Introduction

Docker is a convenient way to manage virtual environments as "containers." We will be using it to run `Ubuntu 22.04 (jammy)`, a Linux-based operating system.

Why are we using this? With virtualization, we can specify a standard development environment on any machine, so your code will work no matter where it's run. *(You won't need to log into the department machines to write, test, or hand in your code!)*

Your computer is probably running Mac OS, Windows, or Linux. These different operating systems all come with different libraries and preinstalled software. A Virtual Machine can establish the same software environment for everyone.[1]

In this class, we will use a **container**, a technology that emulates an OS without the full overhead of a VM. The container runs a Linux-based operating system, **Ubuntu**. The autograder also runs a Linux-based OS, so if your code works in the container, it will work on the autograder.

Docker

Docker is one of the most popular container solutions and widely used in industry. In this class, we use Docker because it lets you run a course container on Windows, Mac OS, or Linux. You may download Docker here. On Linux machines, follow the instructions here.

After downloading Docker, follow Docker's instructions to install it on your OS. Accept if Docker asks for privileged access. On Windows or Mac OS, open the Docker Desktop application after it has been installed. You may see a message similar to "Docker Desktop is Starting…". Once this message goes away, your Docker has started successfully!

Verify Docker is installed by executing the following command in a terminal:

```
docker --version
```

A Docker version number should be printed. After installing Docker, a Docker process (the Docker daemon) will run in the background. Run the following command to verify:

```
docker info
```

This should print some information about your Docker installation. If you see the following error:

```
ERROR: Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the
    docker daemon running?
```

it means Docker hasn't started running yet. On Windows or Mac OS, ensure your Docker Desktop is running. On Linux, try the command `sudo systemctl docker restart` in a terminal.

You can follow additional instructions from the Brown University CS300 lab if you are on Windows and do not have WSL (Windows Subsytem for Linux) set up yet.

Setting Up the Dev Environment

In Docker, an environment is defined as a **Docker image**. An image specifies the operating system environment that a container provides. An image can also contain additional software dependencies and configurations. These instructions are specified in a file, the so-called **Dockerfile**. You will now download the course's setup code and create the course Docker image!

[1]Note: Software inside a VM believes it is running on a physical computer, even though it is running within an OS within another computer. Achieving this level of virtualization has costs, including the cost of emulating the underlying hardware.

Mac OS only: Install Apple development tools If you're running on Mac OS, you will need to install a set of Apple-recommended command-line tools via the following command:

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

This ensures that your computer has installed git, a program we'll use later. Alternatively, you may also download and install git directly, following instructions from here.

Do the following to set up your development environment.

1. Enter the directory on your computer where you want to do your coursework. For Windows users, choosing somewhere in the C drive will make the following steps easier. Then, enter the following command to download our development environment to the new subdirectory `DEV-ENVIRONMENT` (you can choose your own name, `DEV-ENVIRONMENT` is a placeholder!):

```
    git clone https://github.com/pomona-cs181ca-po/cs181ca-f25-devenv.git DEV-ENVIRONMENT

```

2. `cd DEV-ENVIRONMENT` to enter the directory you have just created.
3. Inside this folder, do the following: `./docker/cs181ca-setup-docker`. This will attempt to pull our Docker image that has been prebuilt. If you wish to do so, `./docker/cs181ca-build-docker` will build the image from scratch (this can take more than 30 minutes).
4. After this, you can run the Docker image with `./cs181ca-run-docker` which will launch a containerized environment with the appropriate libraries installed. You can now interact with the shell of the container and run `Ctrl-D` to exit.

Shared Folders "If my docker container is a separate (virtual) computer than my laptop, how will I move files between the two?" you may ask. Great question! You'll move files between the two in any of the ways you can move files between any two computers! (Bear with me!)

To get files to and from department machines, you may have used things like scp, sftp, and **Cyberduck**, which allow you to copy files over the network. You can use these tools (or other tools like `git`) to transfer files in and out of a running container!

Text Editors and IDEs

What development environment or editor to use is a question that divides people and has led to countless hours being wasted on heated discussion, offline and online for decades. Some people prefer very simple editors that run in their terminal, while others prefer graphical integrated development environments (IDEs). For this course, if it works for you, great!

To make your life easier, however, you probably want to use an editor with syntax highlighting for C++, the main programming language for this course.

You can edit your source code inside the course container, or on your normal operating system. Recall that the home directory inside the container is a mount of your local cs181ca/home directory (or whatever you named it). This means that you can install any editor or IDE of your choice on your local machine, edit your files and work on your assignments locally, and all of your changes will be reflected in your container! You can then compile, build, and run your assignments inside your container.

There are plenty of great choices for text editors. If you're not sure which one to install, we recommend VSCode[2]. It has very nice integration with our Docker container, and it's nicely customizable with extensions.

[2]Navigate to the extensions tab and search for/install teh "Docker", "Dev Containers" and "WSL" extensions.

Installing Programs on Linux (in your course container)

Our containers come with a few programs pre-installed, but what if we want more? With a text-based interface like the command line, it's quite difficult to navigate the web and find a suitable download link. Instead, we use a program called a package manager which can automatically download, install, and update programs. We'll be using a manager called `apt`. `apt` can install many things, including tools to build (or compile), and debug your code. Our containers already come with these tools for your convenience.

As an example, to install fortune, you would run

```
sudo apt update
sudo apt install fortune
```

Make sure the course container is running, and click the green button at the bottom left of VSCode – you should see an option "Attach to running container" and you can select the course container.