Unit Tests

SWPP

April 23th

Juneyoung Lee

Assignment 5. Code Review + GoogleTest

- We'll announce the assignment today soon

1. You will write simple unit tests using GoogleTest framework
   - We'll use the GoogleTest framework that is shipped with LLVM 10.0
   - It will be more lightweight than assignment 4.
   - Please check whether today's scripts work well on your computer...!

Assignment 5. Code Review + GoogleTest

2. We'll give code to each other & have a code review (2~3 per a student)
   - Leave it as C++ comments on the code.
   - The diff should be less than 50 lines & fit in 80 columns (search "80 column rules")
   - Should be written in English.
   - We'll see whether the reviewer...
     - Catched 'trivial' bugs? (you can run it with test cases)
     - Suggested a good library for simplifying the implementation?
     - Etc.. (see the previous practice session slides)

GoogleTest

- A framework that allows you to write unit tests that..
  - Shows the test results in a fancy way
  - Measures elapsed times
  - Has command-line arguments for showing the list of tests, repeat N times, etc

GoogleTest - simple.sh

```
$ ./simple.sh build ~/class/llvm-10.0-releaseassert/bin/
----- build -----

$ ./simpleTest
[==========] Running 2 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 2 tests from MyTest
[ RUN      ] MyTest.OnePlusOneIsTwo
[       OK ] MyTest.OnePlusOneIsTwo (0 ms)
[ RUN      ] MyTest.OnePlusOneIsNotThree
[       OK ] MyTest.OnePlusOneIsNotThree (0 ms)
[----------] 2 tests from MyTest (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test case ran. (0 ms total)
[  PASSED  ] 2 tests.
```

- There is one test-case: MyTest
- There are two tests: OnePlusOneIsTwo, OnePlusOneIsNotThree

Let's see the simpleTest.cpp...!

EXPECT functions

`EXPECT_TRUE(condition);`	`condition` is true
`EXPECT_FALSE(condition);`	`condition` is false

`EXPECT_EQ(val1, val2);`	`val1 == val2`
`EXPECT_NE(val1, val2);`	`val1 != val2`
`EXPECT_LT(val1, val2);`	`val1 < val2`
`EXPECT_LE(val1, val2);`	`val1 <= val2`
`EXPECT_GT(val1, val2);`	`val1 > val2`
`EXPECT_GE(val1, val2);`	`val1 >= val2`

Writing a custom message:

```
for (int i = 0; i < x.size(); ++i) {
  EXPECT_EQ(x[i], y[i]) << "Vectors x and y differ at index " << i;
}
```

ASSERT vs. EXPECT

- ASSERT_*: If fails, halt the whole test cases immediately
- EXPECT_*: If fails, only the test fails.

Fatal assertion	Nonfatal assertion	Verifies
`ASSERT_TRUE(condition);`	`EXPECT_TRUE(condition);`	`condition` is true
`ASSERT_FALSE(condition);`	`EXPECT_FALSE(condition);`	`condition` is false

- For details: see https://github.com/google/googletest/blob/master/googletest/docs/primer.md

How to write LLVM unit tests?

- You need an IR program to test..!
- There are three ways to write the IR program to test:
    1. Write it as a standalone file & read it
    2. Write the program as a C++ string & parse it
    3. Use an IRBuilder to build a function to test
- We'll visit 2 and 3 as they allow us to dynamically generate a program.

Let's test.. alias analysis!

- Are two memory locations overlap?
- A memory location is a pair of (pointer, memory access size).
- 4 results:
  - NoAlias: two pointers (p, q) don't overlap with the given size
  - MustAlias: two pointers (p, q) are equivalent (p == q)
  - PartialAlias: two pointers (p, q) partially overlap with tie given size
  - MayAlias: we don't know!

* note that you don't need to know about alias analysis to do your homework.
  But it will be helpful for your project.

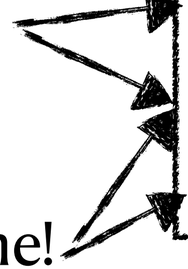
Alias analysis example

These two accesses don't overlap!

(will say NoAlias)

These two accesses are the same!

(will say MustAlias)



```
int *p = malloc(4);
int *q = malloc(4);
// Assume p != null /\ q != null
*p = 1;
*q = 2;
*(q+0) = 3;
```

Let’s parse an IR program first!

- aliasAnalysisTest.cpp

```
TEST_F(MyAliasAnalysisTest, TestUsingParseAssembly) {
  // Let's parse this IR assembly, and create mappings from register names
  // to instruction objects!
  parseAssembly(R"myasm(
    define void @test(i1 %c) {
      %p = alloca i32
      %q = alloca i32
      %p2 = getelementptr i32, i32* %p, i32 1
      %p3 = getelementptr i32, i32* %p, i32 0
      %r = select i1 %c, i32* %p, i32* %q
     ret void
    }
  )myasm”);
  …
}
```

Let's parse an IR program first!

- aliasAnalysisTest.cpp

```
unique_ptr<Module> parseModule(StringRef Assembly) {
  SMDiagnostic Error;
  unique_ptr<Module> M = parseAssemblyString(Assembly, Error, Context);

  string errMsg;
  raw_string_ostream os(errMsg);
  Error.print("", os);
  EXPECT_TRUE(M) << os.str();

  return M;
}
```

Let's parse an IR program first!

Note that this function is in a class MyAliasAnalysisTest.

```
// Parse the IR assembly by calling parseModule function.
void parseAssembly(StringRef Assembly) {
  this->M = parseModule(Assembly);
  ASSERT_TRUE(M);

  Function *F = M->getFunction("test");
  ASSERT_TRUE(F) << "Test must have a function @test";
  if (!F)
    return;

  for (inst_iterator I = inst_begin(F), E = inst_end(F); I != E; ++I) {
    if (I->hasName())
      this->Instrs[I->getName().str()] = &*I;
  }
  this->testF = F;
}
```

Let’s parse an IR program first!

```
TEST_F(MyAliasAnalysisTest, TestUsingParseAssembly) {
  // Let's parse this IR assembly, and create mappings from register names
  // to instruction objects!
  parseAssembly(R”…”);

 // Mappings created!
  Instruction *p = this->Instrs["p"];
  Instruction *p2 = this->Instrs["p2"];
  outs() << "----- MyAliasAnalysisTest.TestUsingParseAssembly -----" << "\n";
  outs() << "p: " << *p << "\n";
  outs() << "p2: " << *p2 << "\n";
}
```

Creating IR function using IRBuilder

```
// Create an empty function & store it into testF.
void createEmptyTestFunction() {
  this->M = unique_ptr<Module>(new Module("MyModule", Context));

  // No argument, returns void
  FunctionType *FTy =
    FunctionType::get(Type::getVoidTy(Context), {}, false);

  this->testF = Function::Create(FTy, Function::ExternalLinkage, "test", *M);
}
```

Creating IR function using IRBuilder

```
// TestUsingIRBuilder creates a test by constructing an input program using
// IRBuilder & running unit tests.
TEST_F(MyAliasAnalysisTest, TestUsingIRBuilder) {
  // Create an empty IR function & store it into this->testF.
  createEmptyTestFunction();

  // Create an entry basic blcok.
  BasicBlock *Entry = BasicBlock::Create(Context, "entry", testF);
  IRBuilder<> EntryBuilder(Entry);
  auto *p = EntryBuilder.CreateAlloca(Type::getInt32PtrTy(Context), nullptr, "p");
  auto *q = EntryBuilder.CreateAlloca(Type::getInt32PtrTy(Context), nullptr, "q");
  auto *p2 = EntryBuilder.CreateGEP(p, ConstantInt::get(Type::getInt32Ty(Context), 1));
  auto *p3 = EntryBuilder.CreateGEP(p, ConstantInt::get(Type::getInt32Ty(Context), 0));
  …
}
```