Mutation Testing Simulator
Final Project – Fall 2019
Software Validation
ECSE 429

In this project you are asked to design a mutation test simulator. The goal of your simulator is to test software program for potential mutations due to programming typo errors. You can write your simulator in any programming language. In this project you are considering mutations of arithmetic operations. More specifically, the mutants are: “+”, “-“. “*”, “/” (addition, subtraction, multiplication and division).

In your project you must achieve the following mile stones:

a) Mutant list generation
b) Mutant injection
c) Killing mutants (fault simulation)
d) Parallel simulation

1. Mutant list generation

This is the first element of your project. You write a program, which accepts you fault-free software as an input. Your software is traversed line by line from primary inputs to primary outputs. If the currently processed line contains an arithmetic operation “=”, “-“, “*” or “/” then the line number and arithmetic operation are stored in fault list file. In addition, the three mutants, which are to be considered later are generated in association with the fault free arithmetic operation encountered in processed software.

After the traversal of the software under test (SUT) is completed the fault list file should store the complete information about all sites of potential mutant injection into the SUT as well as listing of types of mutants to be injected in each location.

2. Mutant injection

In this part of the project you perform a single mutant injection into your SUT. Each mutant from mutant fault list generated in part 1 must be considered.

3. Killing mutants (fault simulation)

Prepare a simulation file, which you are going to use for mutant killings. You should design a simulation file appropriately to SUT. For example, you could consider boundary values as a part of simulation space. You use your simulation file to kill mutants from your mutant list. After simulation of a mutant is completed you mark in the mutant list whether a given mutant was killed or not, and if killed then by what vector.

Note, when you are simulating SUT with a mutant, you must compare the simulation results to the analogous simulations of mutant-free (original) software.

After all mutants are considered, you generate the mutant coverage of your SUT. Mutant coverage is the ratio of all killed mutants to all mutants in mutant list.

4. Parallel simulations

Simulation of your mutants was performed one at a time. Devise a method to speed up the procedure by parallel mutant simulation. Consider simulating in parallel 3 mutants.

5. Additional information

a. SUT

You select your software under test. SUT can be written in any language of your choice, however, it must be compatible with your testing platform. As you are designing your test suites (part 3) then you must be familiar with application of your SUT. Although you do not need to write your own SUT for this project (you can use software written in other courses or some other source of software). However, you must provide a good description of what is SUT implementing, and further on, you must derive a specification of the application for which your SUT was designed.

You should incorporate information about your SUT behavior into designing your test suites.

b. Programming language

You can implement the project using programming language of your choice. All your team members must be familiar with programming language of your choice.

Further, you must make sure, that you will be able to demo your project at McGill (in case you select the programming platform not supported by McGill tools).

6. Deliverables

There are two deliverables of this project:

a. Demo: you have to demo your project in the scheduled time (last part of the course),
b. Report: you will submit one report per group at the end of the semester. Requirements regarding the report will be posted latter in the course

7. Grading

a. Mutation Testing Simulator:	80%
b. Report:	20%