(1) Steps needed to get going on your GitHub Classroom Based Assignment

1. **Get a GitHub account if you don't already have on(https://github.com/)**

2. **Set up an SSH key**

   This will allow you to download and upload (clone and push) your repository from GitHub to your development environment

3. **Accept the assignment by going to the specific link provided on Canvas** `Modules > Week 1 > Assignment-0`
   - For the first assignment you will need to bind your Canvas account with the Inginious autograder (might need to clear cache, use different browser)
     - Use your colorado.edu google account
   - This will generate your own private repository, containing all the files you will need for your assignment

4. **Clone the private repository into your "local" environment to begin working on the assignment**

   - The first time you use git on your machine you need to set up your git global variables: use the same username and email as for your GitHub account to keep things simple
   - Use JupyterHub https://coding.csel.io/

5. **Use a set of *git* commands to frequently backup your work.**

6. **Submit your assignment by pasting your repo link into the Assignment item**

on Canvas

(2) C++ review: program layout; single file

Single file compilation example:

someProgram.cpp

```
#include <iostream>
#include string;
using namespace std;
double foo(int x, string y);

int main(){
    double z;
    z = foo(7, "hello");
    return 0;
}

double foo(int x, string y){
    cout << y << endl;
    return x/2.0;
}
```

```
term$ g++ std=c++11 someProgram.cpp
```

(3) C++ review: program layout; multiple file

```
// headerFile.hpp
double foo(int x, string y);
```

```
// defs.cpp
#include <iostream>
#include string;
#include "headerFile.hpp"
using namespace std;

double foo(int x, string y){
    cout << y << endl;
    return x/2.0;
}
```

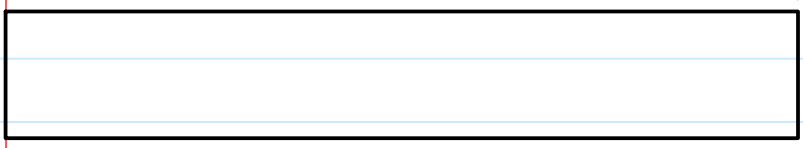
```
// driver.cpp
#include <iostream>
#include string;
#include "headerFile.hpp"
using namespace std;

int main(){
    double z;

    z = foo(7, "hello");

    return 0;
}
```

```
Compilation
$ g++ std=c++11 driver.cpp defs.cpp

Note: your assignment framework uses
CMake and makefile which automate this
process for you
```

(4) C++ review: arrays

Arrays:

- Gets a data type, just like any variable
- C++ arrays have to have a specified size
- The elements are contiguous in memory space



Usage:

```
// Declaration:
//    int arr[6];
// OR:
    int arr[] = {1,2,3,1,2,3};

// Writing to individual elements:
    arr[0] = 2270; // write a value to first element
    arr[6] = 12;  // BAD outside of array bounds
    arr[2] = 1.9;
// Reading from:
    int x;
    x = arr[2];
    cout << arr[5] << endl;

//As function arguments:
    void foo(int arr[], int size);  // declaration
// remember: need to pass array size
```

(5) C++ review: structures

IMPORTANT:
Classes vs Structs:
- by default all struct members are public

Structs:

A C++ struct allows us to store multiple variables in a single entity. It is a way to organize data.

E.G. Create a struct to store tree information (real-world tree, not a DS abstraction :))

Usage:

```
struct Tree{
    string species;
    double height;
    bool deciduous;
    int numberBranches;
    // Okay to have array as member of struct
    // must hard-code size.
    int lengthEachBranch[100];
};
```

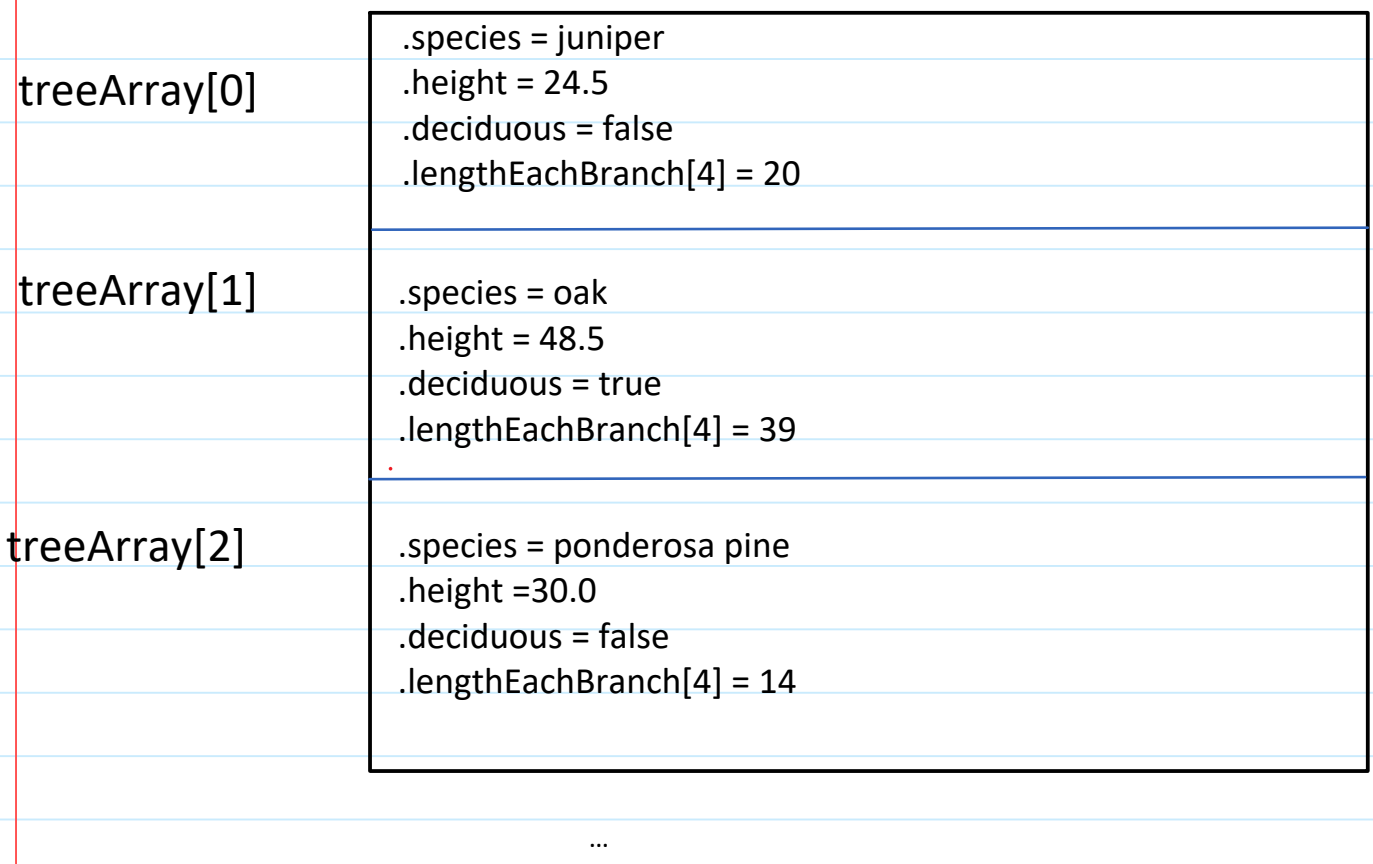
Struct usage:

Once a custom struct is defined, need a declaration in order to use it.

```
int main(){

    Tree t0;

    // use access operator to
    // write to individual members
    t0.species = "juniper";
    t0.height = 24.5;
    t0.deciduous = false;
    t0.numberBranches = 17;
    t0.lengthEachBranch[4] = 20;

...
}
```

(6) C++ review: array of structs

```
// Array of Structs
// declare array of length 5
Tree treeArray[5];
// store info of first element:
treeArray[0].species = "juniper";
treeArray[0].height = 24.5;
...
treeArray[0].lengthEachBranch[4] = 20;
```



(7) C++ review: external file input

How to read in external data files?

1) Use the <fstream> library
2. Declare a stream object
3. Connect stream object the external file
4. Read in data until a delimiter is reached
5. Repeat step 4 until desired data has been read into your program.

Read until end of file is reached

Method A) assume white-space delimiter

```
int x;
while(myInStream >> x)
    cout << x;
```

Method B) Custom delimiter

```
string arr[10];
for(int i=0; i<10; i++)
   getline(myInStream, arr[i], ',');
```

(8) C++ review: external file input example

Example:

Objective: read in file and store student info in an *array of structs*.

students.txt

```
444, Justine Henderson, Mechanical
933, Abdullah Lang, Aerospace
517, Marcus Cruz, Mechanical
262, Thalia Cobb, CompSci
935, Mathias Little, Aerospace
777, Eddie Randolph, Mechanical
215, Angela Walker, Mechanical
336, Jonathon Sheppard, Aerospace
870, Kamila Benton, Mechanical
551, Waylon Dalton, CompSci
```

Approach (single file):

1. Design a stuct for storing data.

```
struct Student{

// ???


};
```

2. Declare and array of structs.

3. Declare an input stream and connect with external file.

4. Decide on method for parsing the input stream

1. Loop step-4 x10

(9) C++ review: Cmd Line Arguments

Since main() is a function, it can be defined to take in arguments! (It can only be done in a specific way.)

By doing so, when the compiled program is executed, the user can pass information to the program.

```
// cmdLine.cpp
int main(int argc, char* argv[]){
    // now we can interact with user's console inputs

    // argc: number of inputs user called program with
    cout << "Number of terminal inputs: " << argc << end;

    // argv: array of char arrays
    for(int i = 0; i<argc; i++){
        cout << argv[i] << endl;
    }
}
```

```
in terminal:
term$ g++ cmdLine.cpp
term$ ./a.out
1
term$ ./a.out hello
2
hello
term$ ./a.out hello bob
3
hello
bob
```

(10) Putting it all together: file input + cmd line arguments

Given a text file "students.txt":

```
444, Justine Henderson, Mechanical
933, Abdullah Lang, Aeropsace
517, Marcus Cruz, Mechanical
262, Thalia Cobb, CompSci
935, Mathias Little, Aeropsace
777, Eddie Randolph, Mechanical
215, Angela Walker, Mechanical
336, Jonathon Sheppard, Aeropsace
870, Kamila Benton, Mechanical
551, Waylon Dalton, CompSci
```

Create a program that reads in the contents of the file and writes them to an array of structs. Each element of the array should contain all the info for one student (SID, name, major).

Requirements:

1. Program should take in the name of data file as a cmd line argument.
2. Throw an error message if user doesn't call program with name of file.
3. Use a function to read in file and populate array.

See L3b.cpp