# REPORT OF MODIFICATION:

## IMPROVING RELIABILITY AND MODULARITY In REFACTORED CODE:

I improve reliability and modularity in refactored code in a sense I can change all the variable names with the sensible variable names. There are a few modular coding technique that I use in my day-to-day work that make my code a lot more organized and maintainable.

## improves software quality using robust:

We are improving software robustness with dependability cases we introduced new cases that are highly preferable for our software testing. Traditional approaches to reducing exception failures, such as code reviews, walkthroughs and formal testing, while very useful, are limited in their ability to address a core problem: the programmer’s inadequate coverage of exceptional conditions. The problem of coverage might be rooted in cognitive factors that impede the mental generation (or recollection) of exception cases that would pertain in a particular situation, resulting in insufficient software robustness. This paper describes a study to test the hypothesis that robustness for exception failures can be improved through the use of dependability cases.

## Using nested functions to encapsulate blocks of code:

We are using nested functions in our code for better performance and information hiding the benefit of these nested functions are that the data which are passing through functions that are hidden from one another. We realize that some of the code looks repetitive, so We decide to create another function.

## tools and techniques:

The tools and techniques that we are using in code refactoring is We use two tools that are given below:

* DEV C++
* TERMINAL OF UBUNTO

## WHAT ISSUES OCCURS DURING IMPLEMENTION:

The one of the biggest issues is that we are facing is how we can implement all the things reliably and the other issues are given below:

* Selecting test cases
* Not enough automation
* Lack of right testing tools and environment
* Not achieving the testing results

## How we resolve these issues:

We resolve these issues one by one and through the tool of dev c++ we are getting close to find out accurate results. We have writing a lot of test cases and run all of these one by one which is suitable we can add it and which are not we are removing it.

# SUMMARY:

The first thing which we learn in this assessment is that how we use better pair programming. In the source code we modified variable names into better understandable variable name and using pascal naming convention during writing the code. And we remove some test case that are not suitable for our code and we introduced some best test cases in source code. For example: We introduced the test case that checks the missing element between the route name and the user entered name and which means every time user entered and route name must be same and also available in map. And so, we introduced the test case that finds the nearby nodes in map if the user entered name is so far than the first edge there is an option to collect nearby point rather than actual point. We are using separate test suits and these test suit contains many other test cases that is suitable in this test suit. We are using best testing tools for creating test cases. We are trying to be fixing bugs in this test suits. It is easy to set up and use. It can apply an arbitrary number of test cases to the unit. Here, a test case is a list of input values, with an associated list of expected outputs. Each test case is a separate instance of the same test, not a new kind of test. After applying input values to the unit, the test tool compares the outputs of the unit against the list of expected outputs. It provides a simple success/fail indication for each test case. It is a portable and powerful, yet simple, unit testing framework for handling automated tests in C++. The focus lies on usability and extendability. Several output formats are supported and new ones are easily added. The programmer does not need to study the outputs to see if the unit passed a particular test case. It allows programmers to add, modify, or delete test cases without triggering a massive compile.

# Conclusion:

If want to test functions with more than three input or output parameters, you'll have to play some special tricks when you implement the function apply. Note that you could conceivably pass vector arguments to apply and translate them to the parameters required by the unit under test. Alternatively, you can pass string arguments that combine the required parameters as a single string. This can get a little messy, of course. In this modification code we simply convert the direction of code into easier and more introduced best testcases and test suites which are highly suitable for this gpx master file. Testcases names has probably defines what the actual functionality performs in it.