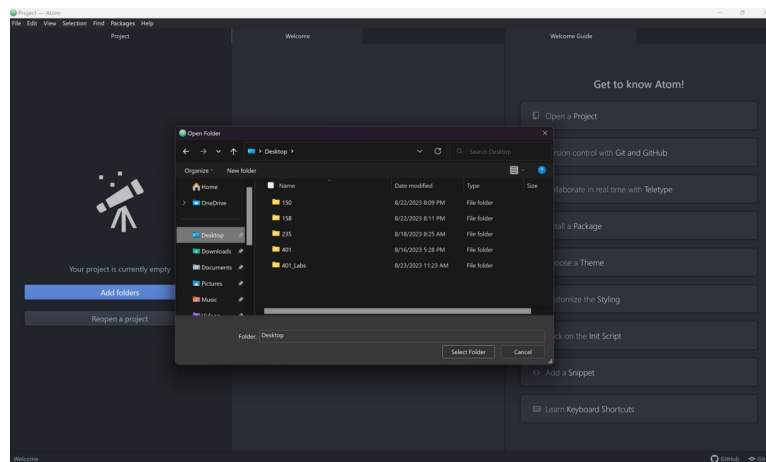
3. When done, open Atom. You should see something like this:



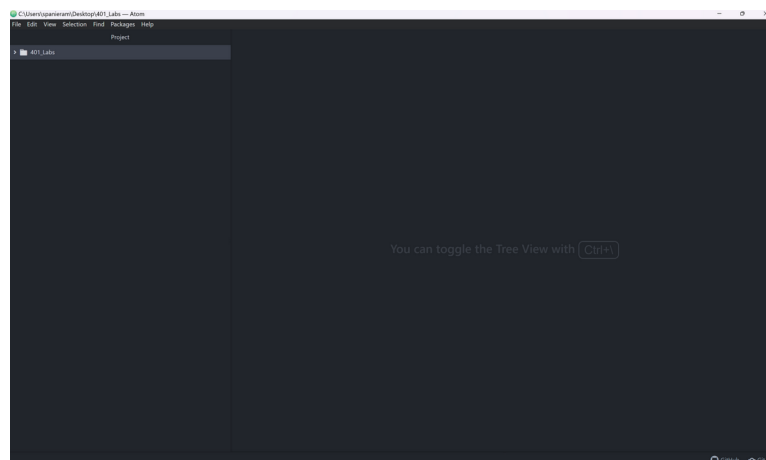
- Let's get rid of the pesky welcome guide when we open Atom. To do this uncheck "Show Welcome Guide when opening Atom" on the "Welcome Tab"



- Click "Add Folders". Navigate to your Desktop and select 401_Labs and hit "Select Folder"



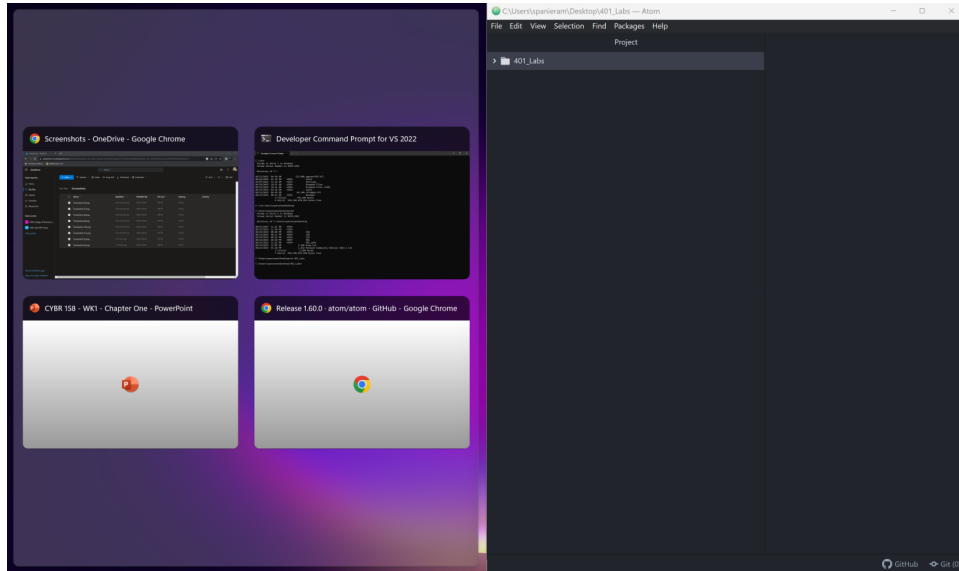
- Next, click the small x to close the "Welcome" and "Welcome Guide" Tabs. We won't need them.
- At this point, you should have an IDE that looks something like this.



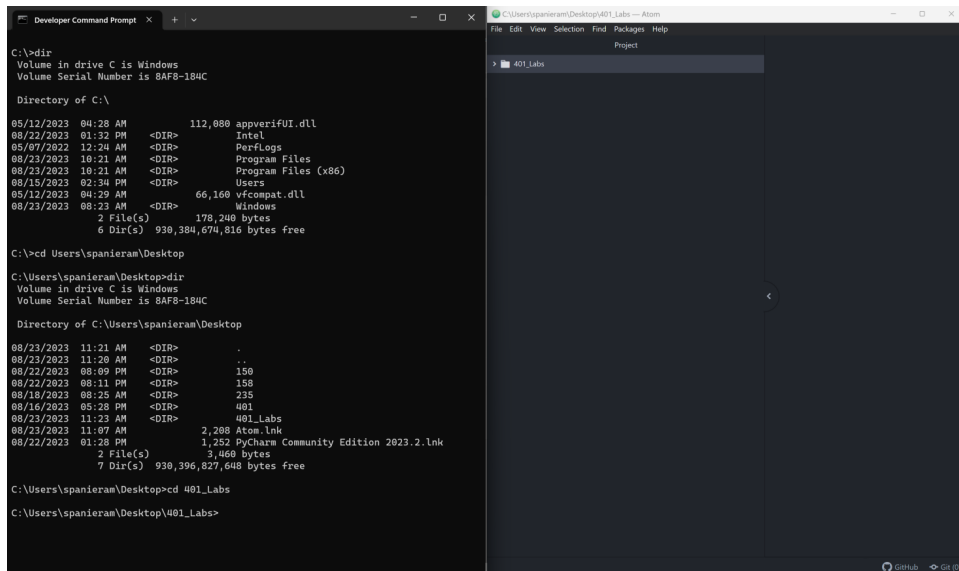
TASK 4

Really quick. Let's arrange our workspace for C development.

1. Click and drag the Atom editor to the right side of your screen. It should pop into a half-screen window and allow you to choose what should occupy the opposite side of the screen.



2. Choose the **Developer Command Prompt** window. It should look like this:



3. This setup will allow you to write code in your editor and compile and run code in your Console.

TASK 5

Let's write some C!

First, in your Atom Project Pane, right click the "401_Labs" directory and choose "new file". Title the file *lab1.c*. The new file should appear in a new Atom Tab.

To get our C programs to work, we're going to need to **include** a few prebuilt packages from the C Standard Library. Where Python uses packages to define helper and utility functions, C uses what are called *Header Files*. Header functions are denoted by a **.h** postfix. These header files provide useful functions like input, output, and memory allocations.

The first header we'll use is the *Standard Input/Output* library called *stdio.h*. To pull this in we'll write:

```
1  #include <stdio.h>
```

This header file provides standard input and output functions like `printf` and `scanf`.

Before we can test our code, C requires that a *main()* method is present in every executable C program. So, let's write one.

```
3  int main() {  
4  
5  }
```

To test our program, let's go ahead and tell the world "Hello!". Let's use the *printf()* function pulled in from the *standard input and output* header file. Just for our own benefit, let's also add a newline. Here's the complete file.

```
lab1.c  
1  #include <stdio.h>  
2  
3  int main() {  
4      printf("Hello!\n");  
5  }
```

Save the file so we can compile it.

Congratulations! You've officially written code in C! Let's run it!

To do this, you'll need to navigate into the directory containing your *lab1.c* file from either your **Developer Command Prompt (DCP)** window OR your **Terminal** window.

- For DCP:
 - Run: **cd lab1.c**
 - Run: **lab1.exe**


```
Developer Command Prompt x + v
C:\Users\spanieram\Desktop\401_Labs>cl lab1.c
Microsoft (R) C/C++ Optimizing Compiler Version 19.37.32822 for x86
Copyright (C) Microsoft Corporation. All rights reserved.

lab1.c
Microsoft (R) Incremental Linker Version 14.37.32822.0
Copyright (C) Microsoft Corporation. All rights reserved.

/out:lab1.exe
lab1.obj

C:\Users\spanieram\Desktop\401_Labs>lab1.exe
Hello!
C:\Users\spanieram\Desktop\401_Labs>
```

- For Terminal
 - Run: gcc lab1.c -o lab1
 - Run: ./lab1

```
aspanier@comeback-kid:~/Desktop/401_Labs$ gcc lab1.c -o lab1
aspanier@comeback-kid:~/Desktop/401_Labs$ ./lab1
Hello!
aspanier@comeback-kid:~/Desktop/401_Labs$
```

TASK 6

Let's go a bit deeper... In C we can build functions just like any programming language. So let's do that. C function headers are comprised of a return type, a title, and a set of input parameters. Let's look at the main method and see what we have.

int : This is the return type. Anything returned from our *main()* method will be an integer in this case. If we think for a second about what this implies, we now have a mistake in our *main()* method... What might that be?

main : This is the function title. This needs to be in camelcase and should adequately describe the function.

() : These parenthesis hold the input args. In this case we have no input args.

Before we go much further, it's important to note that C compiles from the top to the bottom. This means that a method called must be defined *above* it. For instance, if I want to call a function *foo()* from the *main()* method, *foo()* must be defined before the *main()* method is.

In C, it is customary to write the *main()* method at the top of the file. This puts us in a conundrum... If we need to define functions before they are called in the *main()* method, but we make sure the *main()* method is at the top, it seems we're at an impasse...

Enter in *function prototypes*. Function prototypes are nothing more than a simple version of the function header sketched out without any function coding. To understand how this works, let's write a simple function called *printTen()*.

1. First, let's define the function prototype. We do this by defining the output type, the name, and the input parameters. For this it will return nothing and take nothing, so we'll use void and empty parens. Here's a code now with the function prototype included.

```
1  #include <stdio.h>
2
3  void printTen();
4
5  int main() {
6      printf("Hello!\n");
7  }
```

2. Let's compile (using `cl` or `gcc`) and run (using `.exe` or `./`)
3. You should see it compiles with no problem and runs the same "Hello!" script. This is because we've simply told C that a function called *printTen()* is incoming. The C compiler recognizes that and puts a pin in it. Since there is not further definition, it assumes that the prototype is enough. Let's go ahead and flush it out.

We'll start by providing the output type, the name, and the input parentheses. We'll then enclose the function in `{}`. In the function we'll simply print the number 10 and return. To do this, we'll need to dive a bit deeper into *printf()*.

To print an integer in C using the *printf()* function, we'll need to first provide an input string, then the value we want to print. `%d` indicates we'll want to print an integer. `%f` indicates we want to print a float. `%s` indicates an array of characters. Here's what our code should look like:

```
1  #include <stdio.h>
2
3  void printTen();
4
5  int main() {
6      printf("Hello!\n");
7  }
8
9  void printTen() {
10     printf("%d\n", 10);
11 }
```

4. Now we just need to call the function from the main method!

```
1  #include <stdio.h>
2
3  void printTen();
4
5  int main() {
6      printf("Hello!\n");
7      printTen();
8  }
9
10 void printTen() {
11     printf("%d\n", 10);
12 }
```

5. The last thing we'll do is recompile and run the program. You should see this as your output.

```
aspanier@comeback-kid:~/Desktop/401_Labs$ gcc lab1.c -o lab1
aspanier@comeback-kid:~/Desktop/401_Labs$ ./lab1
Hello!
10
aspanier@comeback-kid:~/Desktop/401_Labs$
```

TASK 7

Save your work and submit your .c source file on Canvas!

Please spend some time writing some C functions and code. Mess around a bit with just how you set everything up and run things. Next week, the lab will be much more difficult!