CSE-381: Systems 2
Exercise #1
Max Points: 20

You should save/rename this document using the naming convention MUid.docx (example: `raodm.docx`).

Objective: The objective of this exercise is to:

1. Review basics of working and programming under GNU/Linux from CSE-278
2. Review basics of HTTP from CSE-278
3. Review basics of C++ programming from CSE-278

Fill in answers to all of the questions. For some of the questions you can simply copy-paste appropriate text from the Terminal/NetBeans window into this document. You may discuss the questions with your neighbor or your instructor.

Name: Noah Dunn

Part #1: Review basic `bash`-shell skills

Estimated time: 20minutes

Exercise: This course will use the following Linux server throughout this course. You should memorize the name of this server: **`os1.csi.miamiOH.edu`**. For off-campus access, you will need to use Virtual Private Network (VPN) due to security concerns. You can get VPN software for free from: http://miamioh.edu/vpn. VPN also need 2-factor authentication via Duo.

Exercise: Complete this part of the exercise via the following steps:

1. First review the video (< 1 minute) titled "Access a GNU/Linux server from a Mac terminal" via https://youtu.be/SCBmLGzsAhs

2. Using the above video demonstration as reference, perform the following operation:
   a. Open a terminal on your laboratory computer
   b. SSH into the Linux server `os1.csi.miamioh.edu`

   If you have trouble logging onto the `os1.csi.miamioh.edu`, seek help. Confirm that you are logged onto os1 via the `hostname` command run as the bash shell prompt (`$`) as and pressing the ENTER (↵) key, shown below:

```
raodm@os1:~$ hostname↵
os1
raodm@os1:~$
```

3. Briefly review the following commonly used GNU/Linux commands for use in the next part of the exercise:

Commonly used Linux shell commands		
Command	***Description***	***Example usage***
`exit`	Log out of the Linux box	`$ exit`
`cd`	Change directory	`$ cd /usr/X11R6/bin`
`pwd`	Show present working directory	`$ pwd`
`ls`	List files.	`$ ls -l` `$ ls -l ../*.s`
`mkdir`	Make new directory	`$ mkdir csa-470`
`rmdir`	Remove empty directory	`$ rmdir csa-570`
`cp`	Copy file or files or entire directories on the same computer.	`$ cp ../a.s .` `$ cp -r ../a?b*.s .`
`scp`	Copy files from from one computer to another. Syntax: `scp SourceFilePath LoginID@server.edu:DestFilePath`	`$ scp a.txt rao@a.edu:cse381/` `$ scp rao@a.edu:ex1.cpp ./`

4. Double check your login – When you log onto the Linux machine, you will start off in a default directory called your **home** directory. You should create all your files and save your work off sub-directories under your home directory. To figure out what your home directory is, you need to use the `pwd` (present working directory) command (that is, type `pwd` at the shell (`$`) prompt and press ENTER key, which is indicated by ↵) as shown below:

```
raodm@os1:~$ pwd ↵
```

Note: The home directory will be in the format `/home/MUID` (where `MUID` is your Miami University unique ID), example: `/home/raodm`. In addition, note the following important terminology associated with directory hierarchies:

- **Absolute path**: In Linux, paths always start with a / (forward slash or just slash, *i.e.*, the division sign) indicating the root directory. Example: `/home/raodm`, `~/`, or `/usr/bin/ls` etc.
- **Relative path**: Paths that **do not start** with a / are relative paths. Relative paths indicate directory and file structures with respect to `pwd` (present working directory). Example: `../cse278` or `../` or `../../courses/csex43/exercises` or `cse278/exercises` etc.

5. Briefly practice using the above commands to perform the following tasks:
   a. Check to see your present working directory
   b. Change to your home directory (hint: `~/`)
   c. Create a directory named cse381 in your home directory. Verify that the directory has been created by listing the files.
   d. Copy a file named `/usr/share/common-licenses/GPL` to the newly created `cse381` directory. What command(s) can you use to confirm that you have successfully copied the file?

Provide brief responses to the following questions:

6. What is the server-name or hostname of the Linux server being used for this course?

The hostname of the Linux server is: `os1.csi.miamioh.edu`

7. Assume you would like to access the server when you are not on-campus. What extra software do you first need to run before trying to access the server while off-campus?

A VPN

8. What was the full `ssh` command you used (from Windows-powershell or a Mac-Terminal) to log onto the Linux server:

`Ssh dunnnm2@os1.csi.miamiOH.edu`

9. Your friend is running a program and it is generating the following error – "`Input file a.txt not found`". Your friend is perplexed. Searching the Internet did not seem to yield useful solutions. What suggestions can you offer to your friend to help troubleshoot this issue?

Your friend likely does not have a file named "a.txt" in his current working directory. He should utilize the "ls" command to list the files in the current working directory and determine if the file is in it. If it isn't, he/she may either utilize the "cd" command to access the correct directory, or alternatively create the file in the directory.

Part #2: Recap basics of HTTP protocol from CSE-278

Estimated time: 20 minutes

Background

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web (WWW). In HTTP a "client" (typically a Browser) send a request to a web-server requesting information, typically a file. If the request is valid, the server response with information. HTTP is a relatively straightforward multiline textual protocol. A key aspect of HTTP are headers, that have the following characteristics:

- HTTP headers consists of multiple lines. Each line is terminated with "`\r\n`" (aka CR-LF – Carriage return and Life feed).
- Request and response have 1 or more HTTP headers starting with the second line.
- Each header line is in the format "`Name: Value`" (*i.e.*, "Name Colon Value" pair). The "name" typically are predefined values, such as: "`Host`", "`Connection`", etc.
- The headers are terminated by 1-blank line (i.e., just "`\r\n`")

Exercise: In this part of the exercise you are expected to observe HTTP request and response headers from a browser. Several browsers (such as: Google `Chrome`) include "Developer tools" that permit you to view the actual HTTP request and response that the browser is used. It is useful to be able to view the headers for understanding the HTTP protocol and troubleshoot

problems. **However, bear in mind every browser has a different way to show the same information.**

1. Launch Google `Chrome` (on your local laboratory machine) and open a new tab.
2. Open `Developer Tools` (opens in the bottom of window) using the following keyboard shortcut:
   - Mac: Cmd + Opt + I
   - Windows: Ctrl + Shift + I
3. Next switch to the `Network` tab in `Developer Tools`.
4. Now access the following URL: http://ceclnx01.cec.miamioh.edu/~raodm/exercise1_numbers.txt. You should see a HTTP request in the `Network` tab. Stop recording by clicking the stop button (●) at the top-left corner of the `Network` tab.
5. From the Network tab copy-paste just one of the request headers in the space below:

```
GET /~raodm/exercise1_numbers.txt HTTP/1.1 Host: ceclnx01.cec.miamioh.edu
Connection: keep-alive Upgrade-Insecure-Requests: 1 User-Agent: Mozilla/5.0
(Macintosh; Intel Mac OS X 10_14_5) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/76.0.3809.100 Safari/537.36 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=
0.8,application/signed-exchange;v=b3 Accept-Encoding: gzip, deflate Accept-
Language: en-US,en;q=0.9 Cookie: _ga=GA1.2.1594359463.1566835480;
_gid=GA1.2.1193510725.1566835494
```

6. From the above request headers observe:
   - How the URL has been converted to an HTTP GET request.
   - Note how the file name and path was encoded.
   - Note how the `Host` header was used to transmit the host name from the URL.

7. Next, from the Network tab copy-paste just the response headers in the space below:

```
HTTP/1.1 200 OK Date: Mon, 26 Aug 2019 16:25:08 GMT Server: Apache/2.4.29
(Ubuntu) Last-Modified: Thu, 15 Aug 2019 15:45:04 GMT ETag: "21-59029c1df6131"
Accept-Ranges: bytes Content-Length: 33 Keep-Alive: timeout=5, max=100 Connection:
Keep-Alive Content-Type: text/plain
```

8. From the above response headers observe:
   - The first line is an HTTP status code in the form "`HTTP/1.1 200 OK`".
   - Note how the "`Server:`" header shows the name to be "Apache". Apache is one of the most popular web-servers developed in C/C++.
   - You should also observe a "Content-Type" header with value "text/plain" indicating that the file is a plain text file.

Part #3: Setting-up C++ projects in NetBeans

Estimated time: 15 minutes

Background: In this course it is highly recommended to use `NetBeans` 8.2 (using version 8.2 is important) for C++ programming. NetBeans is already installed on the lab computers. If-and-when you install it on your computer, you will need to configure it as shown in the `InstallNetBeansPlugin.pdf` in Canvas → `Files` → `Handouts` folder.

Exercise: Complete this part of the exercise via the following steps:

1. If you have not used NetBeans before, review the following video (< 8 minutes) demonstrating the process of creating, compiling, and running a C++ program in NetBeans: https://youtu.be/421zFJmrLkw (video available on Canvas as well).

2. Create a remote `NetBeans` project called `exercise1` on the Linux server **`os1.csi.miamioh.edu`**. Ensure you use `Miami University C++ Project` `to create` your `NetBeans` project. You don't need to create a main file (as you are given starter code for this exercise).

3. Double check you have created the project correctly.
   a. **Double check to ensure that the project is really on the server by ensuring that the server name is shown adjacent to the project name.**
   b. Note the directory where `NetBeans` created your remote project using your project's properties (`NetBeans` → Right-click-on-project → `Properties` → `General`) and ensure it is on the server. You will need this directory information in the next step.
4. Download the supplied starter files to your local computer and copy the files to your NetBeans project directory via the following steps:
   a. Note where the files are being downloaded to your local computer. Typically they are stored in the `~/Downloads` folder.
   b. Open a new/separate terminal on your local laboratory computer.
   c. From the local terminal, secure-copy the 2 starter files to your remote project via the following `scp` command. You should be familiar with `scp` from your CSE-278 course. If not, you should invest time to learn how to use it. It is one of the most important tools in the industry today. **Note**: Each one of the following commands are typed on 1 line. The commands appear wrapped onto more than 1 line merely because of page width.

```
BEN010-Mac ~/ $ cd ~/Downloads
```

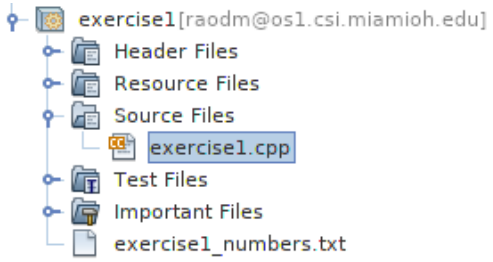
```
BEN010-Mac ~/ $ scp exercise1.cpp exercise1_numbers.txt
MUID@os1.csi.miamioh.edu:~/NetBeansProjects/exercise1
```

Troubleshooting: Having issues with `scp`? Here are a few things to keep in mind:

- **`scp` should be run on your local computer and not on the Linux server!**
- First form a mental model that you are working with 2 completely different computers.
- Remember syntax: `scp SourceFilePath LoginID@server.edu:DestFilePath`

- The files you are copying are in two different directories on the 2 computers. Use `ls` command to double check if files are present in the source directory from where you are trying to copy.
- Ensure you have your destination directory on the Linux server is correctly specified. Use the path from your `NetBeans` project settings.

5. Next, in your `NetBeans` project, add the files you have copied to the server to your project via: Right-mouse-click on `NetBeans` Project and use Add Existing Item… menu option to add the two files to your project. You may have to drag-n-drop `exercise1.cpp` into the `Source Files` category. After this step your project should appear as shown in the adjacent screenshot.



6. If you have setup your project, then the starter code should compile and run to produce the following output (inputs typed-in by the user are shown in **Blue** color, followed by Control+D on a separate line to input logical end-of-file input):

```
raodm@os1:~/NetBeansProjects/exercise1$ ./exercise1
Using standard I/O (type inputs and press Ctrl+D)...
1
2
3
Maximum value = 3
Using file I/O...
Socket and HTTP-based I/O...
Maximum value = 10
```

7. For future use remember these 4 steps: ❶ Create `NetBeans` Project using Miami University C++ project ❷ Download starter files to local computer, ❸ `scp` to server, ❹ Add files copied to Linux server to `NetBeans` project.

Part #4: Recollecting basic C++ concepts

Estimated time: 15 minutes

Background

Recollect the following key concepts regarding C++ program from CSE-278:

1. I/O is performed using `operator<<` (stream insertion operator), `operator>>` (stream extraction operator). Use `std::getline` method only for reading a whole line.
2. Prefer to use comparison operators (*e.g.*, <, >, ==, !=, etc.) for both primitive data types and object data types. This streamlines algorithmic thinking.
   - Within body of methods it is easier to use `auto` keyword so as not to worry about data type.
3. Rules-of-thumb for parameter passing:
   - Use pass-by-value for primitive types -- *e.g.*, `doIt(int i);`

- Use pass-by-reference for objects -- *e.g.*, `doIt(std::string& s);`
- Use const liberally -- *e.g.*, `doIt(const int i, const std::string& s);`

4. Prefer to use `std::vector` instead of arrays or lists.
5. The `std::unordered_map` can be used as an associative array – *i.e.*, index into the map need not be integer, *e.g.*, `map["one"]`
6. If you use the 'n'-word (*i.e.*, `new`) in your C++ programs, you will be dropped from the course!

Exercise

Study the supplied starter code that reviews the use of different I/O streams while paying attention to the following details:

1. The starter code has the following 3 methods:
   a. `printMax`: Prints (to a given output stream) maximum of values read from an input stream.
   b. `setupHttpStream`: Uses a Boost socket and HTTP GET request to retrieve a file from a given web-server.
   c. `main`: Calls the `printMax` method with different combinations of I/O streams.

2. Study the `printMax` method while paying attention to the following details:
   a. Note the parameter passing-convention and use of default parameter values. This is handy for defining APIs.
   b. Note the use of `InputType`, which is an alias for int. In this program we can change "`using InputType = int;`" to "`using InputType = std::string;`" (*i.e.*, a 1-word change) to make the program work with a completely different data type. This approach streamlines code reuse and maintenance.
   c. Notice how the while-loop is structured to keep reading input from is until end-of-file. This is a shortcut for "`while (!is.eof(), is >> input) {`". When using `std::cin`, we simulate EOF by pressing Control+D .
   d. Notice how the if-check "`input > max`" does not change whether inputs are `int`, `double`, `std::string`, `std::complex`, *etc.*, – that is, the code & logic for finding maximum does not change based on datatype and is very algorithmic (none of the stupid, `.equals` vs. `.compare` nonsense).

3. Study the `setupHttpStream` method while paying attention to the following additional details:
   a. Notice the use of `const` keyword to clearly communicate that the method has no plans on changing these values.
   b. Notice the connection to port 80, the default port number for WWW.
   c. Notice how the GET request is being generated. Compare with the GET request that your recorded from the Browser in the earlier part and you should see the same pattern here.
   d. Notice how `std::getline` is used to read line-by-line of data.
   e. Notice the empty while-loop to skip over HTTP response headers

4. Finally, study `main` while paying attention to the following details:
    a. Notice how `std::ifstream` (input-file-stream) and `std::ofstream` (output-file-stream) are used for file-based I/O.
    b. Notice how the `printMax` method is called with different streams. This should be the style of thinking you need to develop – write core logic without worrying about where the data is coming from – later on you can "wire" your core logic to read/write data appropriately.
    c. Think about how you could write code to get `printMax` method to read data from a web-server and write output to a file.

5. Additional details to pay attention to:
    a. Note the style and quality of comments. This would be the quality expected when you showcase your solutions online for future jobs.
    b. Note that there is a blank space between each word as in "`a + b`" and not "~~`a+b`~~", etc. **Don't worry – we have a style checker from Google that will help you learn good habits ☺**
    c. Note the format of the copyright statement at the top of the source code – by putting this copyright you are stating that you are in compliance with CSE department's student code of conduct -- http://miamioh.edu/cec/academics/departments/cse/academics/academic-integrity/index.html

Part #5: Submit to Canvas via CODE plug-in

Estimated time: 10 minutes

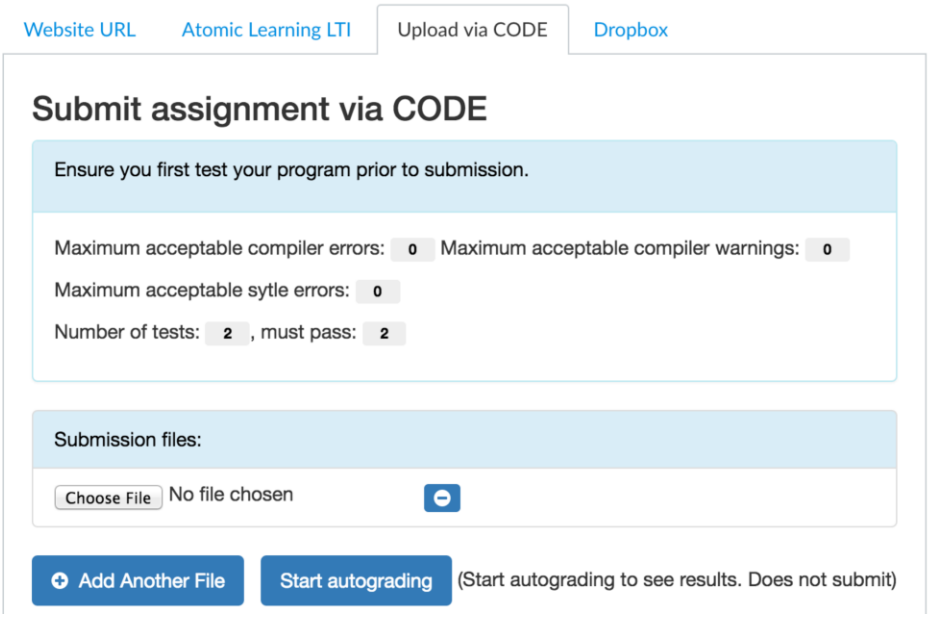
Background

Most (if not all) programming homework/project are required to be submitted via the CSE department's CODE plug-in. The plug-in has been designed to facilitate learning and teaching in programming centric courses such as this one. The CODE plug-in is setup to ensure: (1) your program compiles, (2) passes style checks, and (3) correctly operates for test cases. This eliminates issues with accidental incorrect submissions, incorrect solutions, etc. while promoting higher quality learning and outcomes for all.

Exercise

In this part of the exercise, you will be submitting the supplied starter code `exercise1.cpp` via Canvas CODE plug-in.

1. If this is your first time using the CODE plug-in then review the submission process via the following brief video demonstration -- https://youtu.be/P2bWUt5KqbU.
2. Save this MS-Word document as a PDF file – **Do not upload Word documents. Only submit PDF file**.
3. Practice submitting solutions via the CODE plug-in by submitting the following files:
    a. The supplied `exercise1.cpp` starter code
    b. Your lab notes saved as a PDF file.
4. Upload the files using the "`Upload via CODE`" tab shown in the screenshot below:



Ensure you actually **submit** the URL generated by CODE plug-in in the final step as shown in the Video demonstration.