- 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 4** code will be marked by a Python script. You are being given the script so that you may make sure your code runs correctly. As such, submissions with improper files, configuration, or function signatures will not be accepted.

---

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 `Episode` class from Assignment 2, Section 6.2.
2. Header and source files for two additional **classes**, `HeapArrays` and `StackArrays`.

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. Submit `tutorial4.zip` to Brightspace by the deadline. DO NOT USE .tar OR .tar.gz FILES. Use .zip only please.

2 Testing Your Tutorial With `t4test.py`

`t4test.py` is a test script that is very similar to what will be used to mark your tutorial (basically I will change the input and expected output ... unless I get lazy, then I won't change anything). So the mark you see here should be the mark you receive (as long as you did not hard code output. If you try a shortcut you might get burned). 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 `Episode` class from Assignment 2, Section 6.2. Then you will make 4 different kinds of primitive `Episode` arrays and allocate and deallocate them correctly.

4 Instructions

4.1 Overview

In this tutorial you will write the `Episode` (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 Episode Class

Complete Section 6.2 in Assignment 2. You will notice that you will not be able to initialize arrays of `Episode` objects. Add a no-argument constructor that gives appropriate default values to the member variables.

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 `Episode` objects.
   (b) A *statically allocated* array of `Episode` pointers.
2. A no-argument constructor. Initialize the array of `Episode` pointers with dynamically allocated `Episode` objects (you may use the no-argument constructor here).
3. A destructor. Ensure that all dynamically allocated memory is freed.
4. Member functions - make a getter for each array. Since these are test classes, to keep it simple, use return values rather that output parameters:
   (a) `getObjectArray()` should return the *statically allocated* array of `Episode` objects.
   (b) `getPointerArray()` should return the *statically allocated* array of `Episode` 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 `Episode` objects.
   (b) A *dynamically allocated* array of `Episode` pointers.
2. A no-argument constructor. Initialize both arrays such that every array location either contains an `Episode` object or points to an `Episode` object.
3. A destructor. Ensure that all dynamically allocated memory is freed.
4. Member functions - make a getter for each array. Since these are test classes, to keep it simple, use return values rather that output parameters:
   (a) `getObjectArray()` should return the *dynamically allocated* array of `Episode` objects.
   (b) `getPointerArray()` should return the *dynamically allocated* array of `Episode` pointers.

4.5 Makefile

Your Makefile should compile three object files, `Episode.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.