- No late tutorials will be accepted.
- If there are specific instructions for making a function, please follow them exactly. That means that
  - function names
  - function return types
  - parameter types and order

  should all be **EXACTLY** as described. If the script can't read it, you will receive 0 for that part.
- Your **Tutorial 3** code will be marked by a Python script running on the Gradescope server. You are being given a similar script so that you may make sure your code runs correctly.

1 Submission Instructions

Download "tutorial4.zip". Unzip it into your working directory. There is a directory "tutorial4" and the test file "t4test.py". In the "tutorial4" folder are "test.cc", "defs.h" and "Makefile" to get you started. To the "tutorial4" folder you should add the following files.

1. Header and source files for the `Photo` class from Assignment 2, Section 6.2, with the appropriate modifications.
2. Header and source files for two additional **classes**, `HeapArrays` and `StackArrays`.

Once you have written the classes and completed your tests, submit this tutorial to Gradescope. The Gradescope server is flexible in how you submit. You may zip the folder, zip the files, or submit the folder itself or individual files. What follows are one set of instructions that will work on Gradescope. You will zip the "tutorial4" directory into a file "tutorial4.zip". If you are doing this in the course VM you must do this from the command line. Open a terminal in the folder that contains "tutorial4". Use the command `zip -r tutorial4.zip tutorial4`. This will zip the `tutorial4` folder, or update it if you change the contents. DO NOT USE tar FILES. These do not seem to work well with Gradescope. Submit to Gradescope by the deadline. You will receive your mark immediately, and you may submit as many times as you like. You may also submit late tutorials, but there will be a penalty (somewhere between 10% and 50%). Presently the server is set to receive tutorials up to a week late.

2 Testing Your Tutorial With `t4test.py`

`t4test.py` is a test script that is very similar to the script that is being run on Gradescope (there might be slightly different input or different even tests). So the mark you see here should be close to the mark you will receive on Gradescope. To run `t4test.py`, open a terminal in the directory that contains your "tutorial4" folder. You may have to make the script executable, so type `chmod +x t4test.py`. You may run the script as is, in which case it will look for a file to unzip. Or, if you have not zipped your files yet you may supply a "-nozip" argument, in which case it looks for the "tutorial4" folder.

To have the script unzip `tutorial4.zip` and then test your code, run `./t4test.py`. To skip the unzip step use `./t4test.py -nozip`. When your tutorial is being officially marked we expect a zipped file.

Running this script will generate a file "results.txt" just outside of the "tutorial4" folder. This will have some useful output as well as the mark.

3 Learning Outcomes

In this tutorial you will first write the `Photo` class from Assignment 2, Section 6.2. Then you will make 4 different kinds of primitive `Photo` arrays and allocate and deallocate them correctly.

4 Instructions

4.1 Overview

In this tutorial you will write the `Photo` (Section 6.2) class from Assignment 2. You will write two other classes, `HeapArrays` and `StackArrays`. There is one test file provided, `test.cc` that will test the functions that you provide from those classes. You must provide a Makefile that compiles all classes into object files and compiles `test.cc` into an executable called `test`. As usual the test script, `t4test.py` is provided. This time the script is run with `valgrind`, so it will check the `valgrind` output to see if there are memory leaks.

4.2 Photo Class

Complete Section 6.2 in Assignment 2. Then make the following modifications.

- ✓ You will notice that you will not be able to initialize arrays of `Photo` objects. Add a no-argument constructor that gives appropriate default values to the member variables. You may include this in your assignment if you wish (but you don't have to).
- ✓ Add a copy constructor. This should copy the `title` and `Date`, but instead of copying the `content`, set the new `content` to a message from the RCMP about violating copyright infringement. Be sure your `content` includes the term "RCMP", as the test script will search for it. You should **NOT** include this copy constructor in your assignment. Your assignment uses the copy constructor, but requires that the content be copied as well.

4.3 StackArrays Class

Use the `ARR_SIZE` preprocessor constant from `defs.h` to initialize each array to its proper size.

✓1. Member variables:

   (a) A *statically allocated* array of `Photo` objects.

   (b) A *statically allocated* array of `Photo` pointers.

   (c) An `int` that keeps track of the current number of `Photos` in this data structure.

✓2. A no-argument constructor. Initialize the member variables appropriately.

✓3. A destructor. Ensure that all dynamically allocated memory reachable by this class is freed.

✓4. Member functions - make a getter for each array:

   (a) `getObjectArray()` should return the *statically allocated* array of `Photo` objects.

   (b) `getPointerArray()` should return the *statically allocated* array of `Photo` pointers.

5. Make a `void addPhoto` that passes in a `Photo` object. This `Photo` should be added to both arrays. You should assign this `Photo` parameter to the next available location in the array of `Photo` objects (using the assignment operator, `=`). You should also make a dynamically allocated *copy* of this `Photo` and add the pointer to the next available location in the array of `Photo` pointers.

4.4 HeapArrays Class

Use the `ARR_SIZE` preprocessor constant from `defs.h` to initialize each array to its proper size.

✓ 1. Member variables:

   (a) A *dynamically allocated* array of `Photo` objects.

(b) A *dynamically allocated* array of `Photo` pointers.

(c) An `int` that keeps track of the current number of `Photos` in this data structure.

✓ 2. A no-argument constructor. Initialize the member variables appropriately.

✓ 3. A destructor. Ensure that all dynamically allocated memory reachable by this class is freed.

4. Member functions - make a getter for each array:

✓ (a) `getObjectArray()` should return the *dynamically allocated* array of `Photo` objects.

(b) `getPointerArray()` should return the *dynamically allocated* array of `Photo` pointers.

✓ 5. Make a `void addPhoto` that passes in a `Photo` object. This `Photo` should be added to both arrays. You should assign this `Photo` parameter to the next available location in the array of `Photo` objects (using the assignment operator, `=`). You should also make a dynamically allocated *copy* of this `Photo` and add the pointer to the next available location in the array of `Photo` pointers.

4.5 Makefile

Your Makefile should compile three object files, `Photo.o`, `StackArrays.o` and `HeapArrays.o`. It should link these object files to the `test` executable. In addition your Makefile should contain an `all` command that creates the `test` executable and a `clean` command that removes all executables and object files.

4.6 t4test.py

Run this python script to test the functions described above. Correct all errors.