Getting Started: Remotely log in to the Linux network, create a new `lab5` directory in your `cmsc240` directory, and then recursively copy all the files and directories from the appropriate directory in my home directory: `cp -r /home/dszajda/outbox/cs240/lab5/* .`

Answers to all underlined questions must go in the `cmsc240_lab5_NETID.txt` (which you must rename!). And as in the previous lab, this "answer" file also has parts of questions. Note that unlike some previous labs, the `.txt` file is *not* the only file you need to submit. Instead, you will be tarring up your entire `lab5` directory and submitting that (either as a `.tar` file or as a gzipped tar ball). The files that must be contained in the directory are:

- ProgramOne.cpp (after you have modified it)
- ProgramOne (the executable you built *on the cluster*)
- ProgramTwo.cpp (after you have modified it)
- ProgramTwo (the executable you built *on the cluster*)
- ProgramThree.cpp (after you have modified it)
- ProgramThree (the executable you built *on the cluster*)
- ProgramFour.cpp (after you have modified it)
- ProgramFour (the executable you built *on the cluster*)
- Student.h
- Student.cpp
- cmsc240_lab5_NETID.txt

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_lab5_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.

Naming:

Same as usual. The name of your submission file for this lab **MUST** be `cmsc240_lab5_netID.tar` (or `.tgz` if a gzipped tar ball), where the `netID`entry is 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. If your submission requires more than one file, you should `tar` or `zip` your files together to create your single submission file.

The email address for this lab is

`lab5.7r0zv7j6lf1orfg7@u.box.com`.