DUE DATE: **By the deadline specified on Canvas**

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 review the following concepts from CSE-278

1. Basics of working and programming under GNU/Linux from CSE-278
2. Review HTTP protocol from CSE-278
3. Review using NetBeans for C++ programming
4. Review the CSE department's programming style requirements
5. Recap the use of debugger
6. Review handout to translate C++ → Java
7. Understand the procedure for submitting programs

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: **RAObot**

Part #0: Setup GNU/Linux Virtual Machine in VirtualBox

Estimated time: 15 minutes

In case you have not already done so, setup the GNU/Linux virtual machine for use in this course as demonstrated in the video titled "`Setup of GNU/Linux & VirtualBox`" in the VM & NetBeans video demonstrations page on Canvas.

In case you have trouble setting up the virtual machine, ensure that you seek help from your instructor or TA during your laboratory session. In general, do not delay seeking help. Instead, attend lab sessions so that you can learn the necessary skills and concepts to ensure success in this course and in your future careers.

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

Estimated time: 20 minutes

Background: This course will primarily use a GNU/Linux virtual machine for laboratory experiments and homework programming projects.

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

1. Start up your virtual machine.

2. 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/login` (where `login` is your login 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). Examples: `../cse278` or `../` or `../../courses/csex43/exercises` or `cse278/exercises` etc.

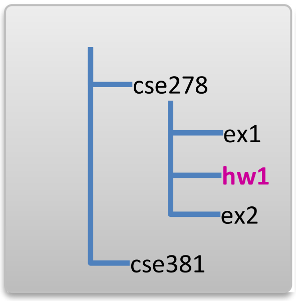
3. Now, briefly review the use of the first 8 commonly used GNU/Linux commands from page #3 in the `CommonMethodsAndCommands.pdf` file in the `Handouts` folder on Canvas. The first 8 commands in that document are: `exit`, `cd`, `pwd`, `ls`, `mkdir`, `rmdir`, `cp`, `scp`. By now you should have memorized these 8 commands from CSE-278, if not, you are missing important skills, not just for this course, but for your future career.

4. Briefly practice some of the Linux commands (using the above handout) to perform the following tasks (**Ensure you seek help if you get stuck or have questions**):
   a. Check to see your present working directory
   b. Change the PWD 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. Show the command(s) that can you have used to confirm that you have successfully copied the file below:

I used the `ls` command to ensure that the file has been copied as in:

```
raodm@os1:~/courses/cse381_spring20$ ls -l
~/courses/cse381_spring20/
total 40
drwxr-xr-x 2 raodm uuidd  4096 Jan 25 19:48 exercises
-rw-r--r-- 1 raodm uuidd 35147 Jan 25 20:36 GPL
```

5. Now, let's practice a few conceptual questions (similar to exam questions in this course) using the directory hierarchy shown in the adjacent figure. **Note that the absolute paths are not known (so you will need to use only relative paths).**



   a. Assume the present working directory in `hw1`. Illustrate a command that can be used to verify if a file named `hello.cpp` exists in `ex2` directory.

```
$ ls ../ex2/hello.cpp
```

   b. Assume the present working directory in `hw1`. Show a single shell command to copy a file named `main.cpp` from `cse278` directory to `cse381` directory.

```
$ cp ../main.cpp ../../cse381
```

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 Line-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"`)

Specifically, given an URL of the form `http://miamioh.edu/index.html`, an HTTP client (*e.g.*, a web-browser) converts the URL into the following HTTP request (i.e., multiple lines of text):

```
GET /index.html HTTP/1.1\r\n
Host: miamioh.edu\r\n
Connection: Close\r\n
\r\n
```

Notice how the resource (i.e., everything after the hostname) is placed between the words `GET` and `HTTP/1.1` on the first line. This line is often called the GET line or a GET-request. Notice how the hostname is included as an HTTP header. This is how every web-browser converts an URL to an HTTP request to be sent to a web-server. A web-browser may include additional request headers (similar to `Host:` or `Connection:`). A blank line, just a `"\r\n"`, is used to denote end of the request.

Using the above example convert each of the following URL to a corresponding HTTP GET request.

URL	Corresponding HTTP GET request

`http://os1.csi.miamioh.edu/~raodm/test.txt`	`GET /~raodm/test.txt HTTP/1.1\r\n`
	`Host: os1.csi.miamioh.edu\r\n`
	`Connection: Close\r\n`
	`\r\n`

`https://pc2lab.cec.miamioh.edu/`	`GET / HTTP/1.1\r\n`
	`Host: pc2lab.cec.miamioh.edu\r\n`
	`Connection: Close\r\n`
	`\r\n`

Part #3: Studying HTTP protocol using a web-browser

Estimated time: 20 minutes

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 machine and not on the VM) and open a new tab.
2. Open `Developer Tools` (opens in the bottom of browser window) using the following keyboard shortcut (you can use browser menus as well):
    1. Mac: Cmd+Opt+I
    2. Windows: Ctrl+Shift+I
3. Next switch to the `Network` tab in `Developer Tools`.
4. Now copy-paste the following URL (You may have to press Shift and the "↻ **Reload**" button if you have already accessed the 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 the request headers (click view source) in the space below:

```
GET /~raodm/exercise1_numbers.txt HTTP/1.1
Host: ceclnx01.cec.miamioh.edu
Connection: keep-alive
Cache-Control: max-age=0
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
DNT: 1
Accept: text/html, ,application/xml;q=0.9,image/webp,image/apng,*/*
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: _ga=GA1.2.486049154.1559230408
If-None-Match: "21-59029c1df6131"
```

6. From the above request headers briefly respond to the following questions on how the URL has been converted to a GET request:
   1. What was the host name in this example? Where did the host name (i.e., server name) get placed in the HTTP request?

      The host name in this example was `ceclnx01.cec.miamioh.edu`. It is always placed as the value for the HTTP header named `Host`.

   2. What was the file that we requested? Where did the file we requested get placed?

      The file that we requested was `~raodm/exercise1_numbers.txt`. Path to this file is always placed in the 1st line after the word "`GET`"

   3. We will work a lot with HTTP in this course. So, at this stage if you still feel unsure about how an URL gets converted to an HTTP-GET request, ensure you

7. Next, from the Network tab copy-paste just the response headers (click view source) in the space below:

```
HTTP/1.1 200 OK
Date: Mon, 19 Aug 2019 23:32:37 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=94
Connection: Keep-Alive
Content-Type: text/plain
```

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

Part #4: Setting up C++ project in NetBeans

Estimated time: 15 minutes

Background: In this course it is highly recommended to use `NetBeans` from a VM for C++ programming. To aid your memory briefly review the video demonstration titled "`C++ programming using NetBeans`" in the VM & NetBeans video demonstrations page on Canvas.

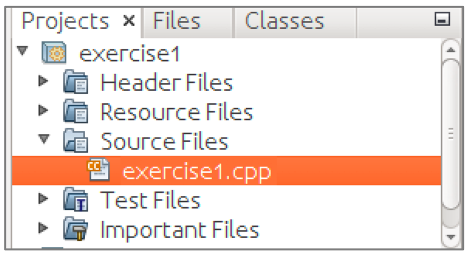
Exercise: Complete this part of the exercise via the following steps:

1. Create a new `NetBeans` project called **`exercise1`** in your GNU/Linux VM. 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).

2. Update the FireFox on your VM so that DUO is happy via the following steps:
   a. Open a Terminal in your GNU/Linux VM
   b. At the bash shell prompt (`$`) type the following command to update FireFox
   ```
   $ sudo apt install firefox
   ```
   When prompted type in your default password `student123`

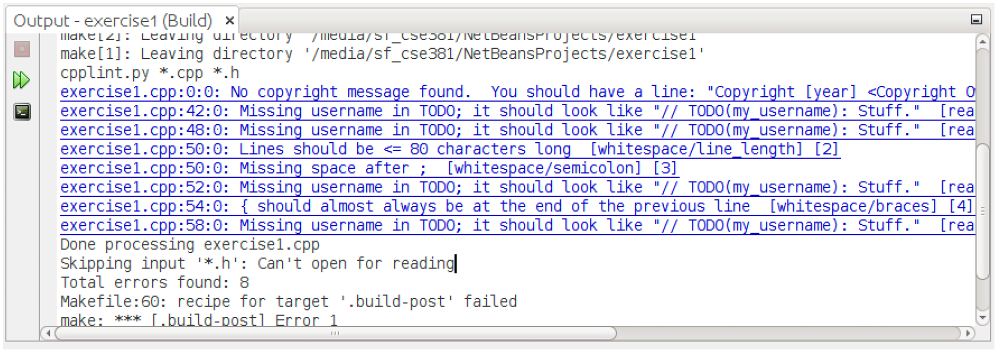
3. Download (use FireFox in your VM to make your life easier) the supplied starter files for this exercise to your `NetBeans` project directory on your VM. You can find out your project's directory by right-clicking on the project and in the pop-up menu, look at the `Properties`.

4. 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.



5. For future use remember these 3 steps: ❶ Create `NetBeans` Project using Miami University C++ project ❷ Download starter files to your project directory, ❸ Add files copied to Linux server to `NetBeans` project.

6. If you have setup your project, then the starter code should compile but will produce the style errors shown below:



```
make[2]: Leaving directory '/media/sf_cse381/NetBeansProjects/exercise1'
make[1]: Leaving directory '/media/sf_cse381/NetBeansProjects/exercise1'
cpplint.py *.cpp *.h
exercise1.cpp:0:0: No copyright message found.  You should have a line: "Copyright [year] <Copyright O
exercise1.cpp:42:0: Missing username in TODO; it should look like "// TODO(my_username): Stuff."  [rea
exercise1.cpp:48:0: Missing username in TODO; it should look like "// TODO(my_username): Stuff."  [rea
exercise1.cpp:50:0: Lines should be <= 80 characters long  [whitespace/line_length] [2]
exercise1.cpp:50:0: Missing space after ;  [whitespace/semicolon] [3]
exercise1.cpp:52:0: Missing username in TODO; it should look like "// TODO(my_username): Stuff."  [rea
exercise1.cpp:54:0: { should almost always be at the end of the previous line  [whitespace/braces] [4]
exercise1.cpp:58:0: Missing username in TODO; it should look like "// TODO(my_username): Stuff."  [rea
Done processing exercise1.cpp
Skipping input '*.h': Can't open for reading
Total errors found: 8
Makefile:60: recipe for target '.build-post' failed
make: *** [.build-post] Error 1
```

7. **Ensure you seek help if you get stuck or if you need any clarifications.**

Part #5: Understanding and fixing style issues

Estimated time: 15 minutes

Background: In your jobs, you will be developing software for other programmers to use, and not just for yourself. Hence, you have to get into the habit of "*thinking about others*" – a key requirement being adhering to coding styles. So, in preparation for your careers, the CSE programming style guidelines will be strictly enforced in this course. The style guidelines are pretty simple (and the same rules apply for Java program as well!)

- First, keep in mind different people use different devices (smartphones, tablets, terminals, etc.). Hence, all technology companies (including Google, Facebook, Amazon, etc.) mandate the following style so that source code renders consistently on all devices:
    - Must use only spaces for indentation. Use 4-spaces. **Lucky for you, when you click the "Tab" key in `NetBeans` it actually uses 4-spaces.**
    - No line should be longer than 80-characters (see thin vertical red line in `NetBeans`).
    - None of the methods should be longer than 25-lines of code (does not include comments or blank lines). This is to ensure good program structure, ease readability, and understandability of methods. If you have methods longer than 25-lines of code, that is an indication that you are lacking structured thinking process. So, you have to get into the practice of developing good problem-solving skills.

  Again, keep in mind you are developing code for other people and you have to demonstrate to companies you can develop good, well-styled code to get good jobs.
- There must be spaces between words (just like in English) – so, "`if(a>b){`" is wrong style. Instead, it should be typed as "`if (a > b) {`" (note spaces between words).

Exercise: In this part of the exercise, you are expected to learn about style issues and fix style errors in the starter code.

1. read the style error messages in the output window and try to fix them. Each time you fix a line, compile your program and ensure you have fixed the problem correctly.

2. As part of the TODO, you are required to add documentation to the process method to explain in full detail what the lines of code are doing. Ensure your comments also clearly communicate **why** the code fragment is present.

3. Once you have fixed the style issues and added documentation, you should be able to run the program and produce the following output:

```
raodm@os1:~/NetBeansProjects/exercise1$ ./exercise1
<html>
  <body>
    <h2>Analysis results</h2>
    <p>Number of values: 13</p>
    <p>Sum of numbers: 83</p>
    <p>Average value: 6.38462</p>
  </body>
</html>
```

4. Briefly state (in your own words) why we will be practicing “Programming style guidelines” in this course (*i.e.*, why can’t you post ugly code on GitHub for your job interview and hope to make a good “first impression”?)
   We all get only one chance to make a “first impression”. Program style is very important in jobs because we will be developing programs for others to use. Good style makes it easy for others to read and understand our program. So, employers pick people who can consistently demonstrate good programming style – it is not enough that I claim I can do good style (I can claim anything I want) but proving I can do good style is what is important and that comes through practice. So, in this course I am glad we will be practicing good style.

Part #6: Debugging a program

Estimated time: 15 minutes

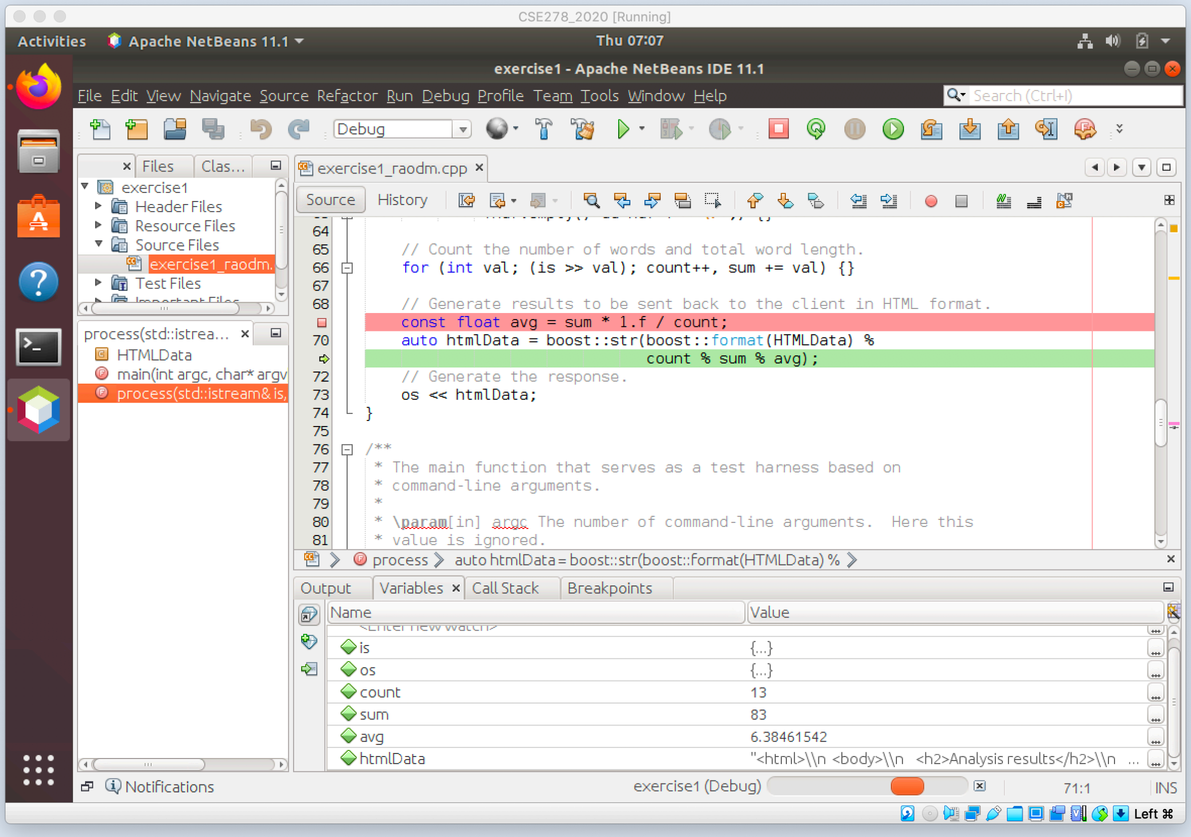
Background: The debugger is a very important tool that you should learn how to use. It will come in handy in future laboratory exercises and homework. Consequently, you should become very comfortable with using the debugger.

Exercise: Complete this part of the exercise using the following procedure:

1. If you have not used the debugger in NetBeans, first review its usage via the following video: https://youtu.be/O7UYNut1oks (video also available on Canvas)
2. In order to use the debugger, you first need to set a breakpoint. Set a breakpoint in `exercise1.cpp`, in the `process` method, at the line “`const float avg = sum * 1.f / count;`”.
3. Run the program via the debugger.
4. Your friend is trying to use the debugger. However, your friend is getting the warning – `&"warning: GDB: Failed to set controlling terminal: Operation not permitted\n"`. Your friend is very confused. What should your friend do?
   Obviously, my friend did not watch the videos posted by the instructor. I would point my friend to the spot in the video where the instructor says to “ignore this warning message” as it is a spurious warning.

5. Observe how the "Call Stack" looks.
6. Observe how to the local variables and their values are shown in the "Variables" tab.
7. Let's observe the change in values of variables –use the "Step Over" button to step line-by-line until you reach the line of code "`os << htmlData;`" (about 2-lines of code).
8. Finally, make a screenshot of your whole Virtual Machine window showing NetBeans and the following:
    a. The current line of code
    b. The stack trace tab should be visible
    c. The variables tab should be visible

Place screenshot of VM with NetBeans debugger showing all of the above 3 pieces of information here



```
process(std::istrea...
    HTMLData
    main(int argc, char* argv
    process(std::istream& is,
```

```
    // Count the number of words and total word length.
    for (int val; (is >> val); count++, sum += val) {}

    // Generate results to be sent back to the client in HTML format.
    const float avg = sum * 1.f / count;
    auto htmlData = boost::str(boost::format(HTMLData) %
                               count % sum % avg);
    // Generate the response.
    os << htmlData;
}

/**
 * The main function that serves as a test harness based on
 * command-line arguments.
 *
 * \param[in] argc The number of command-line arguments.  Here this
 * value is ignored.
```

process > auto htmlData = boost::str(boost::format(HTMLData) %

Name	Value
is	{...}
os	{...}
count	13
sum	83
avg	6.38461542
htmlData	"<html>\\n <body>\\n <h2>Analysis results</h2>\\n ...

Part #7: Review Socket I/O operations

Estimated time: 10 minutes

Background: The supplied starter code is designed to interact with a web-server and obtain data from a web-server. To interact with a remote web-server we use the TCP/IP protocol to send simple textual request to the web-server. Of course, the web-browser (a C++ program) is also doing pretty similar operation.

Exercise: This part of the exercise requires you to review the logic in the `main` function and briefly (2-to-3 sentences) respond to the following questions:

1. Copy-paste the line of code that creates a socket connection to the web-server in the space below.

   The following line of code creates a socket in the main function:
   ```
   tcp::iostream data(host, port);
   ```

2. What is the name of the web-server from where data is being currently obtained?

   The name of the web-server is hard coded to be www.users.miamioh.edu in the following line of code:
   ```
   const std::string host = "www.users.miamioh.edu";
   ```

3. How will you modify the code in order to interact with a different web-server?

   We can change the name of the web-server via:
   - Changing the hard-coded value – *i.e.*, change the value for the `host` constant.
   - Changing the code to read the web-server as input from the user.

4. In the earlier part of this exercise, we studied how the HTTP request is a simple multiline, textual protocol. What lines in the `main` method produce the HTTP-GET request?

   The following lines of code create and send the HTTP-GET request to the web-server:
   ```
       data << "GET "   << path << " HTTP/1.1\r\n"
            << "Host: " << host << "\r\n"
            << "Connection: Close\r\n\r\n";
   ```

5. Using the HTTP-GET request being generated in the `main` function as the reference, show the corresponding URL:

   The GET request being generated in the `main` function corresponds to the following URL:
   ```
   http://www.users.miamioh.edu/raodm/nums.txt
   ```

Part #8: Review Java → C++ translation guide

Estimated time: 10 minutes

Background: Many of you should be comfortable with programming in Java. C++ and Java are syntactically very close. Importantly, the logic (or approach to problem-solving) in the two languages is the same! However, the API (*i.e.*, classes and methods) are different. The biggest difference between Java & C++ arises in input-output (I/O) operations. However, with practice C++ becomes as easy as Java.

Exercise: In order to help you transition between Java and C++, there is a handy Java → C++ translation guide on Canvas. Briefly review this guide via the following steps:

1. View the translation guide on Canvas→Files→Handouts→JavaCppTranslationGuide.pdf.
2. Now, let's use the translation guide to understand the `substr` method in C++. In C++, sub-string method `substr(int index, int len)` the 2nd parameter is the number of characters to return (**because that is how CPUs work**). Ensure you understand substring by completing the table – for each row, fill-in either a call to `substr` to obtain the desired output or indicate output from the `substr` method call.

`std::string str = "0123456789";`

Call to substr method in C++	Desired/Expected output
`str.substr(3, 4)`	"3456"
`str.substr(4, 1)`	"4"
`str.substr(8, 2)`	"89"
`str.substr(7)`	"789"

3. Now, to really ensure we understand `substr` in C++, let's write a 1-liner method in C++ that is equivalent to Java's substring method that takes 2 index positions.

```
/**
 * Implement this method to return a substring from the given
 * string str similar to what Java's substring method would return.
 */
std::string substring(const std::string& str,
                      size_t beginIndex, size_t endIndex) {
    // Write the 1-line C++ code to return substring similar to
    // how Java would return.

    return str.substr(beginIndex, endIndex - beginIndex);
}
```

Part #7: Submit to Canvas via CODE plug-in

Estimated time: 5 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.

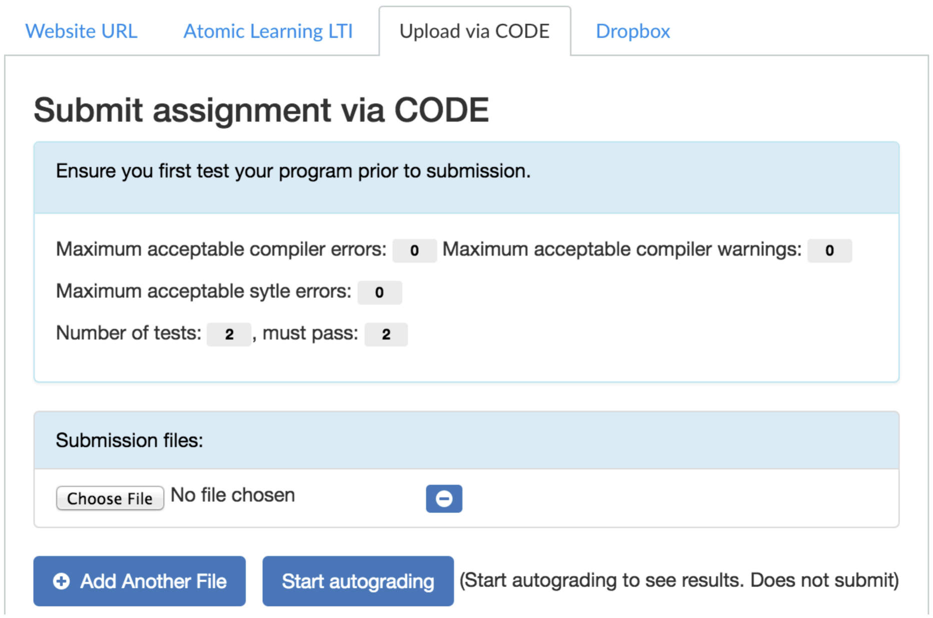
Exercise

DUE DATE: **By the deadline specified on Canvas**

In this part of the exercise, you will be submitting the necessary files via the 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 `exercise1.cpp` starter code
    b. Your lab notes for this exercise (*i.e.*, this document) saved as a PDF file.
4. Upload the files using the "`Upload via CODE`" tab shown in the screenshot below:



Submit assignment via CODE

Ensure you first test your program prior to submission.

Maximum acceptable compiler errors: 0 Maximum acceptable compiler warnings: 0

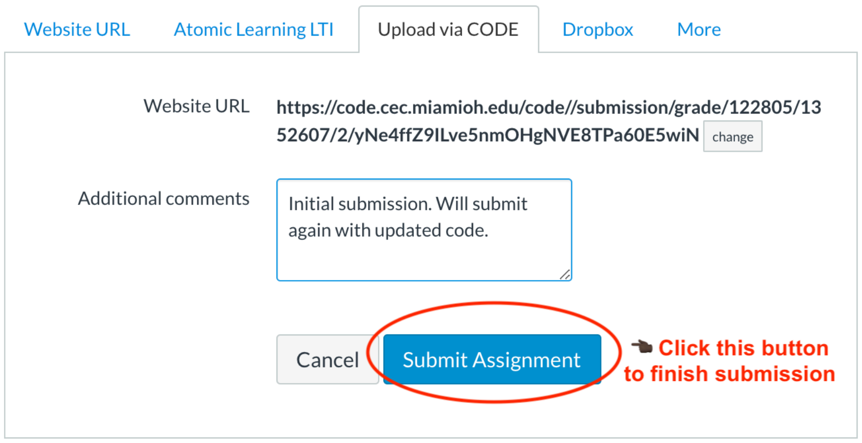
Maximum acceptable sytle errors: 0

Number of tests: 2 , must pass: 2

Submission files:

(Start autograding to see results. Does not submit)

Ensure you actually **submit** the URL generated by CODE plug-in in the final step as shown in the Video demonstration and in the screenshot below:



Website URL https://code.cec.miamioh.edu/code//submission/grade/122805/1352607/2/yNe4ffZ9ILve5nmOHgNVE8TPa60E5wiN

Additional comments: Initial submission. Will submit again with updated code.