NobleProg

Lab Exercise 7 – C++ Build using Bazel

(Unit Testing using gtest)

Objective: Create a lab exercise to teach C++ unit testing using Bazel.

Prerequisites:

- Bazel installed on your machine.
- Basic knowledge of Bazel and C++.
- A testing framework such as Google Test (gtest) installed (you can use other testing frameworks if preferred).

Exercise:

Scenario: You have a simple C++ project, and you want to write and run unit tests for it using Bazel and a testing framework.

Step 1: Project Setup:

Create a directory structure for your lab exercise:

lab_cpp_testing/
|-- WORKSPACE
|-- BUILD
|-- main_test.cpp

Step 2: Write the following code in WORKSPACE file.

```
load("@bazel_tools//tools/build_defs/repo:http.bzl", "http_archive")

http_archive(
  name = "com_google_googletest",
  urls = ["https://github.com/google/googletest/archive/5ab508a01f9eb089207ee87fd547d290da39d015.zip"],
  strip_prefix = "googletest-5ab508a01f9eb089207ee87fd547d290da39d015",
)
```

Step 3: Write a Test File:

Create a C++ test source file, main_test.cpp, that tests the code in main.cpp:

```
#include "gtest/gtest.h"

int add(int a, int b) {
   return a + b;
}

TEST(AddFunctionTest1, BasicTest1) {
   EXPECT_EQ(add(2, 3), 5);
}
TEST(AddFunctionTest2, BasicTest2) {
   EXPECT_EQ(add(-1, 1), 0);
}
```

In this example, we're using Google Test (gtest) to write a simple test case called AddFunctionTest. You can replace the EXPECT_EQ lines with your own test assertions.

Step 4: Create BUILD Files:

In the BUILD file (located in the project root), define rules for your main program and the test:

```
cc_test(
   name = "main_test",
   srcs = ["main_test.cpp"],
   deps = ["@com_google_googletest//:gtest_main",],
)
```

In this example:
main is a binary target for your main program (main.cpp).
main_test is a test target for your test file (main_test.cpp). It depends on the main binary and the Google Test framework (@com_google_googletest//:gtest_main).

Step 5: Build and Run Tests:

Test the project using Bazel:

```
bazel test //...
```

Bazel will build and execute your tests. If all tests pass, you'll see a summary of the test results.

Conclusion:

This lab exercise demonstrates how to write and run C++ unit tests using Google Test and Bazel. It emphasizes the importance of writing tests to verify the correctness of C++ code and shows how Bazel can be used to automate the testing process. This exercise provides hands-on experience in C++ testing with Bazel and a testing framework.