UNIVERSITY OF WASHINGTON
Department of Applied Mathematics

AMATH 483/583 **Assigned:** 3/28/2017
Problem Set #1 **Due:** 4/3/207

1 Preliminaries

1.1 Set up Your Local Workspace

Before beginning with Docker we want to create a folder (directory) that will hold folders (subdirectories) for this and for each of the remaining homework assignments.

For Mac OS X and Windows, we recommend that you create this directory directly under your home directory, and call it amath583work. It is important that this directory have a given name and be in a given location in your workspace, as we will be sharing that directory between the host operating system and Docker — meaning you will be able to edit files in that directory in your host OS and those changes will appear in the Docker container and vice-versa.

You don't need to create this folder in the location we recommend. However, you do need to refer to it correctly when starting up Docker. We will refer to the location as /Users/⟨your-netid⟩/amath583work for Mac OS X or C:\Users\⟨your-netid⟩\amath583work for Windows. If you have this folder in a different location, use that location instead when you start up Docker with the amath583 image.

1.2 Install Docker

For a number of reasons (some of which we discussed in lecture), the platform that supports most high-performance computing today is Linux with its associated terminal-based command-line tools. Since many of you may not yet be familiar with that environment, and since almost all of you have a different system on your day-to-day machines, we are going to provide a standard Linux development environment via Docker (community edition). You will run all of your programming assignments for this course using that image in a Docker container. Since the environment will be the same for everybody (and provided by the AMATH staff), it will be much easier to create programming assignments, assist in diagnosing problems, and in sharing helpful hints with (and among) the class.

Although the environment the class will be running within Docker will be the same for all students, you will be running the Docker application on your own laptop (or other computer) and these will of course vary. Detailed instructions on how to obtain, download, and install Docker can be found on the Docker web site. Some basic steps are provided below, but the instructions on the web site are definitive. Since everything else this and future programming assignments in this course depend on Docker, it is **highly** recommended that, at your earliest opportunity, you install Docker and verify that the amath583/base image works for you.

If your computer and operating system are sufficiently recent, you will be able to run Docker directly. Some less recent machines and operating systems will require running a small virtual machine. Our assumption is that you will be able to run Docker on your laptop or on a machine that you have ready access to. I you have any difficulty in accessing a machine that can run Docker, let one of the teaching assistants for the course know immediately!

Step by step instructions for downloading, installing, and verifying Docker can be found here.

Mac OS X The instructions and installer for Docker for Mac OS X can be found here. Yosemite 10.10.3 or higher is required. Download the version from the "stable" channel. After you have successfully downloaded and installed Docker according to the on-line instructions, verify that the docker icon is in the menubar on your computer and that you get the preferences menu when you click on it. Also verify that the message "Docker is running" is displayed on that menu.

Windows 10 The instructions and installer for Docker for Windows 10 can be found here. Download the version from the "stable" channel. After you have successfully downloaded and installed Docker according to the on-line instructions, verify that the docker icon is in the status bar on your computer and that you get the preferences menu when you click on it. Also verify that the message “Docker is now up and running” is displayed on that menu.

1.3 Verify Your Docker Installation

Steps for verifying Docker are at Step 3 on this page. You should use Terminal.app (“Terminal”) on Mac OS X and PowerShell on Windows.

Hopefully everything about the Docker installation was clear up to this point. However, if not, this is the perfect time to seek (and/or provide) help on Piazza. If you need it going forward, documentation about Docker can be found here.

1.4 Install the AMATH583 Docker Image

In the verification steps on docs.docker.com, you executed some images with the “docker run” command. As mentioned on the site, this first pulled the image to be executed from the docker site and then executed it. For this course we have created a custom image with the tools required to carry out your homework assignments.

To download this image to your local machine, issue the following command in your terminal:
$ docker pull amath583/base
(Note that the $ sign represents the general command-line prompt—you only need to type in what follows the prompt. We will use $ and % as generic command-line prompts in this course.) The image is fairly large (around 2GB) so the download will take some time, depending on the speed of your network connection.

1.5 Connect to Your Local Workspace

There is an important feature to remember about running an image in a Docker container: *Whatever you do in the container is not permanently saved.* If you do your work in the container and save some files in the container, the next time you start the container, those files will be gone.

Fortunately, Docker provides a mechanism to share files between running containers and the host file system (your folders and files). To do that we use a command line option with Docker to map a folder in your home directory (the folder that you were asked to make above) to a directory in the running container.

Now we are ready to run the amath583/base image in a Docker container. Remember the following command (write it down, save it to a note file, etc) as you will be running it every time you need to work on programming assignments for this course.
For Mac OS X:

```
$ docker run -it -v /Users/⟨your-netid⟩/amath583work:/home/amath583/work amath583/base
```

For Windows

```
$ docker run -it -v C:\Users\⟨your-netid⟩\amath583work:/home/amath583/work amath583/base
```

Explanation. The “-it” option in the command tells Docker to run interactively in a terminal. The “-v” option tells Docker to map a local directory (the directory on the left-hand of the “:” to a directory inside of the container (the directory on the right-hand of the “:”. Finally, amath583/base is the image to run.

When Docker has successfully started you should see a command prompt that looks like the following:

```
amath583@892dae99e90b:/$
```

(The numbers in your prompt will be different.)

From this prompt, enter the ”ls” command. You should see:

```
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
```

Now enter the “cd” command. You should see:

```
amath583@892dae99e90b:~$
```

Do you notice the change in the prompt? The "/" in the previous prompt has changed to " ~ ": the cd command without any arguments changes your working directory to your home directory.

To see your home directory, type in "ls" again. You should see

```
tmp  work
```

The "work" directory is the one we mapped to your working directory outside of Docker.

Enter the following commands:

```
$ cd work
$ touch hello
$ ls
```

You should now see:

```
hello
```

Now, to verify that the file system connection is working, go to your working directory (the one outside of Docker). There should now be an empty file there called "hello". From the Finder (Mac OS X) or from the WIndows Explorer, delete the "hello" file. Go back to your Docker terminal and issue the "ls" command. You should now see:

The file is gone.

One last exercise. In Docker, issue the following commands:

```
$ cd
$ touch work/goodbye
$ touch tmp/goodbye
$ ls work
$ ls tmp
```

You should see that you have created a file named "goodbye" in each of "tmp" and "work". Now exit from Docker, you can do that just by entering "exit" at the command prompt. After you have exited Docker, go to your local work directory and verify that the empty file "goodbye" exists there. Rename this file to "hello" (or create some other file). Restart Docker with the "run" command above and issue the following commands:

```
$ cd
$ ls work
$ ls tmp
```

You should see that the changes you made in your local directory are reflected in "work" but that the changes you made to "tmp" are gone. Experiment some more with changing files between your Docker "work" directory and your local (outside of Docker) folder. Make sure you have confidence that you can make changes in one and they will be immediately reflected in the other – and that these changes will persist across restarts of Docker.

2 Warm Up

In the verification steps above we didn't really explain what any of the cryptic commands that you typed in actually did. In this part of the assignment we will explain a bit more about what is going on and give you some ideas about what and how to explore the environment on your own.

2.1 The Shell

The program that runs when you start Docker on the amath583/base image is the "bash" shell. A shell is a program that can execute other programs as well as its own scripts. When run in interactive mode, the shell prints a prompt to the screen and waits for user input. The user enters text and when the user hits return, the shell attempts to execute the command and then again prompts the user for input.

The amath583/base image has documentation available for most of the commands that you will be using. These are generally comprehensive – which is good – but sometimes it can be difficult to locate exactly what you are looking for. In those cases your favorite search engine (and/or Piazza) can be your friend.

Three commands that are essential for navigating a Linux filesystem are "cd", "pwd", and "ls", which change directory, print the working directory, and list the contents of the current directory, respectively. The "ls" command has a number of options that you can use with it to control what is displayed and how. The linux filesystem is basically a tree of directories (folders), each of which can contain files and other directories (sub-directories). The top-level directory is called "/". This is where you start out when amath583/base first starts up. When you issued "ls" in that directory, you listed its contents, which in this case were more directories.

To change to a different directory, use the "cd" (change directory) command. If you issue "cd" without any arguments you will change to your home directory, which is /home/amath583 in your image. If you provide an argument to "cd" the shell will attempt to change your working directory to the indicated path (a path is a sequence of directories, possibly ending in a file). There are two types of paths that you can supply – absolute or relative. An absolute path begins at the root, i.e., "/" while a relative path begins with the name of a subdirectory in the current directory. Finally, you can determine which is the current working directory for the shell by issuing the command "pwd" (print working directory).

For example, if your current working directory is your home directory "/home/amath583" then when you issue the command "pwd" it will print "/home/amath583". The following two commands are equivalent (one is relative, one is absolute):

```
$ cd tmp
$ cd /home/amath583/tmp
```

Note that "ls" can also take arguments, again an absolute or a relative path:

```
$ ls tmp
$ ls /home/amath583/tmp
```

There are two special subdirectories in every directory: "." (dot) and ".." (dotdot). Dot is the name of the current directory. If you issue "ls ." you will get the contents of the current directory. Dotdot is the name of the parent directory (recall that the filesystem is a tree – each directory has a unique parent, up to "/"). Try out "." and ".." with the "ls" and "cd" commands.

2.2 Documentation

The amath583/base image contains documentation for the packages that were installed with it. The typical Linux way of viewing the documentation is the "man" (manual) command. To view documentation of a particular command, invoke man with the name of the command as an argument. For example

```
$ man ls
```

will display the documentation for the "ls" command. Many commands that you issue at the command prompt are executable programs stored in the filesystem (usually in /bin, /usr/bin, or /usr/local/bin). However, some commands are built-in shell commands – "cd" for example is a built-in command. To find out information about built-in shell commands, use "help" (which is also built-in). Note that there is a difference between "man" and "help".

2.3 Explore

Now, take a few minutes to move around the directory structure in the amath583/base image. Some directories to look at might be "/usr/bin" or "/usr/include". Two more commands that you might find handy are "cat" and "more". And remember, except for the shared folder "/home/amath583/work", nothing you do while working in the Docker container is permanent. So don't be reluctant to experiment. If something goes awry, you can exit (or kill) the container and start over.

2.4 Creating and Editing Files

For the rest of this course you will be creating and editing source code files – each assignment will go in a subdirectory of the shared folder ("/home/amath583/work" inside the container) and these subdirectories will contain all of the files you will be using for that assignment. You can work with the subdirectories and the files from either "side" (within the docker container or from your host environment). Which you prefer to use is up to you – but make sure you are comfortable with how the sharing between the container and the host works. The image has some popular Linux-world editors: emacs, vim, and nano – which, though powerful, can have a bit of a learning curve. The nano editor is probably the most straightforward to use immediately. If you have suggestions about another docker-side editor that would be useful, do let us know.

If you plan to edit on the host side, we also recommend downloading and using "atom" – available at "http://atom.io". It is important that on the host side you only work with plain text files. Do not use a word processor such as Microsoft Word. If you want to use a built-in editor on the host side we suggest notepad or textedit (but again, make sure they save your files as a plain text file), but programmer-specific editors have many features to support common programming needs (syntax highlighting, indenting, brace matching) so we strongly recommend "atom" or an equivalent source code editor.

2.5 Hello World!

In your shared working directory, create a sub-directory named "ps1". Within that sub-directory create a file called "hello.cpp" with the following contents:

```
#include <iostream>

int main() {

  std::cout << "Hello World" << std::endl;

  return 0;
}
```

Inside the docker container, and from within the same sub-directory you should see the file "hello.cpp" when you use "ls". Use "cat" or "more" to verify its contents.

Now, execute the following commands:

```
$ c++ hello.cpp
$ ./a.out
```

3 Exercises

The following exercises ask you to do some basic I/O in C++. You do not need to use files for reading or writing – you should read from std::cin and write to std::cout.

All of your files for this assignment should be in a subdirectory of your shared (work) folder. Name this subdirectory "ps1".

3.1 Print a Sequence of Numbers

For this problem you are asked to write a program that will print a consecutive sequence of numbers, starting from 0 and incrementing by 1. The name of your source file for this program should be "N.cpp".

Input. The input will consist of a single line containing a single integer number n. If the value of n is less than zero, your program should not print anything and should return a value of -2.

Output. The output will consist of a header string, the integer n, a sequence of n integers starting at 0 and a trailer string. The header string is "AMATH 583 VECTOR" and the trailer string is "THIS IS THE END".

Example. The following is sample input for this problem.

```
10
```

The following is sample output for this problem.

```
AMATH 583 VECTOR
10
0
1
2
3
4
5
6
7
8
9
THIS IS THE END
```

3.2 Read a Sequence of Numbers into vector<double>

For this problem you are asked to write a program that will read in a sequence of numbers and add them together. The name of the file into which you write this program should be "vsum.cpp".

Input. The input will consist of lines of text, consisting of a header, an integer n, a sequence of n numbers, and a trailer. As above, lhe header string is "AMATH 583 VECTOR" and the trailer string is "THIS IS THE END". The numbers will be in integer or floating point format. Your program should so some basic checks for incorrect input. If either the header or the trailer are incorrect, your program (your main() function) should not print anything and should return a value of -1. If the value of n is less zero your program should not print anything and should return a value of -2. If the value of n is equal to zero your program should print 0 and return 0 (correct execution). For now we can assume all of the input numbers are well formed and that there are a correct number of them.

Output. The output will consist of a single number equal to the sum of the sequence of numbers.

Example. The following is sample input for this problem.

```
AMATH 583 VECTOR
6
0.4
-1.3
```

```
2.141
3.14159
4.0
5
THIS IS THE END
```

The following is sample output for the above input.

```
13.3826
```

Implementation Suggestions. You are free to implement you program in any way that meets the requirements given above. However, in future assignments, this will be one way that we read data into vectors and matrices. You are encouraged to read the data into a std::vector<double> data structure and to use an appropriate standard library function to accumulate (hint) all of the numbers.

3.3 Multi-File Programs (583 only)

For this exercise, we want to factor the vsum program above into separate functions and separate files. The input, output, and error handling are just as before. However, you are asked to create a function named "readVector()" that returns a vector<double> that contains the data that is read. The body for this function should be almost identical to the previous exercise. The difference is that your main() will simply invoke "readVector()" rather than doing the reading itself. Put the "readVector()" function into a separate file from your main() and name that file "vector.cpp", invoke "readVector()" from main() and do the summation in main(). Put your main() function into a file called "vsum2.cpp". Note that you will need to "declare" the "readVector()" function before you can call it from main().

3.4 Written Exercises

Create a text file called ex1.txt in your ps1 folder. In that file, copy the following questions. Start your answers on a new line after each question and leave a blank line between the end of your answer and the next question.

1. What is the complete string in your bash shell prompt when your working directory is /home/amath583/work?
2. What is the indicated host name of your amath583/base container? (You will need to find the right Linux command to execute for this.)
3. How do you specify the name of an output file when you use the compiler?
4. (583 only) Describe how you would separately compile the two files vsum2.cpp and vector.cpp into object files and link them to create an executable.

4 Turning in The Exercises

To turn in your assignment, you will create a compressed tarball to upload to Collect It. For 483 students, use the command

```
$ tar -czf ps1.tgz N.cpp vsum.cpp ex1.txt
```

This calls the "tar" command with flags to indicate what type of compression to use, specifying the creation of "ps1.tgz" that contains compressed versions of the "N.cpp", "vsum.cpp", and "ex1.txt" files. Similarly, 583 students should use the following command:

```
$ tar -czf ps1.tgz N.cpp vsum.cpp ex1.txt vsum2.cpp vector.cpp
```

Before you upload the ”ps1.tgz” file, it is **very important** that you have confidence in your code passing the automated grading scripts (see the course syllabus about regrade request limitations). To this end, you have been provided with a Makefile and a python script (test_ps1.py) on Collect It to test your code for sample inputs. Put both of these files into your ”ps1” directory. 483 students should then use the command ”make tests483” that will unpack your tarball and show diagnostic information to the screen. For 583 students, the command ”make tests583” should be used. Once you are convinced your code is producing the correct output for the given input, upload ”ps1.tgz” to Collect It.

5 Learning Outcomes

At the conclusion of week 1 students will be able to

1. Install Docker on their own laptops (or other computing resource), pull the amath583/base container, and carry out course assignments within it
2. Successfully create, compile, and run a one-file C++ program that prints a simple text message
   (a) Using a single compilation command
   (b) Designating the name of the executable
3. Successfully create, compile, and run a multi-file C++ program
4. Write and use a Makefile to automate compiling and linking a single multi-file C++ program
5. Write a one-file program to read whitespace-separated ASCII text and interpret as a container of numbers
6. Break the functionality for reading a container of numbers into separated functions and into separate files from the main executable function
7. Describe what in “include file” is and why a programmer would use one
8. Explain the C++ statement “using namespace std;”
9. Correctly create and use a C++ string datatype to create and print a text message