Mutation Testing Report

Suyash Ajit Chavan (IMT2021048) Deepkumar Patel (IMT2021011)
Team : Gene-X

November 23, 2024

GitHub Repository

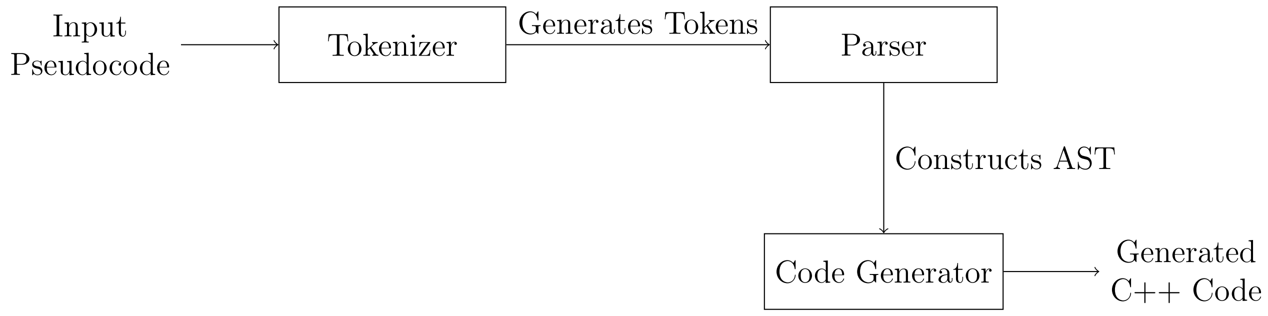
https://github.com/Suyashchavan10/Software_Testing_Project

Contents

1 Project Overview

Source-Code Language : C++

This project converts pseudocode into C++ code through three main stages: tokenization, abstract syntax tree (AST) generation, and code generation. The system consists of the following components:



1.1 Tokenizer

The tokenizer processes the input pseudocode and generates tokens. Tokens represent atomic elements of the pseudocode such as keywords, identifiers, operators, and literals.
Key Features:

- Identifies numbers, strings, keywords, and operators.
- Classifies tokens into predefined types.

1.2 Parser

The parser uses the tokens to construct an Abstract Syntax Tree (AST). The AST represents the hierarchical structure of the pseudocode.
Key Features:

- Handles constructs like declarations, assignments, and loops.
- Validates the syntax of the pseudocode.

1.3 Code Generator

The code generator translates the AST into valid C++ code.
Key Features:

- Supports variable declarations, conditionals, loops, and assignments.
- Generates output in a consistent and readable format of C++ code.

1.4 Example Conversion of Pseudocode to C++ code

Pseudocode

Pseudocode

```
Declare x As Integer
Declare y As Integer
Declare result As Integer
Declare arr As Array Of Integer[5]
Declare matrix As Array Of Integer[3][3]
Declare str As String
Declare concatResult As String

Assign x = 5
Assign y = x + 5 * 3
Print result

If result > 10 Then
    If x = 3+4 Then
        Assign x = 3*3
    End If
Else
    If x < 23 Then
        Print "Result is 10 or less"
        Assign x = 10
    Else
        Declare z As Integer
        Assign z = 234
    End If
End If

For i=0 To 2 Do
    For j=0 To 2 Do
        If i = 0 Then
            Print i
        Else
            Print j
        End If
    End For
End For

While i < 4 Do
    If i = 0 Then
        Print "Zero"
    Else
        Print "NoZero"
    End If
    Assign i = i + 1
End While
```

Generated C++ Code

Generated C++ Code

```
#include <bits/stdc++.h>
using namespace std;

int main() {

    int x;
    int y;
    int result;
    int arr[5];
    int matrix[3][3];
    string str;
    string concatResult;
    int i;
    x = 5 ;
    y = x + 5 * 3 ;
    cout << result << endl;

    if ( result > 10 ) {
        if ( x = 3 + 4 ) {
            x = 3 * 3 ;
        }
    }
    else {
        if ( x < 23 ) {
            cout << "Result␣is␣10␣or␣less" << endl;
            x = 10 ;
        }
        else {
            int z;
            z = 234 ;
        }
    }

    for (int i = 0 ; i <= 2 ; i++) {
        for (int j = 0 ; j <= 2 ; j++) {
            if ( i = 0 ) {
                cout << i << endl;
            }
            else {
                cout << j << endl;
            }
        }
    }

    while (i < 4 ) {
        if ( i = 0 ) {
            cout << "Zero" << endl;
        }
        else {
            cout << "NoZero" << endl;
        }

        i = i + 1 ;
    }

    return 0;
}
```

2 Introduction to Software Testing

Software testing ensures the correctness, robustness, and quality of software. There are several types of testing, each targeting different aspects of software validation. Below are the key testing methods employed in this project:

2.1 Unit Testing

Unit testing focuses on verifying the correctness of individual functions or methods. It isolates each unit of the codebase and ensures that it behaves as expected. Mocks and stubs are often used to simulate dependencies.

2.2 Integration Testing

Integration testing ensures that multiple units of the code interact correctly. It validates interfaces and dependencies, ensuring the software functions as a cohesive system.

2.3 Mutation Testing

Mutation testing evaluates the quality of test cases by introducing small changes (mutations) into the code and verifying if the test suite detects these changes.

- **Unit Mutation Testing:** Focuses on mutating individual units (functions or methods).
- **Integration Mutation Testing:** Targets interactions between integrated units.

3 Tools Used

3.1 GoogleTest for Unit Testing

GoogleTest, also known as GTest, is a powerful framework for writing and running C++ unit tests. It provides a comprehensive and user-friendly environment to ensure the correctness of individual components in the project.

Key Features:

- **Assertion Mechanisms:** Offers a variety of assertions like `ASSERT_EQ`, `EXPECT_EQ`, and more to validate test outcomes.
- **Test Case Management:** Organizes tests into suites for better readability and maintenance.

3.2 Mull for Mutation Testing

Mull is a mutation testing tool specifically designed for C and C++ projects. It automates the process of introducing mutations, running test cases, and analyzing results to measure the effectiveness of test suites.

Key Features:

- **Automated Mutation Generation:** Applies mutations like arithmetic operator changes, logical operator changes, and constant modifications.
- **Detailed Reporting:** Categorizes mutations into killed, survived, and ignored, providing insights into test coverage gaps.

4 Unit Testing using GoogleTest

Separate unit testing has been implemented for each of the three core classes: ‘Tokenizer‘, ‘Parser‘, and ‘CodeGenerator‘. These tests ensure the correctness and robustness of each individual component of the system in isolation.

4.1 Test Tokenizer

1. Make unit testcases in test_tokenizer.cpp file.

test_tokenizer.cpp

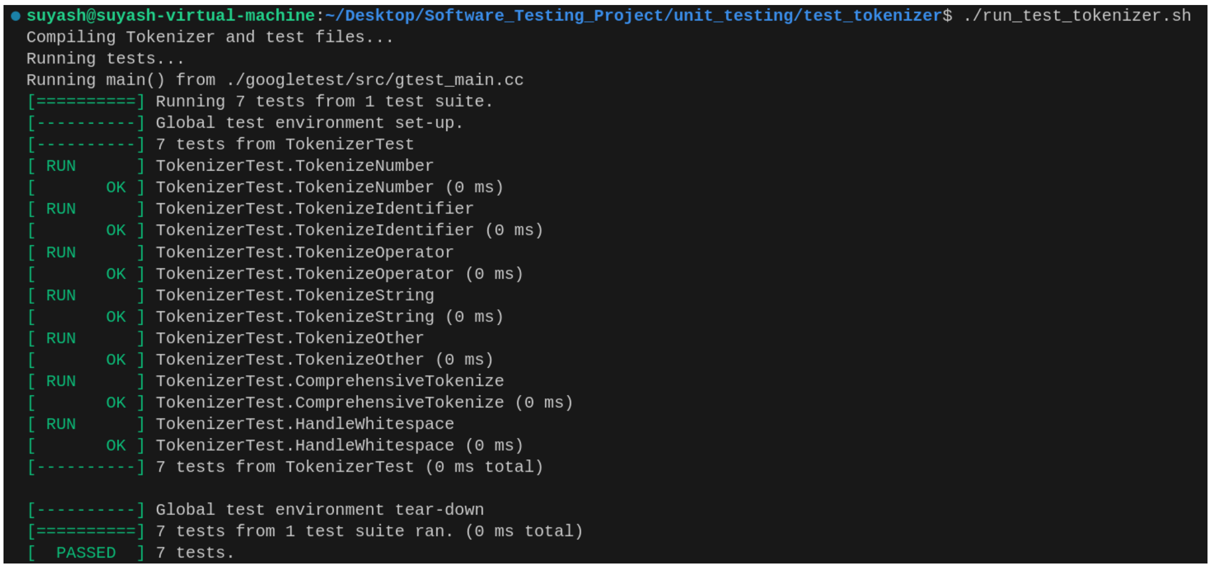
```
#include "../../src/tokenizer/tokenizer.h"
#include <gtest/gtest.h> // GoogleTest header
using namespace std;

// Test tokenizeNumber() method
TEST(TokenizerTest, TokenizeNumber) {
    string input = "12345";
    Tokenizer tokenizer(input);
    auto tokens = tokenizer.tokenize();

    ASSERT_EQ(tokens.size(), 2); // One number token + END_OF_FILE
    EXPECT_EQ(tokens[0].type, TokenType::NUMBER);
    EXPECT_EQ(tokens[0].lexeme, "12345");
}

//Similarly, include remaining testcases
```

2. Run above unit testcases using following bash script.

run_test_tokenizer.sh

```
#!/bin/bash

# Ensure the script stops on any error
set -e

# Define paths for source files and the output executable
TOKENIZER_SRC="../../src/tokenizer/tokenizer.cpp"
TEST_TOKENIZER_SRC="test_tokenizer.cpp"
OUTPUT_EXEC="tokenizer_test"

# Define the path to GoogleTest
GTEST_INCLUDE_PATH="/usr/include/gtest"
GTEST_LIB_PATH="/usr/lib/x86_64-linux-gnu"

# Step 1: Compile the source files and tests
echo "Compiling␣Tokenizer␣and␣test␣files..."
g++ -std=c++11 -isystem $GTEST_INCLUDE_PATH -pthread $TOKENIZER_SRC
$TEST_TOKENIZER_SRC -lgtest -lgtest_main
-o $OUTPUT_EXEC -L$GTEST_LIB_PATH

# Step 2: Run the tests
echo "Running␣tests..."
./$OUTPUT_EXEC
```

3. Output and results are as follows:



```
suyash@suyash-virtual-machine:~/Desktop/Software_Testing_Project/unit_testing/test_tokenizer$ ./run_test_tokenizer.sh
Compiling Tokenizer and test files...
Running tests...
Running main() from ./googletest/src/gtest_main.cc
[==========] Running 7 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 7 tests from TokenizerTest
[ RUN      ] TokenizerTest.TokenizeNumber
[       OK ] TokenizerTest.TokenizeNumber (0 ms)
[ RUN      ] TokenizerTest.TokenizeIdentifier
[       OK ] TokenizerTest.TokenizeIdentifier (0 ms)
[ RUN      ] TokenizerTest.TokenizeOperator
[       OK ] TokenizerTest.TokenizeOperator (0 ms)
[ RUN      ] TokenizerTest.TokenizeString
[       OK ] TokenizerTest.TokenizeString (0 ms)
[ RUN      ] TokenizerTest.TokenizeOther
[       OK ] TokenizerTest.TokenizeOther (0 ms)
[ RUN      ] TokenizerTest.ComprehensiveTokenize
[       OK ] TokenizerTest.ComprehensiveTokenize (0 ms)
[ RUN      ] TokenizerTest.HandleWhitespace
[       OK ] TokenizerTest.HandleWhitespace (0 ms)
[----------] 7 tests from TokenizerTest (0 ms total)

[----------] Global test environment tear-down
[==========] 7 tests from 1 test suite ran. (0 ms total)
[  PASSED  ] 7 tests.
```

Figure 1: Results of test_tokenizer

4.2 Test Parser

1. Make unit testcases in test_parser.cpp file.

test_parser.cpp

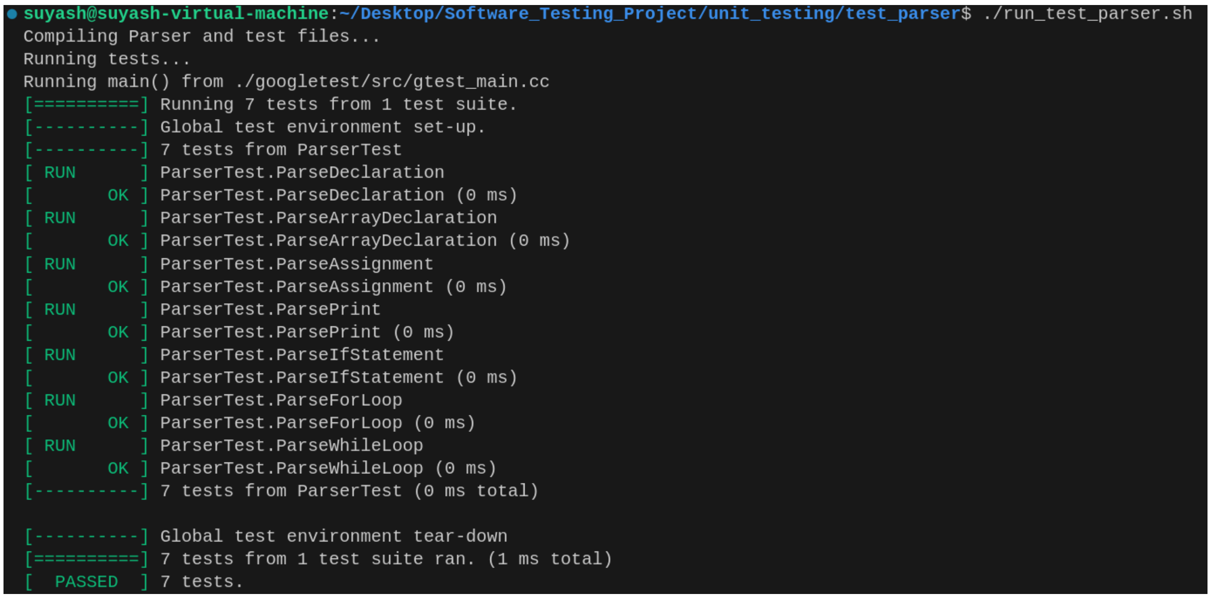
```cpp
#include "../../src/tokenizer/tokenizer.h"
#include "../../src/parser/parser.h" // Header for the Parser class
#include <gtest/gtest.h> // GoogleTest header
using namespace std;

// Test parsing declarations
TEST(ParserTest, ParseDeclaration) {
    string input = "Declare␣x␣As␣Integer";
    auto tokens = tokenizeInput(input);
    Parser parser(tokens);

    Node ast = parser.parse();
    ASSERT_EQ(ast.children.size(), 1);
    Node declaration = ast.children[0];

    EXPECT_EQ(declaration.type, NodeType::DECLARATION);
    ASSERT_EQ(declaration.children.size(), 2); // Identifier and Type
    EXPECT_EQ(declaration.children[0].token.lexeme, "x");
    EXPECT_EQ(declaration.children[1].token.lexeme, "Integer");
}

//Similarly, include remaining testcases
```

2. Run above unit testcases using following bash script.

run_test_parser.sh

```bash
#!/bin/bash

# Define paths for source files and the output executable
PARSER_SRC="../../src/parser/parser.cpp"
TEST_PARSER_SRC="test_parser.cpp"
OUTPUT_EXEC="parser_test"

# Define the path to GoogleTest
GTEST_INCLUDE_PATH="/usr/include/gtest"
GTEST_LIB_PATH="/usr/lib/x86_64-linux-gnu"

# Step 1: Compile the source files and tests
echo "Compiling␣Parser␣and␣test␣files..."
g++ -std=c++11 -isystem $GTEST_INCLUDE_PATH -pthread
$PARSER_SRC $TEST_PARSER_SRC -lgtest -lgtest_main
-o $OUTPUT_EXEC -L$GTEST_LIB_PATH

# Step 2: Run the tests
echo "Running␣tests..."
./$OUTPUT_EXEC
```

3. Output and results are as follows:



```
suyash@suyash-virtual-machine:~/Desktop/Software_Testing_Project/unit_testing/test_parser$ ./run_test_parser.sh
Compiling Parser and test files...
Running tests...
Running main() from ./googletest/src/gtest_main.cc
[==========] Running 7 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 7 tests from ParserTest
[ RUN      ] ParserTest.ParseDeclaration
[       OK ] ParserTest.ParseDeclaration (0 ms)
[ RUN      ] ParserTest.ParseArrayDeclaration
[       OK ] ParserTest.ParseArrayDeclaration (0 ms)
[ RUN      ] ParserTest.ParseAssignment
[       OK ] ParserTest.ParseAssignment (0 ms)
[ RUN      ] ParserTest.ParsePrint
[       OK ] ParserTest.ParsePrint (0 ms)
[ RUN      ] ParserTest.ParseIfStatement
[       OK ] ParserTest.ParseIfStatement (0 ms)
[ RUN      ] ParserTest.ParseForLoop
[       OK ] ParserTest.ParseForLoop (0 ms)
[ RUN      ] ParserTest.ParseWhileLoop
[       OK ] ParserTest.ParseWhileLoop (0 ms)
[----------] 7 tests from ParserTest (0 ms total)

[----------] Global test environment tear-down
[==========] 7 tests from 1 test suite ran. (1 ms total)
[  PASSED  ] 7 tests.
```

Figure 2: Results of test_parser

4.3 Test CodeGenerator

1. Make unit testcases in test_codeGenerator.cpp file.

test_codeGenerator.cpp

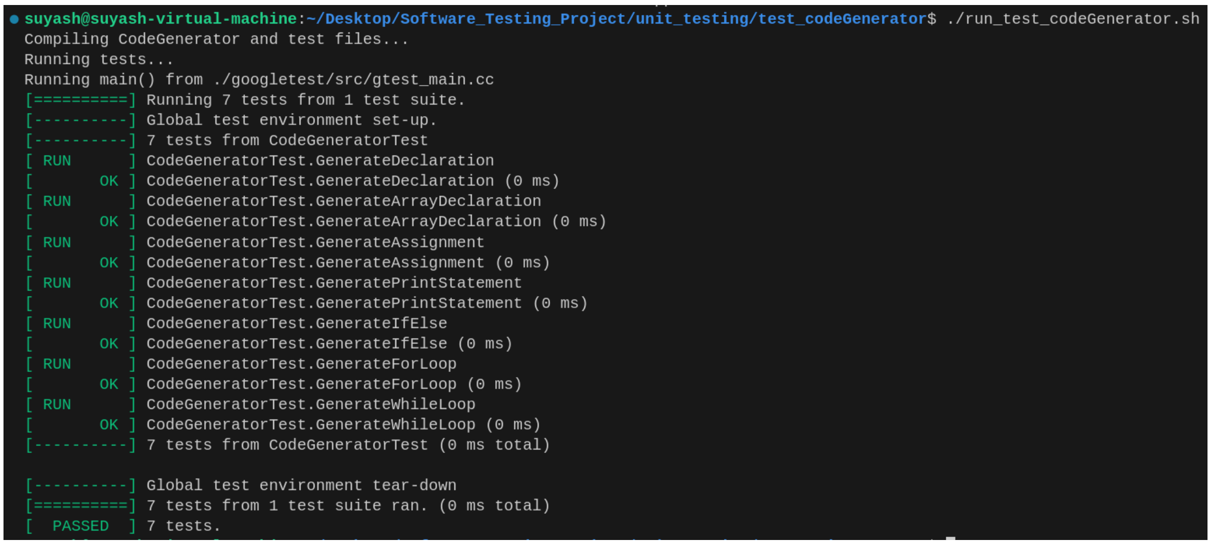
```
#include "../../src/codeGenerator/codeGenerator.h"
#include "../../src/parser/parser.h" // Header for the Parser class
#include <gtest/gtest.h> // GoogleTest header
using namespace std;

// Test code generation for variable declarations
TEST(CodeGeneratorTest, GenerateDeclaration) {
    string input = "Declare␣x␣As␣Integer";
    Node ast = parseInput(input);
    CodeGenerator generator;
    string generatedCode = generator.generateCode(ast);

    string expectedCode = R"(
␣␣␣␣␣␣␣␣#include␣<bits/stdc++.h>
␣␣␣␣␣␣␣␣using␣namespace␣std;
␣␣␣␣␣␣␣␣int␣main()␣{
␣␣␣␣␣␣␣␣␣␣␣␣int␣x;
␣␣␣␣␣␣␣␣␣␣␣␣return␣0;
␣␣␣␣␣␣␣␣}
␣␣␣␣)";
    EXPECT_EQ(normalizeWhitespace(generatedCode),
    normalizeWhitespace(expectedCode));
}

//Similarly, include remaining testcases
```

2. Run above unit testcases using following bash script.

run_test_codeGenerator.sh

```
#!/bin/bash

# Define paths for source files and the output executable
CODEGENERATOR_SRC="../../src/codeGenerator/codeGenerator.cpp"
TEST_CODEGENERATOR_SRC="test_codeGenerator.cpp"
OUTPUT_EXEC="codeGenerator_test"

# Define the path to GoogleTest
GTEST_INCLUDE_PATH="/usr/include/gtest"
GTEST_LIB_PATH="/usr/lib/x86_64-linux-gnu"

# Step 1: Compile the source files and tests
echo "Compiling␣CodeGenerator␣and␣test␣files..."
g++ -std=c++11 -isystem $GTEST_INCLUDE_PATH -pthread
$CODEGENERATOR_SRC $TEST_CODEGENERATOR_SRC -lgtest -lgtest_main
-o $OUTPUT_EXEC -L$GTEST_LIB_PATH

# Step 2: Run the tests
echo "Running␣tests..."
./$OUTPUT_EXEC
```

3. Output and results are as follows:



```
● suyash@suyash-virtual-machine:~/Desktop/Software_Testing_Project/unit_testing/test_codeGenerator$ ./run_test_codeGenerator.sh
  Compiling CodeGenerator and test files...
  Running tests...
  Running main() from ./googletest/src/gtest_main.cc
  [==========] Running 7 tests from 1 test suite.
  [----------] Global test environment set-up.
  [----------] 7 tests from CodeGeneratorTest
  [ RUN      ] CodeGeneratorTest.GenerateDeclaration
  [       OK ] CodeGeneratorTest.GenerateDeclaration (0 ms)
  [ RUN      ] CodeGeneratorTest.GenerateArrayDeclaration
  [       OK ] CodeGeneratorTest.GenerateArrayDeclaration (0 ms)
  [ RUN      ] CodeGeneratorTest.GenerateAssignment
  [       OK ] CodeGeneratorTest.GenerateAssignment (0 ms)
  [ RUN      ] CodeGeneratorTest.GeneratePrintStatement
  [       OK ] CodeGeneratorTest.GeneratePrintStatement (0 ms)
  [ RUN      ] CodeGeneratorTest.GenerateIfElse
  [       OK ] CodeGeneratorTest.GenerateIfElse (0 ms)
  [ RUN      ] CodeGeneratorTest.GenerateForLoop
  [       OK ] CodeGeneratorTest.GenerateForLoop (0 ms)
  [ RUN      ] CodeGeneratorTest.GenerateWhileLoop
  [       OK ] CodeGeneratorTest.GenerateWhileLoop (0 ms)
  [----------] 7 tests from CodeGeneratorTest (0 ms total)

  [----------] Global test environment tear-down
  [==========] 7 tests from 1 test suite ran. (0 ms total)
  [  PASSED  ] 7 tests.
```

Figure 3: Results of test_codeGenerator

5 Mutation Testing using Mull

5.1 Objective

Mutation testing measures the effectiveness of the test suite by introducing deliberate errors (mutations) into the code. Separate mutation testing has been implemented for each of the three core classes: ‘Tokenizer‘, ‘Parser‘, and ‘CodeGenerator‘.

5.2 Testing Workflow

1. Original and mutated code were compiled.
2. Tests were executed against both versions.
3. Test failures confirmed the mutation’s detection.

5.3 Test Tokenizer

1. Make mutation testcases in test_tokenizer.cpp file.

test_tokenizer.cpp

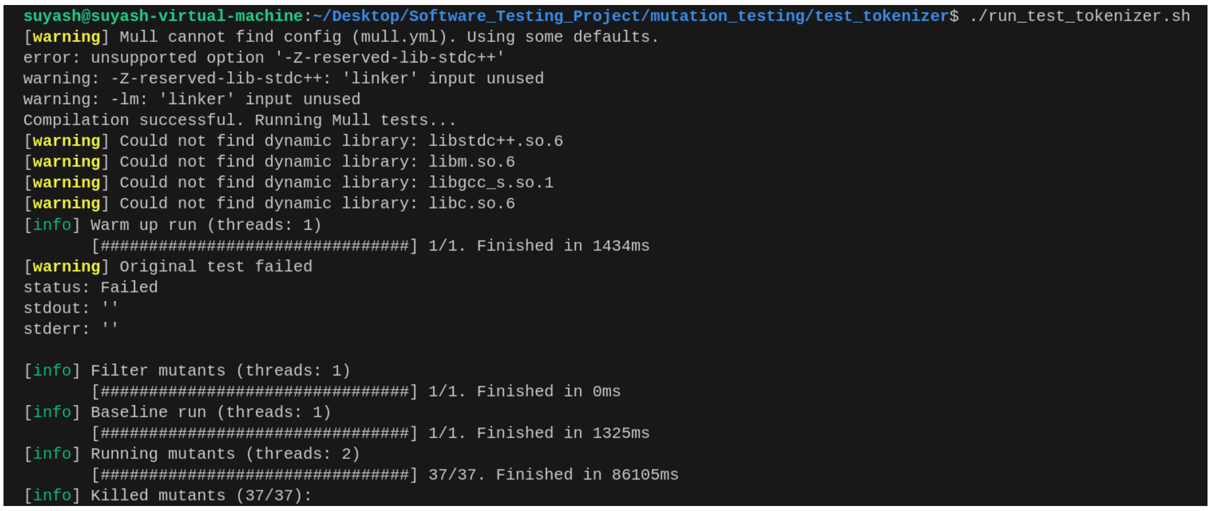
```
#include "../../src/tokenizer/tokenizer.cpp"
#include <iostream>
#include <vector>
#include <string>
using namespace std;

int main() {
    // Test Case 1: Tokenize numbers
    {
        string input = "12345";
        Tokenizer tokenizer(input);
        vector<Token> tokens = tokenizer.tokenize();

        if (tokens.size() != 2 || !checkToken(tokens[0],
        TokenType::NUMBER, "12345")) {
            return 1; // Test failed
        }
    }
}

//Similarly, include remaining testcases
```

2. Run above mutation testcases using following bash script.

run_test_tokenizer.sh

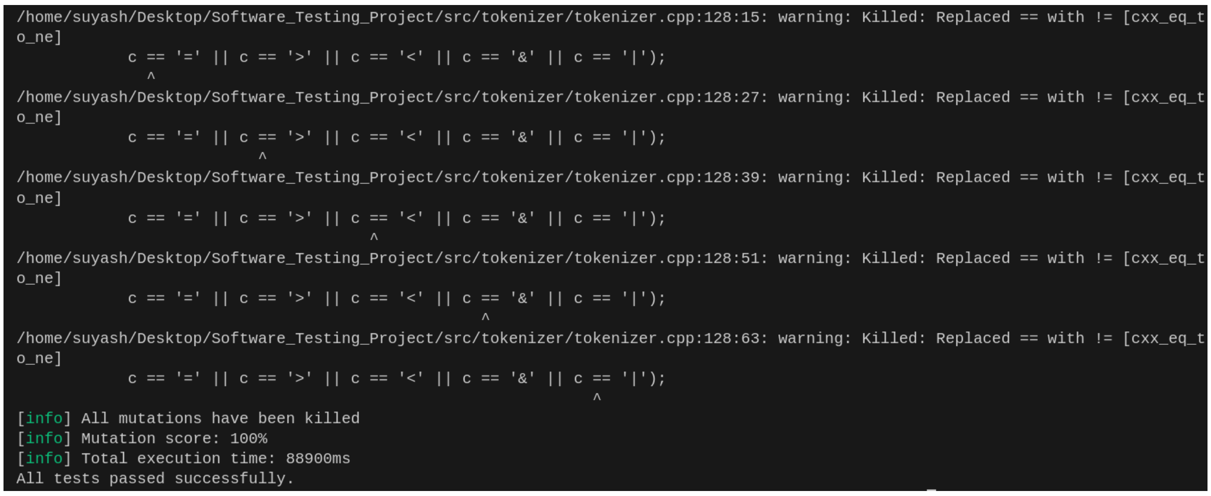
```bash
#!/bin/bash

# Define paths for source files and the output executable
TOKENIZER_SRC="../../src/tokenizer/tokenizer.cpp"
TEST_TOKENIZER_SRC="test_tokenizer.cpp"
OUTPUT_EXEC="tokenizer_test"

# Step 1: Compile the source and test files with clang-12 and Mull
clang-12 -fexperimental-new-pass-manager \
         -fpass-plugin=/usr/lib/mull-ir-frontend-12 \
         -stdlib=libstdc++ \
         -g -grecord-command-line \
         $TEST_TOKENIZER_SRC -o $OUTPUT_EXEC -lstdc++ -lm

# Check if compilation was successful
if [[ $? -ne 0 ]]; then
    echo "Compilation␣failed.␣Exiting..."
    exit 1
fi

echo "Compilation␣successful.␣Running␣Mull␣tests..."

# Step 2: Run the compiled executable with Mull runner
mull-runner-12 -ide-reporter-show-killed $OUTPUT_EXEC
exit 0
```

3. Output and results are as follows:



```
suyash@suyash-virtual-machine:~/Desktop/Software_Testing_Project/mutation_testing/test_tokenizer$ ./run_test_tokenizer.sh
[warning] Mull cannot find config (mull.yml). Using some defaults.
error: unsupported option '-Z-reserved-lib-stdc++'
warning: -Z-reserved-lib-stdc++: 'linker' input unused
warning: -lm: 'linker' input unused
Compilation successful. Running Mull tests...
[warning] Could not find dynamic library: libstdc++.so.6
[warning] Could not find dynamic library: libm.so.6
[warning] Could not find dynamic library: libgcc_s.so.1
[warning] Could not find dynamic library: libc.so.6
[info] Warm up run (threads: 1)
       [################################] 1/1. Finished in 1434ms
[warning] Original test failed
status: Failed
stdout: ''
stderr: ''

[info] Filter mutants (threads: 1)
       [################################] 1/1. Finished in 0ms
[info] Baseline run (threads: 1)
       [################################] 1/1. Finished in 1325ms
[info] Running mutants (threads: 2)
       [################################] 37/37. Finished in 86105ms
[info] Killed mutants (37/37):
```

Figure 4: Results of test_tokenizer 1



```
/home/suyash/Desktop/Software_Testing_Project/src/tokenizer/tokenizer.cpp:128:15: warning: Killed: Replaced == with != [cxx_eq_t
o_ne]
            c == '=' || c == '>' || c == '<' || c == '&' || c == '|');
              ^
/home/suyash/Desktop/Software_Testing_Project/src/tokenizer/tokenizer.cpp:128:27: warning: Killed: Replaced == with != [cxx_eq_t
o_ne]
            c == '=' || c == '>' || c == '<' || c == '&' || c == '|');
                          ^
/home/suyash/Desktop/Software_Testing_Project/src/tokenizer/tokenizer.cpp:128:39: warning: Killed: Replaced == with != [cxx_eq_t
o_ne]
            c == '=' || c == '>' || c == '<' || c == '&' || c == '|');
                                      ^
/home/suyash/Desktop/Software_Testing_Project/src/tokenizer/tokenizer.cpp:128:51: warning: Killed: Replaced == with != [cxx_eq_t
o_ne]
            c == '=' || c == '>' || c == '<' || c == '&' || c == '|');
                                                  ^
/home/suyash/Desktop/Software_Testing_Project/src/tokenizer/tokenizer.cpp:128:63: warning: Killed: Replaced == with != [cxx_eq_t
o_ne]
            c == '=' || c == '>' || c == '<' || c == '&' || c == '|');
                                                              ^
[info] All mutations have been killed
[info] Mutation score: 100%
[info] Total execution time: 88900ms
All tests passed successfully.
```

Figure 5: Results of test_tokenizer 2

5.4 Test Parser

1. Make mutation testcases in test_parser.cpp file.

test_parser.cpp

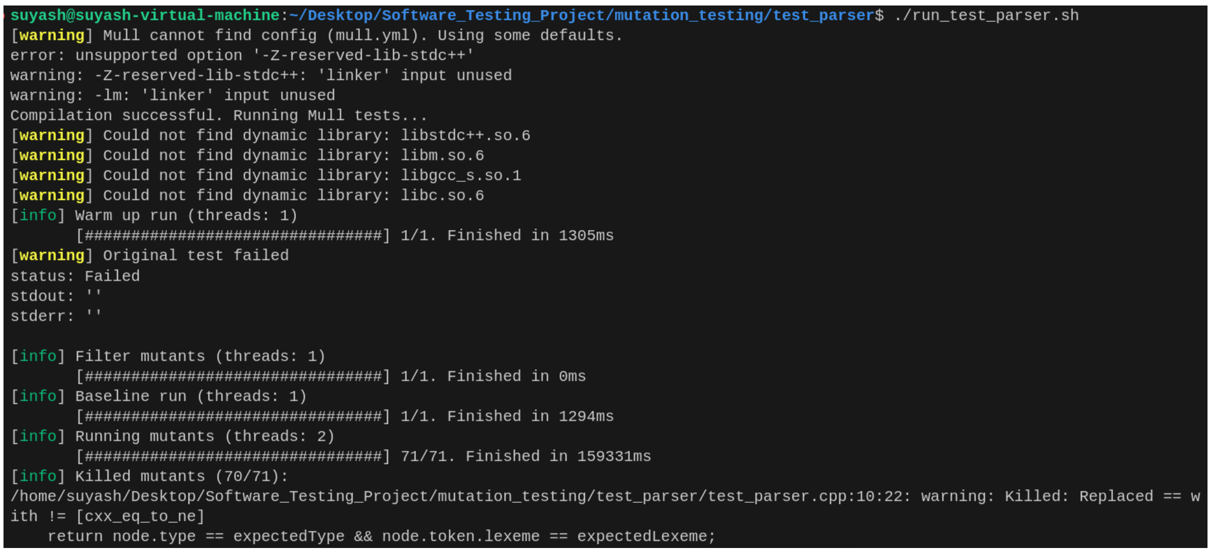
```cpp
#include "../../src/parser/parser.cpp"
#include "../../src/tokenizer/tokenizer.h"
#include <iostream>
#include <vector>
#include <string>
using namespace std;

int main() {
    // Test Case 1: Parse variable declaration
    {
        string input = "Declare␣x␣As␣Integer";
        Node ast = parseInput(input);

        if (ast.children.size() != 1 ||
            !checkNode(ast.children[0], NodeType::DECLARATION,
            "Declare") ||
            ast.children[0].children.size() != 2 ||
            !checkNode(ast.children[0].children[0],
            NodeType::IDENTIFIER, "x") ||
            !checkNode(ast.children[0].children[1],
            NodeType::IDENTIFIER, "Integer")) {
            return 1; // Test failed
        }
    }
}

//Similarly, include remaining testcases
```

2. Run above mutation testcases using following bash script.

run_test_parser.sh

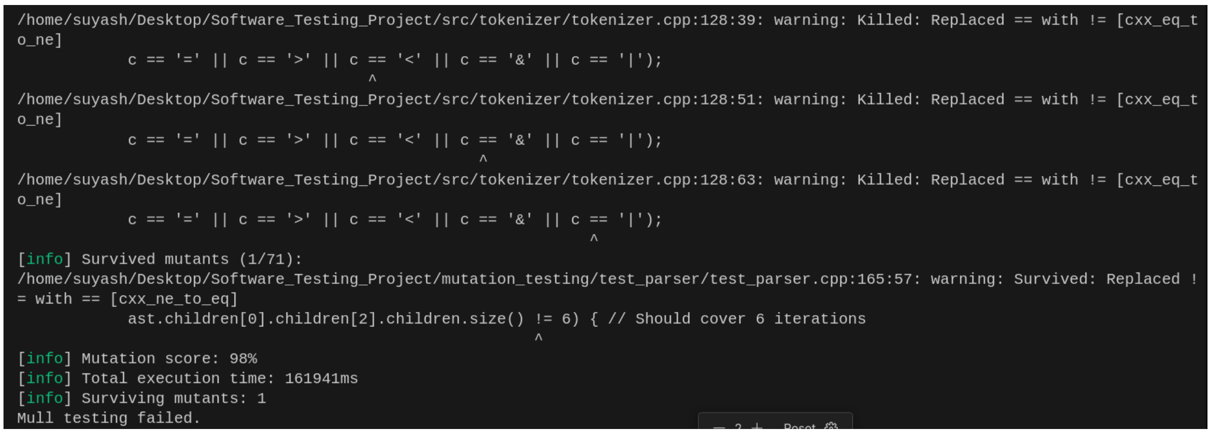
```bash
#!/bin/bash

# Define paths for source files and the output executable
PARSER_SRC="../../src/parser/parser.cpp"
TEST_PARSER_SRC="test_parser.cpp"
OUTPUT_EXEC="parser_test"

# Step 1: Compile the source and test files with clang-12 and Mull
clang-12 -fexperimental-new-pass-manager \
         -fpass-plugin=/usr/lib/mull-ir-frontend-12 \
         -stdlib=libstdc++ \
         -g -grecord-command-line \
         $TEST_PARSER_SRC -o $OUTPUT_EXEC -lstdc++ -lm

# Check if compilation was successful
if [[ $? -ne 0 ]]; then
    echo "Compilation␣failed.␣Exiting..."
    exit 1
fi

echo "Compilation␣successful.␣Running␣Mull␣tests..."

# Step 2: Run the compiled executable with Mull runner
mull-runner-12 -ide-reporter-show-killed $OUTPUT_EXEC
exit 0
```

3. Output and results are as follows:



```
suyash@suyash-virtual-machine:~/Desktop/Software_Testing_Project/mutation_testing/test_parser$ ./run_test_parser.sh
[warning] Mull cannot find config (mull.yml). Using some defaults.
error: unsupported option '-Z-reserved-lib-stdc++'
warning: -Z-reserved-lib-stdc++: 'linker' input unused
warning: -lm: 'linker' input unused
Compilation successful. Running Mull tests...
[warning] Could not find dynamic library: libstdc++.so.6
[warning] Could not find dynamic library: libm.so.6
[warning] Could not find dynamic library: libgcc_s.so.1
[warning] Could not find dynamic library: libc.so.6
[info] Warm up run (threads: 1)
       [################################] 1/1. Finished in 1305ms
[warning] Original test failed
status: Failed
stdout: ''
stderr: ''

[info] Filter mutants (threads: 1)
       [################################] 1/1. Finished in 0ms
[info] Baseline run (threads: 1)
       [################################] 1/1. Finished in 1294ms
[info] Running mutants (threads: 2)
       [################################] 71/71. Finished in 159331ms
[info] Killed mutants (70/71):
/home/suyash/Desktop/Software_Testing_Project/mutation_testing/test_parser/test_parser.cpp:10:22: warning: Killed: Replaced == w
ith != [cxx_eq_to_ne]
    return node.type == expectedType && node.token.lexeme == expectedLexeme;
```

Figure 6: Results of test_parser 1



```
/home/suyash/Desktop/Software_Testing_Project/src/tokenizer/tokenizer.cpp:128:39: warning: Killed: Replaced == with != [cxx_eq_t
o_ne]
            c == '=' || c == '>' || c == '<' || c == '&' || c == '|');
                                       ^
/home/suyash/Desktop/Software_Testing_Project/src/tokenizer/tokenizer.cpp:128:51: warning: Killed: Replaced == with != [cxx_eq_t
o_ne]
            c == '=' || c == '>' || c == '<' || c == '&' || c == '|');
                                                   ^
/home/suyash/Desktop/Software_Testing_Project/src/tokenizer/tokenizer.cpp:128:63: warning: Killed: Replaced == with != [cxx_eq_t
o_ne]
            c == '=' || c == '>' || c == '<' || c == '&' || c == '|');
                                                               ^
[info] Survived mutants (1/71):
/home/suyash/Desktop/Software_Testing_Project/mutation_testing/test_parser/test_parser.cpp:165:57: warning: Survived: Replaced !
= with == [cxx_ne_to_eq]
            ast.children[0].children[2].children.size() != 6) { // Should cover 6 iterations
                                                         ^
[info] Mutation score: 98%
[info] Total execution time: 161941ms
[info] Surviving mutants: 1
Mull testing failed.
```

Figure 7: Results of test_parser 2

5.5 Test CodeGenerator

1. Make mutation testcases in test_codeGenerator.cpp file.

test_codeGenerator.cpp

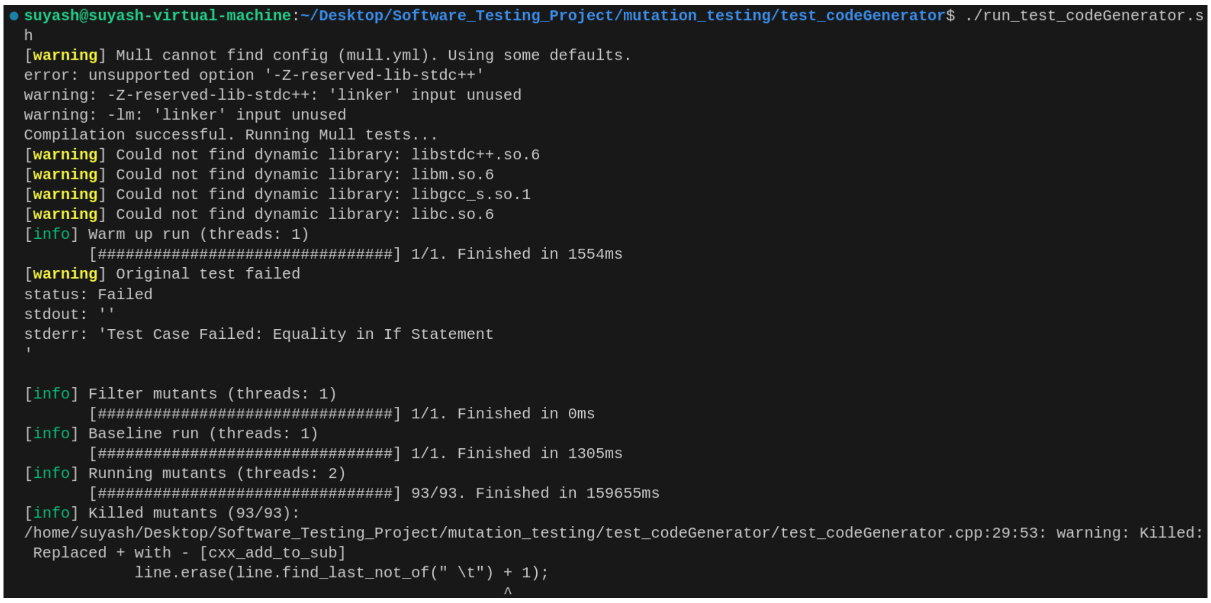
```
#include "../../src/codeGenerator/codeGenerator.cpp"
#include "../../src/parser/parser.h"
#include "../../src/tokenizer/tokenizer.h"
using namespace std;

int main() {
    // Test Case 1: Generate code for variable declaration
    {
        string input = "Declare␣x␣As␣Integer";
        Tokenizer tokenizer(input);
        vector<Token> tokens = tokenizer.tokenize();
        Parser parser(tokens);
        Node ast = parser.parse();

        CodeGenerator generator;
        string generatedCode = generator.generateCode(ast);

        string expectedCode = R"(
␣␣␣␣␣␣␣␣␣␣␣␣#include␣<bits/stdc++.h>
␣␣␣␣␣␣␣␣␣␣␣␣using␣namespace␣std;
␣␣␣␣␣␣␣␣␣␣␣␣int␣main()␣{
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣int␣x;
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣return␣0;
␣␣␣␣␣␣␣␣␣␣␣␣}
␣␣␣␣␣␣␣␣)";

        if (!checkGeneratedCode(generatedCode, expectedCode)) {
            cerr << "Test␣Case␣1␣Failed:␣Variable␣Declaration" ;
            return 1;
        }
    }
}

//Similarly, include remaining testcases
```

2. Run above mutation testcases using following bash script.

run_test_codeGenerator.sh

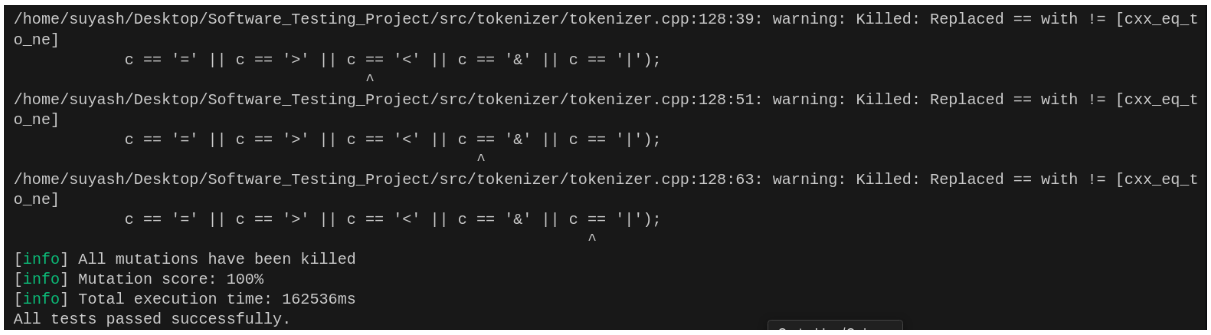
```bash
#!/bin/bash

# Define paths for source files and the output executable
CODEGENERATOR_SRC="../../src/codeGenerator/codeGenerator.cpp"
TEST_CODEGENERATOR_SRC="test_codeGenerator.cpp"
OUTPUT_EXEC="codeGenerator_test"

# Step 1: Compile the source and test files with clang-12 and Mull
clang-12 -fexperimental-new-pass-manager \
         -fpass-plugin=/usr/lib/mull-ir-frontend-12 \
         -stdlib=libstdc++ \
         -g -grecord-command-line \
         $TEST_CODEGENERATOR_SRC -o $OUTPUT_EXEC -lstdc++ -lm

# Check if compilation was successful
if [[ $? -ne 0 ]]; then
    echo "Compilation␣failed.␣Exiting..."
    exit 1
fi

echo "Compilation␣successful.␣Running␣Mull␣tests..."

# Step 2: Run the compiled executable with Mull runner
mull-runner-12 -ide-reporter-show-killed $OUTPUT_EXEC
exit 0
```

3. Output and results are as follows:



```
● suyash@suyash-virtual-machine:~/Desktop/Software_Testing_Project/mutation_testing/test_codeGenerator$ ./run_test_codeGenerator.s
  h
  [warning] Mull cannot find config (mull.yml). Using some defaults.
  error: unsupported option '-Z-reserved-lib-stdc++'
  warning: -Z-reserved-lib-stdc++: 'linker' input unused
  warning: -lm: 'linker' input unused
  Compilation successful. Running Mull tests...
  [warning] Could not find dynamic library: libstdc++.so.6
  [warning] Could not find dynamic library: libm.so.6
  [warning] Could not find dynamic library: libgcc_s.so.1
  [warning] Could not find dynamic library: libc.so.6
  [info] Warm up run (threads: 1)
         [#################################] 1/1. Finished in 1554ms
  [warning] Original test failed
  status: Failed
  stdout: ''
  stderr: 'Test Case Failed: Equality in If Statement
  '

  [info] Filter mutants (threads: 1)
         [#################################] 1/1. Finished in 0ms
  [info] Baseline run (threads: 1)
         [#################################] 1/1. Finished in 1305ms
  [info] Running mutants (threads: 2)
         [#################################] 93/93. Finished in 159655ms
  [info] Killed mutants (93/93):
  /home/suyash/Desktop/Software_Testing_Project/mutation_testing/test_codeGenerator/test_codeGenerator.cpp:29:53: warning: Killed:
   Replaced + with - [cxx_add_to_sub]
              line.erase(line.find_last_not_of(" \t") + 1);
                                                        ^
```

Figure 8: Results of test_codeGenerator

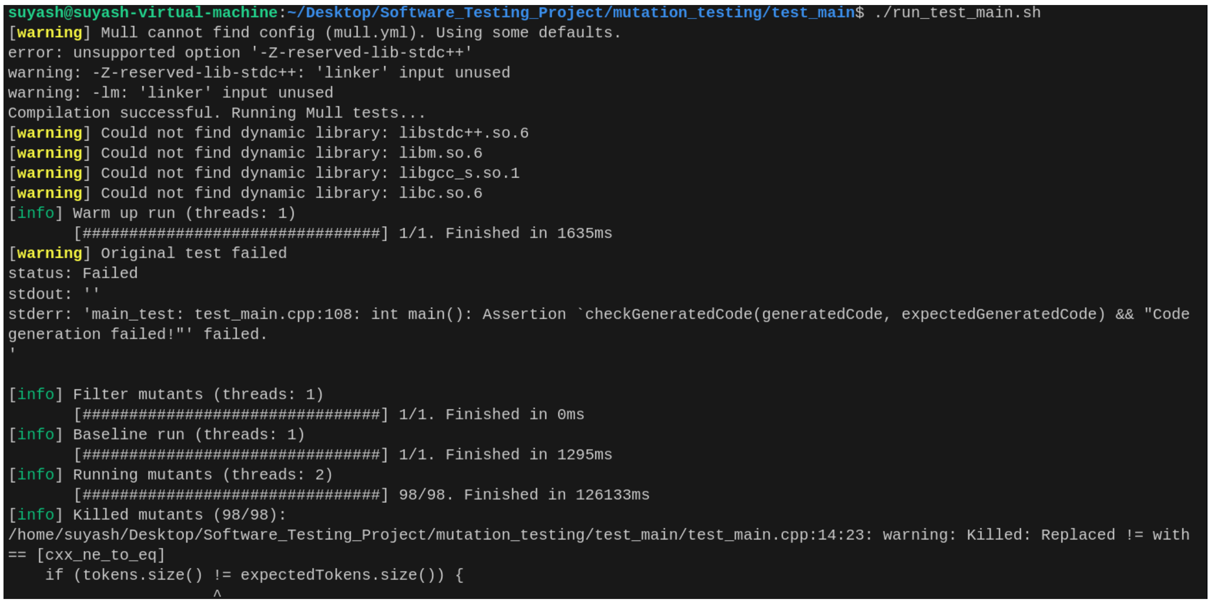


```
/home/suyash/Desktop/Software_Testing_Project/src/tokenizer/tokenizer.cpp:128:39: warning: Killed: Replaced == with != [cxx_eq_t
o_ne]
            c == '=' || c == '>' || c == '<' || c == '&' || c == '|');
                                          ^
/home/suyash/Desktop/Software_Testing_Project/src/tokenizer/tokenizer.cpp:128:51: warning: Killed: Replaced == with != [cxx_eq_t
o_ne]
            c == '=' || c == '>' || c == '<' || c == '&' || c == '|');
                                                      ^
/home/suyash/Desktop/Software_Testing_Project/src/tokenizer/tokenizer.cpp:128:63: warning: Killed: Replaced == with != [cxx_eq_t
o_ne]
            c == '=' || c == '>' || c == '<' || c == '&' || c == '|');
                                                                  ^
[info] All mutations have been killed
[info] Mutation score: 100%
[info] Total execution time: 162536ms
All tests passed successfully.
```

Figure 9: Results of test_codeGenerator

5.6 Test Main

1. Make mutation testcases in test_main.cpp file.
2. Run above mutation testcases using following bash script.
3. Output and results are as follows:



```
suyash@suyash-virtual-machine:~/Desktop/Software_Testing_Project/mutation_testing/test_main$ ./run_test_main.sh
[warning] Mull cannot find config (mull.yml). Using some defaults.
error: unsupported option '-Z-reserved-lib-stdc++'
warning: -Z-reserved-lib-stdc++: 'linker' input unused
warning: -lm: 'linker' input unused
Compilation successful. Running Mull tests...
[warning] Could not find dynamic library: libstdc++.so.6
[warning] Could not find dynamic library: libm.so.6
[warning] Could not find dynamic library: libgcc_s.so.1
[warning] Could not find dynamic library: libc.so.6
[info] Warm up run (threads: 1)
       [################################] 1/1. Finished in 1635ms
[warning] Original test failed
status: Failed
stdout: ''
stderr: 'main_test: test_main.cpp:108: int main(): Assertion `checkGeneratedCode(generatedCode, expectedGeneratedCode) && "Code
generation failed!"' failed.
'

[info] Filter mutants (threads: 1)
       [################################] 1/1. Finished in 0ms
[info] Baseline run (threads: 1)
       [################################] 1/1. Finished in 1295ms
[info] Running mutants (threads: 2)
       [################################] 98/98. Finished in 126133ms
[info] Killed mutants (98/98):
/home/suyash/Desktop/Software_Testing_Project/mutation_testing/test_main/test_main.cpp:14:23: warning: Killed: Replaced != with
== [cxx_ne_to_eq]
    if (tokens.size() != expectedTokens.size()) {
                      ^
```

Figure 10: Results of test_main

6 Conclusion

This project successfully implemented a pseudocode-to-C++ converter with robust unit and mutation testing. The use of tools like Mull ensures comprehensive validation and enhances the quality of the test suite. Future work could include expanding mutation operators and automating the setup for large-scale projects.