As with the last lab, you will work in teams of three, though each of you needs to hand in your own individual submission. Because we now have 20 students in the class, there will be one group of size two.

Getting Started: Log in to the Linux network by opening Terminal on your machine, and then executing the following command:

`ssh -Y` *netid@linux_hostname*

or

`ssh -l` *netid linux_hostname*

(that's a lowercase "ell" in the second command) replacing *netid* with your University netid and *linux_hostname* with any of `mathcs01`, `mathcs02`, ..., `mathcs06` or `turing2`. (If your local account name is the same as your account name on the Linux network, you may omit the "*netid@*" in the command.) Hopefully all of you won't use the same machine for logging in — if you're physically in the lab, one way to distribute load would be to check with your left and right neighbors and be sure you are logging in to three different machines. Upon successful login, you will then be working remotely inside your home directory (∼) on the Linux network.

In your linux home directory, create a new directory called `cmsc240`. 'cd' into this directory and create another subdirectory called `lab3`. Once again, 'cd' into this new subdirectory. Then copy all the files and directories from the appropriate directory in my home directory using the following command. (Note that there is a "." at the end of the line — that is important. It says to place the copied files into the current working directory.)

```
cp ∼dszajda/outbox/cs240/lab3/* .
```

Answers to all 20 underlined questions must go in the `cmsc240_lab3_NETID.txt` (which you must rename appropriately using your netid). Note that this file has space for answers, but also parts of questions (e.g., number 14). Make sure to include your name inside the .txt file as well. Note also that

NOTE: In what follows, we use *parameter* to refer to the names used in the method declaration/definition, and *argument* to refer to the values supplied when a method is called. You may have previously heard these two entities referred to as *formal parameters* and *actual parameters*, respectively.

Lab Exercise #1: Open `ProgramOne.cpp` in an editor and inspect the source code. This program has a `changeValue` method which uses *pass-by-value* parameters.

Now compile and run the program.

1. What changes do you notice in the before and after versions of the UR id and name?
2. Why? Choose best explanation.

Keeping the original method intact, write a new but similar `changeValue` method with a different signature using pointers as the parameters (don't forget your method prototype, if appropriate):

```
void changeValue(int* someInt, std::string* someString);
```

In the definition of the method, make sure that you dereference the parameters when doing the assignment (and continue to assign a different netid and name). Back in `main`, in addition to the existing method call, include a call to this new method (think carefully about what you need to pass in for arguments) and subsequent cout. Recompile and run.

3. What changes do you notice in the before and after versions as a result of calling your new method?
4. Why? Choose best explanation.

Now, change the original pass-by-value version of the `changeValue` method to use *reference parameters*:

```
void changeValue(int& someInt, string& someString);
```

NOTE: When you use reference parameters, C++ handles pass-by-reference for you automatically. In other words, in the body of your method, you treat the parameters as regular variables (not as pointers, i.e., don't dereference within your method), and C++ takes care of the pass-by-reference. Similarly, when you call your method, pass a variable directly — not using the variable's address.

From the C++ Super-FAQ: `https://isocpp.org/wiki/faq/references#overview-refs`

> Important note: Even though a reference is often implemented using an address in the underlying assembly language, please do *not* think of a reference as a funny looking pointer to an object. A reference is the object, just with another name. It **is** neither a pointer to the object, nor a copy of the object. It **is** the object. There is no C++ syntax that lets you operate on the reference itself separate from the object to which it refers.

In the definition of the method, make sure that you **do not** dereference the parameters when doing the assignment. Note that back in `main`, you already have a call to this new method (it **does not** explicitly pass addresses for arguments) and subsequent cout. Compile and run the program again, and you should notice that the reference parameters cause the before and after values to change.

5. What code changes are required when using a pointer as a parameter?
6. What code changes are required when using a C++ reference as a parameter?
7. What is the primary difference (in terms of effect) between pass-by-value and pass-by-reference?

Lab Exercise #2: Open `ProgramTwo.cpp` in an editor and inspect the source code. Note that the method named `changeValue` uses a pointer-type parameter, declares and assigns one local variable, and executes two other assignments on the parameter. Compile and run the program.

8. What changes do you notice in the before and after versions of `anInt` and `intPtr`?
9. For `anInt`, why? Choose best answer.
10. For `intPtr`, why? Choose best answer.

Lab Exercise #3: Open `ProgramThree.cpp` in an editor and inspect the source code. This program makes use of a *struct* to define a student data type, encapsulating a student's UR ID, name, and netid. (A struct is similar to a class in that you can encapsulate a collection of data into one type. Structs in C++ can have methods, but in C, structs are used only for encapsulating data.)

11. What is the best explanation of the apparent intent of the program as given (even if not successful in that intent)?
12. Compile and run the program. Does the program accomplish that intent? Why or why not?

Modify this program so that it has two methods both named `changeID`: one which uses a pointer parameter and one which uses a reference parameter.

> *NOTE: Remember that when using a struct variable, you use a dot (.) to access a field in the struct, but when using a struct pointer, you use an arrow (->) to access a field.*

Modify `main` so that your program calls both of these methods and exhibits pass-by-reference functionality (you will need to use print statements to convince me of the effect).

13. What is the best explanation of dot versus arrow use?

Lab Exercise #4: Inspect the source code for `Student.h` and `Student.cpp`. These two files together define a class corresponding to a student — both data and methods.

14. Based on your knowledge of C++ and your knowledge of Java, select **all** that options (listed in the `cmsc240_lab3_NETID.txt` file) that are correct.

Now inspect the source code for `ProgramFour.cpp`. This program declares two Student objects, one *dynamically allocated* and one not.

15. What are the data types of the variables `first` and `second`? Choose best answer.

Now complete the following sequence of modifications to `ProgramFour.cpp`, and answer the corresponding underlined questions.

- In C++, any dynamically allocated memory must be freed when you are done using that memory. Unlike Java, C++ does not do garbage collection for you automatically on dynamically allocated memory. Before the return statement in your program, add the following two lines:

  ```
  delete first;
  delete second;
  ```

  Try to compile your program (remember to include `Student.cpp` in the compile command):

  ```
  g++ -std=c++17 ProgramFour.cpp Student.cpp -o ProgramFour
  ```

  You should receive an error during compilation.

  16. Copy and paste your compile error.
  17. Why do you get a compile error? Choose the best answer.

  Remove the offending statement from your program so it will compile.

  When adding any more code in steps below, make sure to keep the valid delete statement as the last line just before the return statement — otherwise you will eventually get execution errors.

- Include statements in `main` that call the `print()` method for each of the two objects.

  > *NOTE: Similar to the struct above, use an arrow (->) to invoke a method when using a pointer but use a dot (.) otherwise.*

  Make sure your program compiles and executes correctly.

- Include statements in `main` that call all three of the "set" methods for each of the two objects (one pointer, one not). Use output from the `print()` method to verify the changes take place.

- In your `ProgramFour` program, write a method named `change` that accepts a Student parameter (not a pointer or reference), and then uses that parameter to call all three of the "set" methods, providing different values than those back in `main`. Then in `main`, call your new method appropriately, and then use the `print` method to determine if the changes persist.

  18. What changes in the student info do you notice after calling your method?
  19. Why? Choose the best answer.

- Modify your new method to accept a Student reference (not a pointer). Compile and execute your program, and you should notice that changes enacted within your new method persist.

  20. Why? Choose the best answer.

- Now write a similar but separate method named `changeViaPtr` that accepts a Student pointer (not a reference), and then uses that pointer to call all three of the "set" methods, providing different values than those back in `main`. Then in `main`, call your new method appropriately, and then use the `print` method to verify that changes persist in this case too.

Submitting: Submitting is the same as with lab 2 (only with the appropriate change from "lab 2" to "lab 3"). In future labs, I will not include these insructions for creating a gzipped tarball, etc.

1. First, you will need to package your work into a **tar** archive file for submitting (tar is required here so that you are required to learn how to use it). Let's assume that you have stored all necessary files in a directory called `lab3` (which may contain subdirectories). From the **parent** directory of your `lab3/` directory, execute the following command, replacing `netID` with your netid:

   ```
   tar -czvf cmsc240_lab3_netID.tgz lab3/*
   ```

   This command will combine all the files and subdirectories in your `lab3/` directory into a single tar archive file named `cmsc240\_lab3\_netID.tgz`. (Have a look at the tar man-page about the `czvf` flags.)

   As mentioned in Lab 2, we refer to such a file as a "gzipped tarball".

2. Verify the contents of your gzipped tarball. First, verify that it is of reasonable size (in bytes) and that it is identified as a "gzip compressed data" file:

   ```
   ls -l
   file cmsc240_lab3_netID.tgz
   ```

   The first command gives you a long listing of files in the directory, which should provide the size, in bytes, of all of the files in the directory (well, not all – files that begin with a dot, like `.bash_profile`, and some other special files, won't show up). The `file` command provides the file type. In this case it should say something like

   ```
   testing.tar: POSIX tar archive
   ```

   Next, check that the contents are what you expect:

   ```
   tar tzvf cmsc240_lab3_netID.tgz
   ```

   This should give you a listing of the folders and files contained in your tarball.

3. Attach the gzipped tarball to an email to the appropriate assignment email address (as usual) to submit.

Naming:

As always, because everyone will be submitting their lab to a single Box folder, they can't all be called "Lab3.tgz" or the like. Instead, they must have some indication of who completed the submission. Naming is also important because I sometimes use software to grade submission, and my code expects specific naming conventions. For this lab, your submission **MUST** be named `cmsc240_lab3_netID.tgz`, where `netID` is, of course, your netID.

Submission:

The high level picture is that to submit any labs/project in this course, you send an email to a special email address with your **single** submission file attached. This has the effect of placing your submission in the appropriate Box folder.

The the email address for this lab is `Lab3.r3anskxap9ptfu25@u.box.com`. Thus to submit this lab, you should attach your tar or zip file to an email sent to the email address `Lab3.r3anskxap9ptfu25@u.box.com`.