Ass 1: Turtle Program

Turtle Graphics as part of [Logo](https://en.wikipedia.org/wiki/Logo_(programming_language) was one of the first languages explicitly designed for teaching beginning programming.

A TurtleProgram is a set of instructions directing the on-screen *turtle* to draw graphics. To draw a square, one would execute:

```
Forward 10 Right 90 Forward 10 Right 90 Forward 10 Right 90 Forward 10 Right 90
```

Other commands, such as PENUP, PENDOWN, PENCOLOR, etc, as well as conditionals and looping operators were later added to the language.

To practice dynamically allocated arrays and operator overloading, create a TurtleProgram class that supports the following functionality. For simplicity, we will use “F” instead of “Forward” and “R” instead of “Right”, but assume any string or series of strings is a valid program.

Create a class TurtleProgram to hold the strings that make up the program. The class will need to have:

1. Constructors with no parameter, one parameter and two parameters, copy constructor and destructor
2. Overload <<, so programs can be printed as below
3. Overload equality and inequality operators: operator== and operator!=. Two programs are == if all their instructions are the same.
4. Overload the following operators: operator=, operator+ and operator+=. Adding 2 programs creates a longer program.
5. Multiplying a program with an integer creates a larger program where the same program is repeated that many times. Write the * and *=operators. Multiplying by 0 or negative numbers is not defined. You can silently ignore the operation, throw an error or handle it in a different way. Similarly, multiplying two TurtleProgram’s is not defined.
6. getLength returns the number of strings in the program (i.e. for program [F 10 PENUP R 90 F 100] getLength returns 7.
7. getIndex returns the nth string in the program (i.e. for program [F 10 PENUP R 90 F 100] getIndex(0) would return "F", getIndex(1) would return "10", and so on. Trying to get the index of a negative number or beyond the length of the program is not defined. You can silently ignore the operation, throw an error or handle it in a different way.
8. setIndex is similar to getIndex, but is used to replace the string at the given index.

The data for the TurtleProgram must be in a *private dynamically allocated array of just the right size*. Normally, we would allocate a much larger array, but for this exercise, we are practicing dynamically resizing our data array.

```
int main()
{
    TurtleProgram tp1;
    cout << "tp1: " << tp1 << endl;
    TurtleProgram tp2("F", "10");
    cout << "tp2: " << tp2 << endl;
    TurtleProgram tp3("R", "90");
    tp1 = tp2 + tp3;
    cout << "tp1 now as tp2+tp3: " << tp1 << endl;
    tp1 = tp2 * 3;
    cout << "tp1 now as tp2 * 3: " << tp1 << endl;
    TurtleProgram tp4(tp1);
    cout << "tp4 is a copy of tp1: " << tp4 << endl;
    TurtleProgram tp5("F", "10");
    cout << "tp5: " << tp5 << endl;
    cout << boolalpha;
    cout << "tp2 and tp5 are == to each other: " << (tp2 == tp5) << endl;
    cout << "tp2 and tp3 are != to each other: " << (tp2 != tp3) << endl;
    cout << "index 0 of tp2 is " << tp2.getIndex(0) << endl;
    tp2.setIndex(0, "R");
    tp2.setIndex(1, "90");
    cout << "tp2 after 2 calls to setIndex: " << tp2 << endl;
    cout << "tp2 and tp3 are == to each other: " << (tp2 == tp3) << endl;
    // need to write additional tests for += *=
    cout << "Done." << endl;
    return 0;
}
```

output:

```
tp1: []
tp2: [F 10]
tp1 now as tp2+tp3: [F 10 R 90]
tp1 now as tp2 * 3: [F 10 F 10 F 10]
tp4 is a copy of tp1: [F 10 F 10 F 10]
tp5: [F 10]
tp2 and tp5 are == to each other: true
tp2 and tp3 are != to each other: true
index 0 of tp2 is F
tp2 after 2 calls to setIndex: [R 90]
tp2 and tp3 are == to each other: true
Done.
```

As always expected when programming, comment clearly and thoroughly. Clearly state any assumptions you make in the beginning comment block of the appropriate place, e.g., the class definition. Comments in the class definition file should describe the ADT, all functionality, and

assumptions so someone could use the class and understand behavior and restrictions. Pre and post conditions are fine, but not required. See the example on Assignments page for a well-documented program.

You do NOT need to handle data type errors. We are not executing the program, so any two strings would be acceptable parameters for constructors. You have to use the builtin array structure, not STL Array, vector or other expandable containers. You may not fix the array size at some large constant (although you may make an assumption that arrays larger than can be held in memory will not be handled). The point of this assignment is to review memory management; you will be allocating/deallocating memory often using *new* and *delete*.

I will run my own main to test your code. The main function provided doesn't test TurtleProgram fully, so you need to supplement it.

Write one function at a time. Test it before moving on to the next function. I suggest starting with operator== Use valgrind to check for memory leaks as you develop the program. Much easier to fix things early on.

Submit a single zip file, ass1.zip with the following files:

Class names start with capital letters, but file names are all lowercase for compatibility

turtleprogram.h

turtleprogram.cpp

ass1.cpp – your own testing functions and main

output.txt - the script file, as defined in Connecting and compiling files on linux labs

Once your code is working on your own machine, test it once more on the linux machines (you have been testing incrementally and using valgrind, right?). See Connecting and compiling files on linux labs

Under unix, compile your code using

```
g++ -std=c++14 -g -Wall -Wextra ass1.cpp turtleprogram.cpp -o ass1
```

and create the output.txt file following the instructions on that page.

Grading Rubric

See the general rubric on the assignments page. In addition, pay attention to the following:

1. private dynamically allocated array of correct size (-20)
2. Constructors: Empty, 1 parameter, 2 parameter and Copy Constructor
3. Destructor
4. <<
5. == and !=

6. =
7. + +=
8. * *=
9. getLength
10. setIndex / getIndex
11. memory leaks
12. efficiency and complexity
13. comments.txt - tested on CSS Linux Labs
14. Coding style + ass1.zip constructed properly