# ECE 275 Assignment 4

**DUE DATE:** Thursday, November 20, 11:59PM

In this assignment, you will learn how to provide a unit test for a single class from the Assignment 3. Your program must link against a library provided by the instructor to determine whether that library correctly implements the specifications for the class. You will be testing specifically 4 operations in the class[[1]](#footnote-1). This assignment is meant to encourage you to take a critical look at unit testing for future work.

The assignment name for this assignment is: **visittest**

## Command-line Arguments

Your program will be called without command line arguments.

visittest

Your program will not have a usage statement.

## Class Under Test

Your program will be testing the Visit class from Assignment 3. **Some changes have been made to the specification for the Visit class in order for you to verify your tests.** Whatever you learned about Visit in the previous assignment may not be true, so use the Visit class definition below, along with the following requirements for operations:

* The non-default constructor, should correctly initialize data values in the class; otherwise, tests for this operation fails[[2]](#footnote-2)
* The compare operation should return true if v1 is before v2 (case insensitive); if the strings are equal, then the operation returns true if the startTime is before the other Visit’s startTime; the test fails if these conditions are not met
* The operation addState should only add the state if it does not match the state in the back of the vector; otherwise the test fails.
* The operation getVisitString behaves as before if data are properly entered. However, if data are entered then it should react in the following way:
  + If the States vector is empty, then it must print
    - States: ERROR!
  + If the endTime is less than startTime, or either value is negative, then instead of printing the time it must print one of the below:
    - ERROR! to ERROR!
    - (number) to ERROR!
    - ERROR! to (number)
  + If the location is empty, then for the location information it must print
    - ERROR!
  + If any of these codes for ERROR! are not correct[[3]](#footnote-3), then the test fails.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

Visit class

This class is used to maintain an ordered list of the state of the

GPSIMU while within range of a specific location.

\*/

class Visit {

public:

// Constructor and Destructor

// Should be tested

Visit( std::string location, double startTime, double endTime );

// Need not be tested

~Visit(void);

// Should be tested

// Compare two visits together to see which is before the other.

// Returns true if v1 is before v2, false otherwise.

// A visit is before another if its location is alphabetically before

// the other visit’s location.

// A visit with the same location name as another is before that

// location if its startTime is before the other visit’s startTime.

static bool compare(Visit\* v1, Visit\* v2);

// Should be tested

// Adds this state; if the stateString matches the last state in the

// vector, then the operation returns without adding it to the vector

void addState( std::string stateString );

// Need not be tested

// returns the states vector

std::vector<std::string> getStates( ) const { return this->state; }

// Should be tested

// Returns the string for this visit. it looks as follows for a

// location of "4th Avenue" between 100.00000 and 101.00001 with

// states of [Init, Standby, Init]

// (Note: no trailing spaces)

//

// 4th Avenue

// 100.00000 to 101.00001

// States: Init, Standby, Init

//

// If the States vector is empty, then it must print ERROR!

// If the endTime is less than startTime, then instead of

// printing the time it must print ERROR!

// If the location is empty, it must print ERROR!

std::string getVisitString( ) const;

/\* Mutators below do not need to be tested \*/

void setStartTime( double startTime ) { this->startTime = startTime; }

double getStartTime( ) const { return this->startTime; }

void setEndTime( double endTime ) { this->endTime = endTime; }

double getEndTime( ) const { return this->endTime; }

void setLocation( std::string location ) { this->location = location; }

std::string getLocation( ) const { return this->location; }

private:

// Note that you can assume the week will always

// be the same for start and end times.

std::string location; // The location name for this visit.

double startTime; // The start seconds for the visit.

double endTime; // The end seconds for the visit.

std::vector<std::string> state; // The ordered list of states at this location.

};

#endif //VISIT\_H

## Output Format

Output should be to the standard out (i.e., cout). You should have a test for each operation mentioned above which prints as follows if all tests pass:

Beginning tests.

UNIT TEST PASSED: Visit()

UNIT TEST PASSED: compare()

UNIT TEST PASSED: addState()

UNIT TEST PASSED: getVisitString()

Tests complete.

If a test fails, it should print UNIT TEST FAILED: (operation identifier). Note: UNIT TEST FAILED or UNIT TEST PASSED is preceded by 3 spaces.[[4]](#footnote-4)

## CMake

We have included the source for a version of Visit.cpp and a skeleton CMakeLists structure in a template. We will be replacing the contents of Visit.cpp several times, and recompiling and rerunning your tests against varying versions of Visit.cpp, so please use *exactly* these key points (with your own comments, of course) as the src/CMakeLists.txt:

if( UNIX )

# Covers MacOS, Unix, and Linux (e.g., ece3)

set( CMAKE\_CXX\_FLAGS "-Wall -g" )

ELSE( UNIX )

# Windows-specific to prevent warnings that do not show up on ece3

set( CMAKE\_CXX\_FLAGS "/W3 /EHsc" )

ENDIF(UNIX)

file( GLOB SRCS \*.cpp \*.h )

add\_executable( visittest ${SRCS} )

1. Note: removed one of the operations, so used to say “5 operations” here. [↑](#footnote-ref-1)
2. This used to have a definition for a default constructor, which has been removed. [↑](#footnote-ref-2)
3. Rephrased to ensure meaning of test failure is clearer. [↑](#footnote-ref-3)
4. Note: Corrected an invalid “getVisitAsString( )” statement in the output from the tests here. [↑](#footnote-ref-4)