UNIVERSITY OF CALIFORNIA, MERCED

BERKELEY · DAVIS · IRVINE · LOS ANGELES · MERCED · RIVERSIDE · SAN DIEGO · SAN FRANCISCO

SANTA BARBARA · SANTA CRUZ

CSE 30: Data Structures
Laboratory 1

Fall 2019

1 Introduction

In this lab, we will be setting up your workspace for the rest of the semester. This involves installing all the compilers, tools, and libraries that you will need in order to work with the programming examples and exercises that will be provided in the course.

This document contains instructions specific to operating systems. The support code you will be provided with has been tested on computers running Windows[1] 10, macOS 10.13 or later, and Ubuntu 18.04 and some of its variants.

The rest of this document will guide you through the process of setting up your laptop, and compiling and running the example code, which is a simple web server written in C++, that serves the API for an application that can add two integers.

2 Computer Setup

Please follow the guide relevant to you, based on the operating system installed on your laptop.

2.1 Windows

The following has been tested and verified to work on Windows 10. If you have another version of Windows, you may be able to follow a similar set of steps.

1. **Enable Virtualization** This is a crucial step, as it is needed by Ubuntu Shell, and it is perhaps the most difficult, as the process is slightly different for each laptop manufacturer. First, check to see if virtualization is enabled. To do this, open Task Manager → Performance, and see if Virtualization is Enabled or Disabled. You should see an interface similar to Fig. 1.

   If virtualization is enabled on your computer, you can proceed to the next step. Otherwise, you need to enable virtualization in your computer's BIOS. To enter the BIOS, on most computers, you should press F1 during bootup (immediately after powering on). It may be one of the other F keys, so you should check the exact instructions for your own laptop. We have tested with a Lenovo ideapad, and it is F1, on some machines it is F12. You can also find it by trial and error. Once you enter the BIOS setup, look for a configuration option called Intel Virtual Technology, and set it to Enabled. Save changes and go boot into Windows.

[1] In Windows, we will be using Ubuntu Shell, so we will not exactly build native Windows applications, but things will work.

Please be careful when fiddling in the BIOS setup. While this is unlikely, it is possible for you to damage your computer if you do something wrong. If you are uncomfortable with this step, please ask your Teaching Assistant to help you.

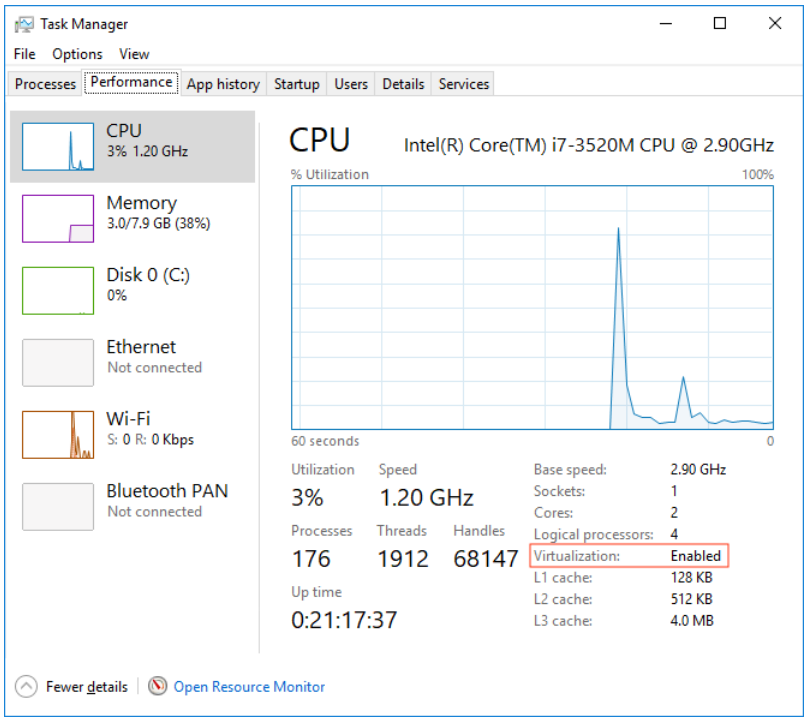

Figure 1: Task Manager window, highlighting virtualization is enabled.

2. **Enable Windows Subsystem for Linux** This is another pre-requisite for Ubuntu to run in your Windows environment. To do this, go to Control Panel → Programs → Turn Windows Features On Or Off, and select the option Windows Subsystem for Linux. This is shown in Fig. 2. Click OK, and you will be prompted to reboot your computer. This feature will not take effect until you reboot, so do it at this time.

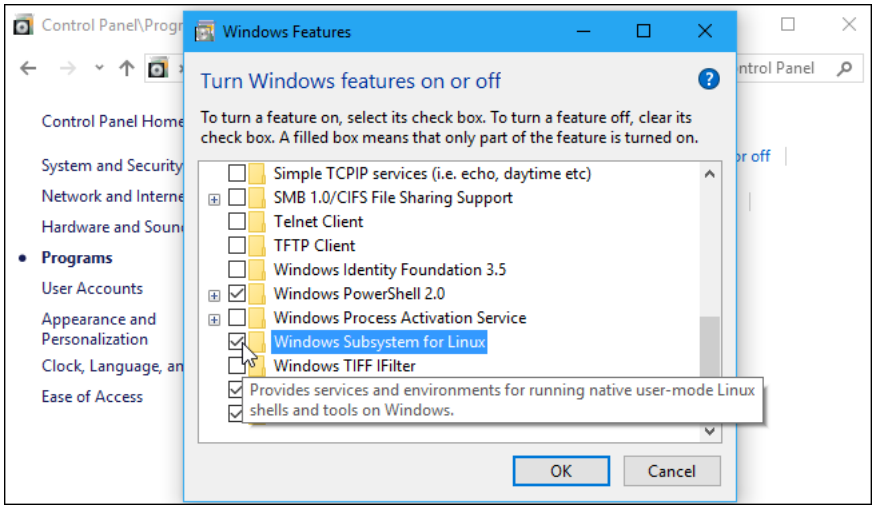

Figure 2: Enabling the Windows Subsystem for Linux

3. **Install Ubuntu** Go to the Microsoft Store and search for "Ununtu". You will see several results, one of which is Ununtu 18.04 LTS. Please install that one, following all the on-screen instructions and prompts. Once finished, you will be asked to Launch the application. The first time you do this, the system will install some additional components. You will see a terminal window, with a message: "Installing, this may take a few minutes...".

   In a little while, you will be prompted to enter an new UNIX username. Please input the username you wish to use for your Ubuntu subsystem. It does not need to be the same as your Windows username. Also, it needs to be all lowercase and contain no spaces. Alphanumeric characters only. You will then be asked to choose a new UNIX password. Again, it does not need to be the same as your Windows password. Please choose a password that you will remember, because you will need to use it a lot. Also note that when you type in the password, there will be no characters appearing in the terminal window, so for those of you who have not experienced this, it will look like you are not typing at all. Do not worry, as this is perfectly normal in Linux. Just type the password and press enter. Then you will be asked to retype the password, so do that and press enter again. You will then be given a message that the installation has successfully completed.

4. **Installing Dependencies** We will assume that you have already installed the Visual Studio Code text editor. If you have not, please install it by downloading from Microsoft's website, and following their instructions.

   In addition to VS Code, we need to install some compilers and other development tools. We will be installing these from within your Ubuntu shell. There is a way to install all these in one step, so in the Ubuntu terminal type in `sudo apt update`, and press enter. You will be asked for your password, which you should type in and press enter. After a short update process you will be ready to install the developer tools. To do that, type

   ```
   sudo apt install build-essential
   ```

   After displaying some information on the terminal, the program will ask you if you want to continue, with a message `Do you want to continue? [Y/n]`. Just press enter there, as the option for "Yes" is the default option, since the letter `Y` is capitalized. This will take some time but it will show you its progress so just wait for it to complete.

   The next thing we need to install is the boost library, which is a collection of algorithms for C++ that lets us do many things for which there is no built-in support. To install the boost library type:

   ```
   sudo apt install libboost-all-dev
   ```

   As before, press enter when asked if you want to continue, and just wait for it to complete. At some point, you may be asked if the installation script can "restart services during package upgrades without asking", you should say "Yes" to that.

   In order to be able to launch Visual Studio from the Ubuntu terminal, just type `code`. The first time you do it, it will install some additional components and will launch Visual Studio Code in Windows. When it opens, make sure its terminal is enabled. If there is no terminal on the bottom right of Visual Studio Code, press Ctrl+`(the key immediately to the left of 1), to toggle the terminal view. Otherwise, use the main menu View → Terminal. The first time you do this, you may have defaulted to the Windows powershell. This is not where we need to be, as we need to use bash. To fix this issue, just type `bash` and press enter. You will switch to bash. To make the change permanent, click on the drop-down which says "1: bash", and click on the option "Select Default Shell". A list appears at the top of the window, and one of the options is "WSL Bash", which is the one you need to select.

   That's all folks.

2.2 macOS

The following will work on any recent version of macOS. It has been tested as far back as High Sierra v10.13.

1. **Install Xcode** You can download and install Xcode from the Mac App Store for free. The process is quite long it installs all the compilers and tools that you will need. Once the installation is complete, open a terminal and type the following:

   ```
   xcode-select --install
   ```

   This will give you all the command line tools you need.

2. **Get Homebrew** In Ubuntu, we have the `apt` utility, which makes it easy to install packages. macOS does not have this utility out of the box but it can easily be installed. Go to the Homebrew website: `https://brew.sh` and follow the installation instructions. It will involve copy-pasting a `curl` command in the terminal and following the prompts.

3. **Installing Dependencies** The only dependency is the boost library, so open a terminal and type:

   ```
   brew install boost
   ```

   We also assume that you have Visual Studio Code, which you downloaded from Microsoft's website. Make sure the Visual Studio Code.app is in your Applications folder, and you are not running it from within the .dmg image that it came in. This is important, so make sure it's done correctly.

2.3 Ubuntu

On Ubuntu the hardest part is actually installing Visual Studio Code. The rest of the stuff is a piece of cake. If you are running Ubuntu (or some other distro) natively on your laptop, then you probably know all this but here it is anyway.

1. **Installing Visual Studio Code** Open a terminal type the following commands:

   ```
   curl https://packages.microsoft.com/keys/microsoft.asc |
      gpg --dearmor > packages.microsoft.gpg
   sudo install -o root -g root -m 644 packages.microsoft.gpg /usr/share/keyrings/
   sudo sh -c 'echo "deb [arch=amd64 signed-by=/usr/share/keyrings/packages.microsoft.gpg]
      https://packages.microsoft.com/repos/vscode stable main" >
      /etc/apt/sources.list.d/vscode.list'
   ```

   The commands above basically add a repository to the apt client on your computer so you can just use the commands below to install:

   ```
   sudo apt-get install apt-transport-https
   sudo apt-get update
   sudo apt-get install code
   ```

   There may be other ways to install, so if you're not having luck with this, you can look for alternatives.

2. **Installing Dependencies**

```
sudo apt install build-essential
```

After displaying some information on the terminal, the program will ask you if you want to continue, with a message `Do you want to continue? [Y/n]`. Just press enter there, as the option for "Yes" is the default option, since the letter `Y` is capitalized. This will take some time but it will show you its progress so just wait for it to complete.

The next thing we need to install is the boost library, which is a collection of algorithms for C++ that lets us do many things for which there is no built-in support. To install the boost library type:

```
sudo apt install libboost-all-dev
```

As before, press enter when asked if you want to continue, and just wait for it to complete. At some point, you may be asked if the installation script can "restart services during package upgrades without asking", you should say "Yes" to that.

3 Running the Example

At this point, you should have a copy of Visual Studio Code, configured correctly (Windows only), and have the boost library installed, which means you are ready to compile and run the example. Download the file WebAppSample.zip and extract it somewhere in your workspace. You should have a folder named WebAppSample, and it it there should be a bunch of folders, like bin, inc, obj, src, static, templates, and then two more files, namely Dockerfile, and Makefile.

From Visual Studio Code, select File → Open Folder, and choose the WebAppSample folder that you just unzipped. The terminal in Visual Studio Code should be at the folder you just opened, which is the `WebAppSample` folder. If for whatever reason you are not at that folder, simply use a series of `cd` commands in order to navigate to the right place. If you are experiencing difficulties with that, your Teaching Assistant will be ale to help you.

Once you have navigated to the folder, simply type:

```
make
```

and press enter. Provided all the dependencies above have been installed correctly, this should build the example and produce an executable called `server` in the `bin` folder. To run this executable, type:

```
./bin/server -t templates -s static
```

If everything has been done correctly, this should start running the server, which serves a web page for a simple application that allows the user to enter two numbers and add them together.

To see this app, go to a web browser and visit the address `127.0.0.1:18080`. You should see something that resembles the interface in Fig. 3.

Feel free to try the application. Try adding the numbers 5 and 3, and make sure it displays the correct answer of 8.

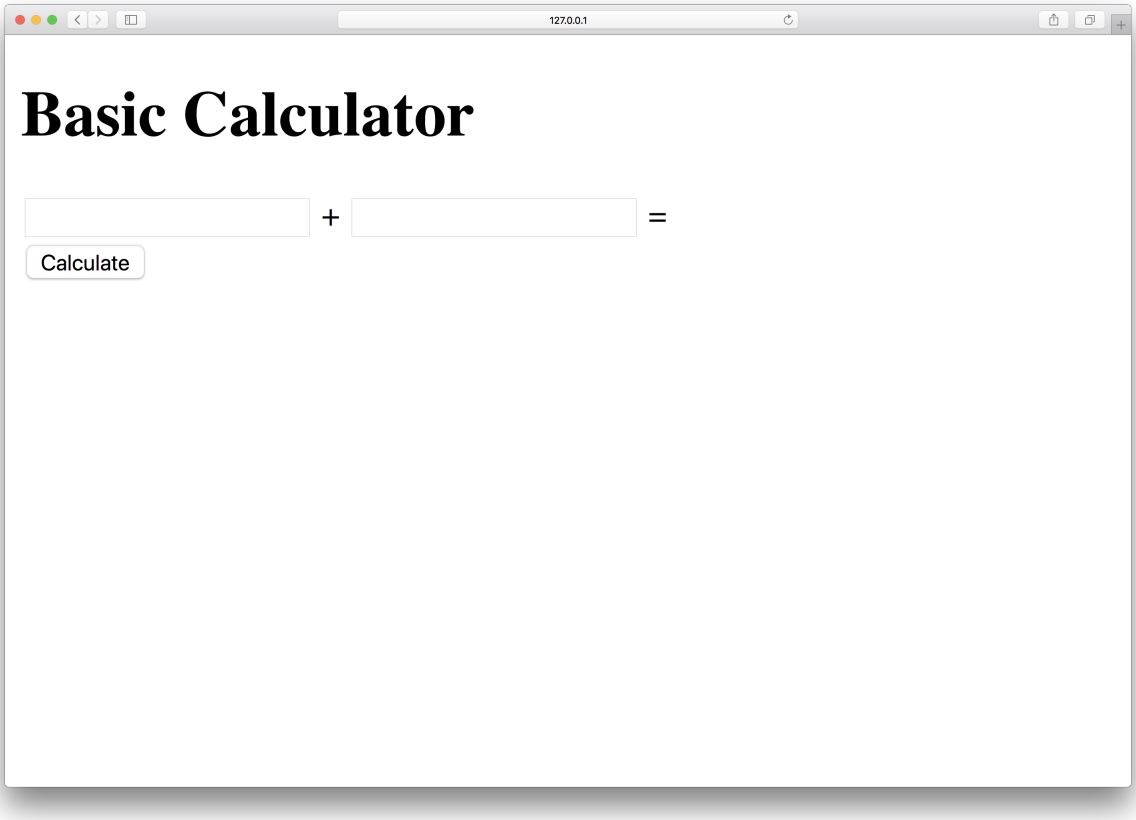

Figure 3: The user interface of the web application

4 Understanding the Example

4.1 The Web Interface

The web interface seen in Fig. 3 is a simple example of HTML and JavaScript. The source code for the HTML is found under the `templates` folder in a file called `index.html`. As some of you may know, HTML is a markup language, so it can be used to format the layout and appearance of the web page. Indeed, if you explore the `index.html` file, you will see, in the `<body>` section, there are elements for the heading (`<h1>` tag), for the input fields (`<input>` tags), and for the button (`<button>` tag). This is basically all that HTML can do. With the help of CSS, the interface specified here can be made to look very aesthetically pleasing, but this is a topic for another time.

4.2 The Logic of the App

Since HTML is used only to format the layout and the appearance of the app, there must be another way to tell the app how to behave. As many of you have heard, the JavaScript language is used for that. So what is the logic of our application? If you explore the file `main.js` in the `static` folder, you'll see that there is a function called `clickCalculate` and it contains some code that may look unfamiliar or strange at first. Do not worry about it at this stage. The only thing we need to do is figure out when the function `clickCalculate` is actually executed. If you look back at `index.html`, on line 16, where the `<button>` tag is defined, you'll see that the function `clickCalculate` is executed whenever the button is clicked. We say that the function is a handler for the click event of the button.

By using the app, in is obvious that the task it is performing is to add numbers. Yet, if you study the code in the `clickCalculate` function, you'll see that there is no code for performing addition of numbers. We could have easily implemented the procedure of adding two integers in `clickCalculate`, using the JavaScript language, but we will not be doing it that way. That kind of intelligence does not belong on the frontend of your application. That is, we do not want the user interface to have any smarts, or any functionality. Why might we want that? It is because, if we decide to change the user interface, and scrap the current one, we will have to re-implement all the smart functionality it had. If it has no smart functionality, then we do not run into that problem. Also, since the frontend runs in a client's browser, it may not have access to data it needs in order to implement its logic. It is for these reasons that user interfaces are always kind of "dumb" (the opposite of smart).

If you study the code in `clickCalculate`, you'll notice that it performs the following tasks. It reads the values that the user has entered in the text fields on the page, lines 3 and 4. It then prepares to send a special request, called an *XML HTTP Request*, or an *AJAX* request to a server, line 6. It then specifies that when the request is answered, it will take the answer (response) it receives and will display it on the interface in a pre-designated spot, right after the equals sign. This happens in lines 8-16. It then prepares to send a request to the `add` url on the server, along with the values of `x`, and `y` that were read earlier, line 17. Finally, the request is sent on lines 18 and 19. It is of crucial importance that you understand how lines 8 to 16 are executed **after** lines 17 to 19.

Do not worry if this does not make sense to you yet. We will be revisiting it many times over. For now all you need to understand is that the HTML page does not perform the addition of the integers. It simply gets the two values the user has entered, and asks the server, or the backend, what their sum is. Once the server sends back a response, the interface simply displays that response next to the equals sign.

4.3 The Server

The C++ code we have in this example is actually a web server. What this means is that the code always runs in the background (unless you kill it), and it listens on a specific port. In our case that port is 18080 because you are unlikely to have another process installed on your computer that is already listening on that port. Anyway, when someone connects to your computer on that port, our code can respond appropriately. When you type `127.0.0.1:18080` in the address bar of your web browser, you are asking it to connect to your own computer on port 18080, which is exactly where our C++ code is listening. If we study the file `server.cpp` in the `scr` folder, we will be able to see how our server responds when someone connects to it.

The address you typed in you browser earlier, `127.0.0.1:18080` means that we are connecting to the so called root of the server. Line 162 of `server.cpp` specifies what should happen when someone visits the root of our server. Without going into too much detail, the code simply says that we should send the `index.html` page to that visitor. The server does that and the browser displays the HTML page, because that's kind of what browsers do.

In addition to the root, there are other *endpoints* we can hit on the server. One of them is called `/add/<int>/<int>`, defined on line 174 of `server.cpp`. This endpoint (url) is a little bit more special because it requires us to give it two integer values. If your server is still running, which it should be, you can visit the address `127.0.0.1:18080/add/3/5`. What you get can be seen in Fig. 4.

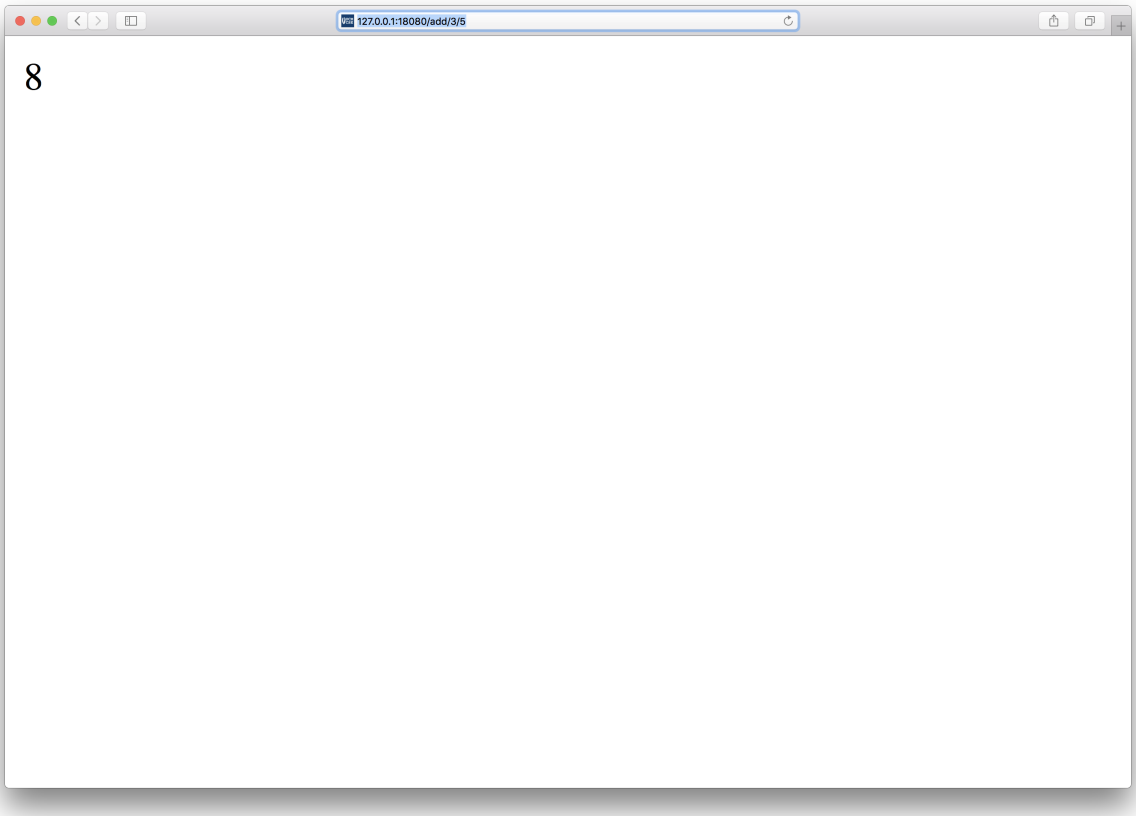

Figure 4: The result of hitting the `/add` endpoint

As you can see that web servers are not only for serving web pages. They can serve information of any kind. What's more is that they can take in some arguments and use them to process some data, and return the

result to us. What we see in this example is exactly that, even though the operation we ask the server to perform is very simple and it could have easily been done on the front end. One does not have to think too long to find an example where the operation is more complicated and can not be done on the front end. Think of the webpage `google.com`. It allows you to enter a search term or phrase, and returns to you a bunch of links to webpages that are relevant to your search query. The frontend of `google.com` does not have that information available to it, so it needs to simply send your search query to the server, wait for a response, and display the response on the page. I am not using Google as an example because I used to work there, I'm using it because it's common knowledge. Our example of adding up the two integers is no different in terms of software design. It is only simpler on the backend.

The function in the JavaScript file `main.js` is essentially visiting the URL `127.0.0.1/add/../..`, getting the response from the server, and using it to update the HTML on the page. So no intelligence on the frontend. It is only an interface and all functionality is implemented on the backend.

4.4 The `add` function

If you look at the code for the `/add` endpoint, you'll see that it actually performs the addition on line 177, by using a function called `add`. Where is this function defined? Its signature appears in `myLib.h`, which is in the `inc` folder, along with some other stuff, and its implementation is in the file `myLib.cpp`, which is in the `src` folder.

Now, this may seem like overkill to some of you, but as discussed in class, all the functions we write for assignments from now on, will be placed in header files, with their own implementation files. The reason for this, other than good organization, is testability. Specifically the ability to write unit tests. If your server is still running, you can visit its root, and try to add some other numbers together. You'll soon realize that whatever number you input, the result is always 8, which coincidentally works for 3 + 5.

4.5 Running the Unit Tests

We know that something is wrong with our program, but instead of rushing to fix the problem by modifying the code, we first run some unit tests. This example is far too easy and we can immediately deduce where the problem is just by trying some inputs, but the idea is that we should have caught it before we even got to the point of running the server and the HTML interface. Unit tests would have caught it on time, but for some reason I neglected to run them before uploading this lab assignment.

To run the unit tests, first go to the terminal where your server is running and kill it by pressing Ctrl+C. In UNIX that kills the current process. Now that you're back at the prompt, type:

```
make test
```

This builds the test suite and places the executable in the `bin` folder. Now run the tests by typing:

```
./bin/test
```

This produces some useful output.

5 Exercises

1. How many test cases did the unit test for the `add` function have? How many of the test cases passed, and how many failed? Of those that failed, how did they fail? In other words describe what output was expected for the test cases that failed, and what output was actually produced. Write your answers in a simple text file and upload it to the relevant CatCourses assignment.

2. Fix the code so that all the tests pass. Compile and run the test suite again, to make sure all test cases pass. Copy the output of the successful test run (showing that all test pass), and paste it in a text file. Upload that text file to the relevant CatCourses assignment.

3. In order to fix the code for the last question, you had to modify one C++ source file. Upload the modified source file after you have fixed the error.

4. (Optional) Create another endpoint, called `/times/<int>/<int>`. It should simply return the product of the two integers supplied. Zip your entire project folder and upload it to the relevant CatCourses assignment.