The project purpose is to perform symbolic execution of SimpleC programs to generate rewritten code with symbolic variables and to generate path conditions of the program. For this, the input variables will be reassigned as symbolic variables, so that these abstract variables cover multiple possible instances of the program in relation to the input. Then the program will be rewritten in terms of these abstract variables, and this will be an output. As well, a binary tree will be generated based on the symbolic variables where every node will be a formula where the unknown variables will appear as symbolic variables. In the binary tree every path is a possible instance of the program, that path is a conjunction of the formulas that make it. These paths will also be part of the output.

The formulas considered will only be the ones of the if conditions. That is, the possible instances of the program will be determined only by the branches generated by the if conditions. Then, the path coverage based on the symbolic variables will only be determined by the if conditions in terms of the symbolic variables. When a condition is reached the binary tree will add at the current node selected, or at all the pertaining nodes, new child nodes for both if the formula were to happen, or if the negation were to. For example, if the condition in the original SimpleC program is $(x > 2)$ then both $(x > 2)$ and $NOT(x > 2)$ will be added as child nodes.

The reassignments to the variables that are mapped as symbolic variables may change, under the following conditions: the variable has been re-assigned to a non-variable value, for instance a literal, 4, then the variable associated will no longer be considered a symbolic variable in the rest of that scope; or, the variable has been used to store a new input (from input instruction), in this case a new symbolic variable will be used. Literal values considered in the program are also expressions as: “4+2+1-1”.

The scopes are separated by functions. As well, the scopes are created and treated with different values in nested “if then else” statements. Every scope keeps a version of both its current symbolic variables and non-symbolic ones. Therefore, the program will allow parallel and nested if-then-else statements.

The way to show correctness and replicability will be through running the set of test cases and compare it with their outputs provided along with this document.

As well the following table describes the purpose of each test case.

Test case	**Purpose**
18paths-nests-and-parallel.simplec	The output for this test case shows that the generation of paths supports nested and parallel conditions, with proper use of the scope. This generates 18 paths.
sixpaths.simplec	Shows the same as the previous test but only with 6 paths.
many-symbs.simplec	Shows that symbolic variables can be created on demand without repetition as they are based on the program counter, number of lines of code relative to the rewritten program.
not-sat1.simplec	The output for this program shows a clear problem in one instance of the program, that is, a contradiction in

	Path1: symbol0>3 symbol0==1 Is not satisfiable, hence the name.
sat1.simplec	Although the name suggest that this is satisfiable unlike the previous test, it is not, since it's essentially the previous test with the second condition more or less negated, x>3 in the previous to x<3 in the current. Here path 2 catches the contradiction: Path N2: !(symbol0<3) symbol0==1 This shows that the generator can be useful to catch less obvious absurdities in the code.
symb-sim-var.simplec	In the rewritten program output, shows the reassignment of variables where the variables associated to symbols take literal values later. Also shows the mixing of symbol variables in many operations.
symb-sim-var-outside-nest.simplec	In the rewritten program output, shows the same as the previous test plus the effect happening only inside their current "if-then-else" scopes. Outside it behaves unaffected of the code not guaranteed to reach.

To sum up:

Input:

- A SimpleC program.

Output:

- The SimpleC program rewritten in terms of symbolic variables.
    - Instructions will be numerated, and the end of compound statements of if-then-else structures will be numerated as a comment at the end.
- The paths, made up of formulas using symbolic variables based on "if then else" conditions, will be printed enumerated.
    - Each path will be printed from a leaf up to the root of the binary tree of conditions.
    - Each path represents a concatenation of the formulas.

Things that were originally planned but won't or couldn't be completely covered for the due date:

- Support branching of while loops to avoid path explosion and so either with default or user defined time/execution constraints or based on loop invariants.
- Independent generation of the output for all functions of the program. Currently, it only works completely for a single function at a time, but the program can be divided into different units by their functions and be covered in such way, although not automatically.
- Assertions that the user could place in points in the code, to be evaluated in relation to the symbolic values
- Automatic translation of the conjunctive formulas (paths from each node) to a popular SMT solver such as Z3.
- With the result of the satisfiability solver the program will print a test case that shows the failure if it was found. Otherwise, it should print that the program is correct.