C++ and Build Systems (CMake, Bazel)

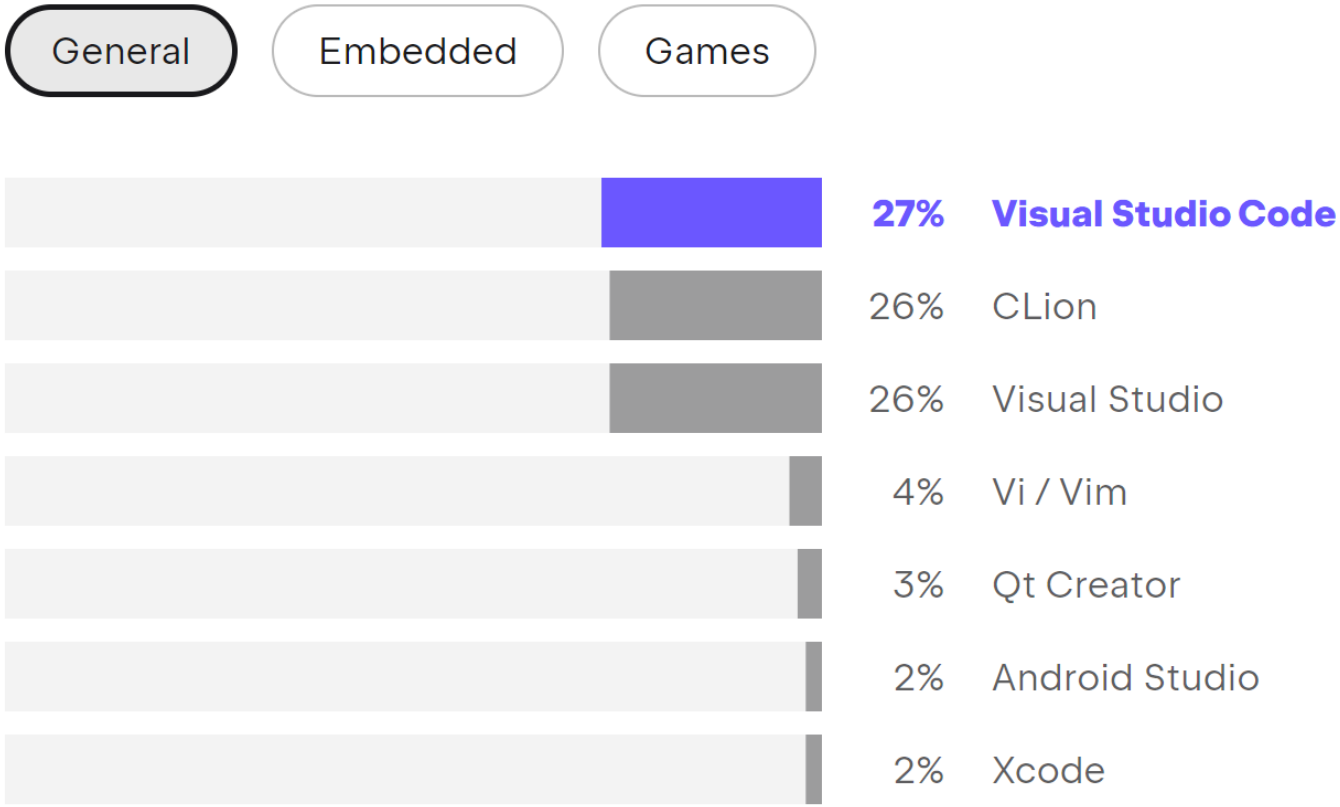
ITP 435
Week 9, Lecture 1

Why do we need a build system?

- C++ build environments are ***heterogeneous***

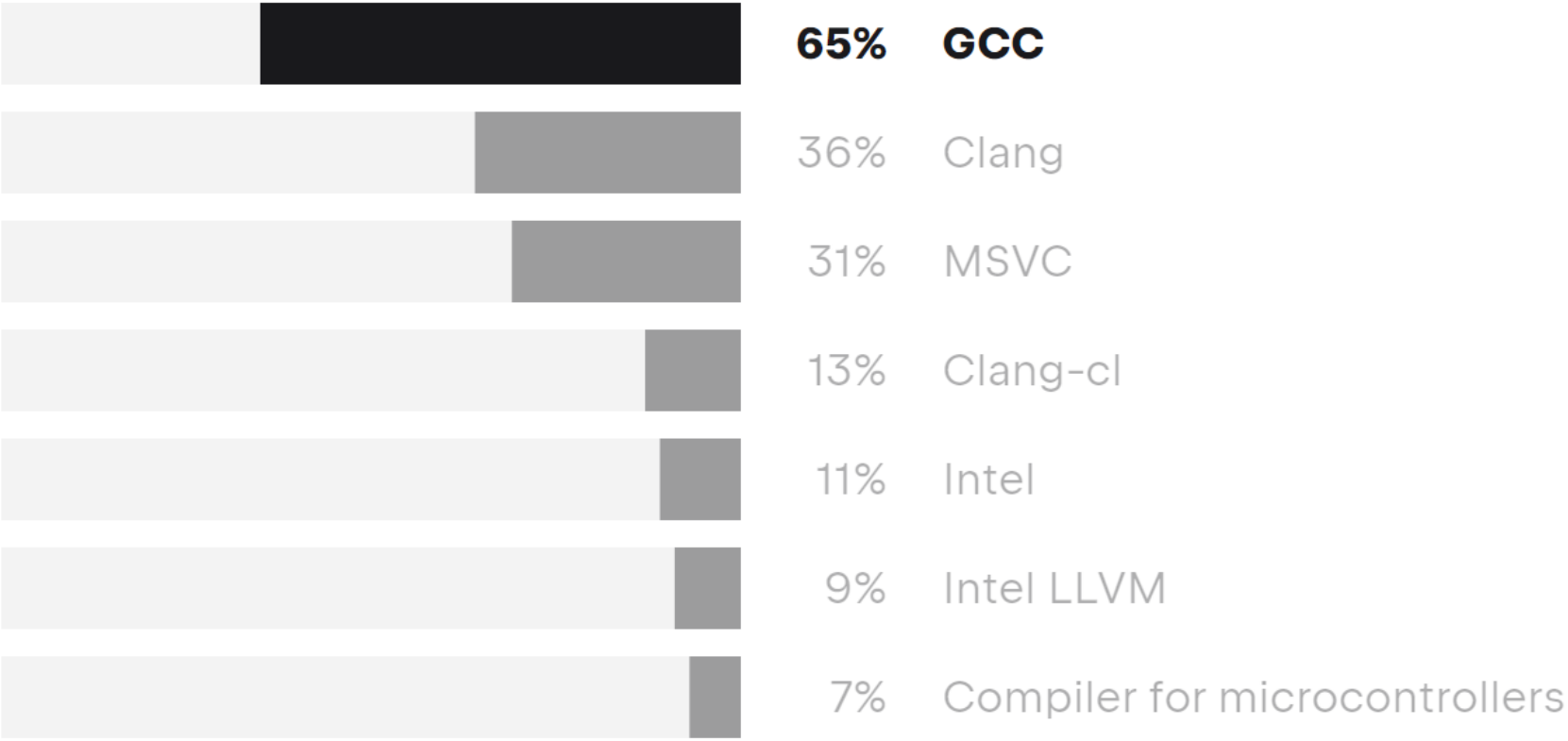
- Unlike other languages there isn't one guaranteed development stack!

Which IDE/editor do you use the most?



"The State of Developer Ecosystem Survey in 2023 (C++)":
https://www.jetbrains.com/lp/devecosystem-2023/cpp/

Which compilers do you regularly use?



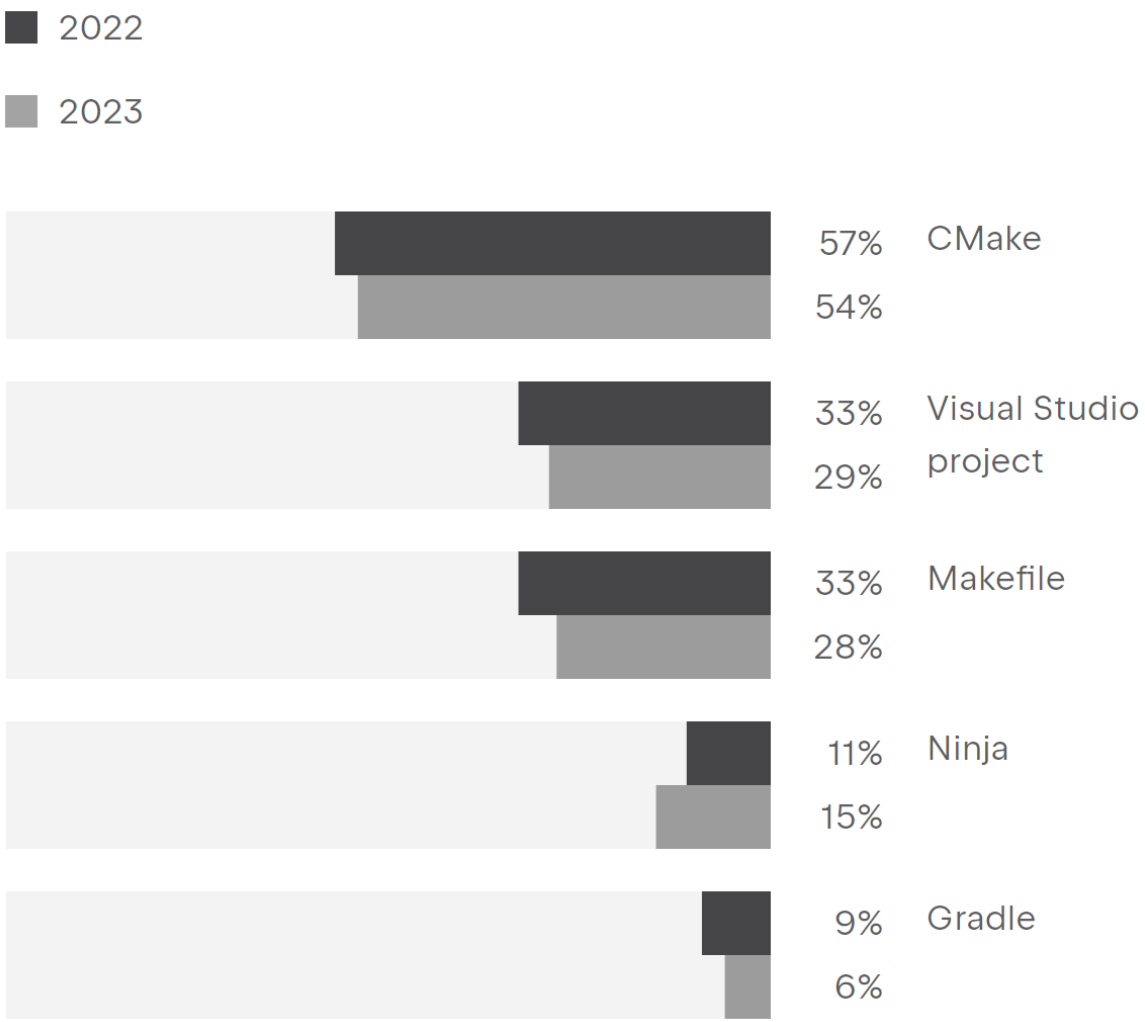
"The State of Developer Ecosystem Survey in 2023 (C++)":
https://www.jetbrains.com/lp/devecosystem-2023/cpp/

So what should we use?

- There are ***a lot*** of choices

- Here are some of them:
  - CMake (https://cmake.org/)
  - Bazel (https://bazel.build/)
  - Meson (https://mesonbuild.com/)
  - build2 (https://build2.org/)

Which project models/build systems do you regular use, if any?



"The State of Developer Ecosystem Survey in 2023 (C++)":
https://www.jetbrains.com/lp/devecosystem-2023/cpp/

CMake Crash Course

CMake Samples

- Most of the concepts we'll go over are also demonstrated on samples on GitHub!

- https://github.com/chalonverse/CMakeSamples

CMake “Hello World” (Ex01 on GitHub)

- Suppose we just have a single Main.cpp:

```
#include <iostream>

int main()
{
   std::cout << "Hello, world!\n";
   return 0;
}
```

- How do we build this with CMake?

CMake “Hello World”

- In the same directory as Main.cpp, create a CMakeLists.txt:

```
# Comments begin with #

# The minimum required version of CMake
# (in this case we're asking for 3.16 or higher)
cmake_minimum_required(VERSION 3.16)

# Name of this top-level project.
# This doesn't have to correlate to any specific executable name.
project(CMakeSamples)

# Create an executable called Ex01 that compiles Main.cpp
add_executable(Ex01 Main.cpp)
```

Building CMake "Hello World"

Procedure to build from the command line (in a UNIX shell):

1. Tell cmake to generate data for platform-specific build system in the build directory

   ```
   $ cmake -B build
   ```

2. Change directory to the build director

   ```
   $ cd build
   ```

3. Tell cmake to execute platform-specific build commands

   ```
   $ cmake --build .
   ```

CMake Generators

- When executing the initial cmake command, you can specify a generator with `-G`

- For example, to generate an Xcode project, run:

```
$ cmake -G Xcode -B build
```

- There are many different generators: https://cmake.org/cmake/help/latest/manual/cmake-generators.7.html

cmake --build vs. Other Build Commands

- Different generators might create different project file formats (like Xcode, Visual Studio, etc)

- While you can certainly use those custom IDEs/tools (like we do with Xcode), `cmake --build` will always try to compile from the command line, regardless of the generator

Requiring a C++ Version (Ex02 on GitHub)

- Often, we need a specific newer version of C++

- To require a specific version, add the following to your CMakeLists, before adding any executable/library targets:

```
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
```

Multiple Files, Basic Way (Ex03 on GitHub)

- Often your executable will have more than one cpp file
- Suppose we have the following files:
  - Hello.h
  - Hello.cpp
  - Main.cpp

- You can specify multiple source files to add_executable:

add_executable(Ex03 Main.cpp Hello.cpp)

- Note that we do ***NOT*** add the .h files to this (there's no need to since the header files aren't separately compiled)

Multiple Files with GLOB (Ex04 on GitHub)

- Manually specifying new files every time you add them is tedious
- You can instead tell CMake to grab all files with a specific extension and build all of them:

```
# This grabs all files in this directory that end with .cpp
# and saves it in a variable called ${source_files}
file(GLOB source_files CONFIGURE_DEPENDS "*.cpp")
# This says to create an executable target called Ex04 that
# compiles the ${source_files}
add_executable(Ex04 ${source_files})
```

- `CONFIGURE_DEPENDS` tells CMake to verify the file list hasn't changed every time you build. Without it, the GLOB would only generate the file list each time you run cmake

GLOB gotcha

- The GLOB rule is a newer thing, and the documentation warns:

We do not recommend using GLOB to collect a list of source files from your source tree. If no CMakeLists.txt file changes when a source is added or removed then the generated build system cannot know when to ask CMake to regenerate. The CONFIGURE_DEPENDS flag may not work reliably on all generators, or if a new generator is added in the future that cannot support it, projects using it will be stuck. Even if CONFIGURE_DEPENDS works reliably, there is still a cost to perform the check on every rebuild.

Building/Linking Against Libraries (Ex05 on GitHub)

- What if you want to make a library in one directory and have the executable include headers from and link against that library

- In this example, we have the following file structure:
  - Ex05Exe
    - CMakeLists.txt
    - Main.cpp
  - Ex05Lib
    - CMakeLists.txt
    - Hello.cpp
    - Hello.h

Building the Library

- First, in Ex05Lib's CMakeLists.txt, use add_library instead of add_executable:

```
# This says to create a library called Ex05Lib that compiles Hello.cpp
add_library(Ex05Lib Hello.cpp)
```

Including Headers and Linking

- The Ex05Exe CMakeLists needs this:

```
# This says we need to be able to include headers from Ex05Lib
include_directories(../Ex05Lib)

# This says to create an executable target called Ex05Exe that
# compiles Main.cpp
add_executable(Ex05Exe Main.cpp)

# This says to link Ex05Exe with the Ex05Lib library
target_link_libraries(Ex05Exe Ex05Lib)
```

Linking External Libraries (Ex06 on GitHub)

- Sometimes you need to link against a common external library (like zlib)

- CMake is able to find many such common libraries, if they are previously installed.

- Some CMake-compatible ways to install common libraries:
  - Windows – vcpkg (though you have to specify an extra CMAKE_TOOLCHAIN_FILE define for this to work)
  - Mac – If not installed by default, use homebrew
  - Linux – If not installed by default, you can `apt-get`

Linking External Libraries

- We need to tell CMake to find Zlib and then link against it:

```
# This says to find zlib and error out if not found
find_package(ZLIB REQUIRED)

# This is a normal executable target
add_executable(Ex06 Main.cpp)

# This says we need to link against ZLIB
target_link_libraries(Ex06 PRIVATE ZLIB::ZLIB)
```

Fetching External Projects (Ex07)

- CMake does support git submodules (if that's your preference), though it's a bit complex to setup

- In newer CMake, you can use "fetch" commands

- For example, say we want to use the Catch testing library in our project, but don't want to manually include the headers and instead have CMake pull it for us at generator time

Fetching External Projects

```
# This is required to issue any FetchContent commands
include(FetchContent)

# This declares "catch" as a git repository we depend on
FetchContent_Declare(
  catch
  GIT_REPOSITORY https://github.com/catchorg/Catch2.git
  GIT_TAG v2.13.3
)
```

Fetching External Projects

```
# This says we want the "catch" we declared available
FetchContent_MakeAvailable(catch)

# The ${catch_SOURCE_DIR} variable is set by MakeAvailable
# In this case we want catch.hpp to be in the include path
include_directories(${catch_SOURCE_DIR}/single_include/catch2)

# This is a normal executable command
add_executable(Ex07 Main.cpp)
```

Platform-specific Settings (Ex08 on GitHub)

- Although you want everything to just work cross-platform, sometimes you have to conditionally set compiler flags (or other things) on different platforms

- You can use if/elseif statements with platform-defined variables such as `APPLE` and `WIN32`

Platform-specific Settings

```
# Are we on Windows?
if (WIN32)
   # Use existing compiler flags plus /WX
   set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} /WX")
# Are we on Mac (or IOS etc)?
elseif (APPLE)
   # Use existing compiler flags plus -Werror
   set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Werror")
# If neither of these, assume another Unix platform
else()
   # Use existing compiler flags plus -Werror
   set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Werror")
endif()
```

Other Steps/Dependencies

- You can add other tools/things to run using add_custom_command

- We won't cover it, but it allows you to specify dependencies of things that need to execute before or after the main build step!

Parallelization

- If you have a large codebase with a lot of cpp files, you may want to compile all the cpp files in parallel

- You can request this at build time with `--parallel` followed by the request number of concurrent jobs (in this case, 16):

```
$ cmake --build . --parallel 16
```

- ***Note***: This only works if the native build tool supports parallelization. For example, both Ninja and Xcode do

More CMake Info

- Keep in mind CMake has been around a long time, so you want to stick to “modern” CMake paradigms when possible!

- A good starting point is here:

https://cliutils.gitlab.io/modern-cmake/

In-class activity

Bazel

Sample Repos

- For these slides use the following repos:
  - Bazel Tutorial: https://github.com/bazelbuild/examples
    - You can follow the tutorial here: https://docs.bazel.build/versions/master/tutorial/cpp.html
  - Bazel C++ Starter Repo: https://github.com/ourarash/cpp-template

What is Bazel?

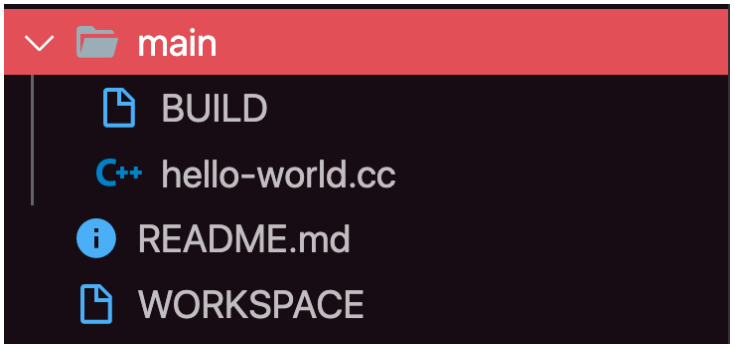
- Bazel is an **open-source** build and **test** tool
- Similar to Make, Maven, and Gradle.
- Human-readable, high-level build language
- Supports projects in **multiple languages** and builds outputs for multiple platforms.
- Supports **large codebases** across **multiple repositories**, and large numbers of users.
- Widely used at Google for building millions of lines of code.

Why Bazel?

- **High-level build language.** Bazel uses an abstract, human-readable language to describe the build properties of your project at a high semantical level.
- **Bazel is fast and reliable.**
  - Caches all previously done work and tracks changes to both file content and build commands.
  - Supports parallelism.
- **Bazel is multi-platform:** Linux, macOS, and Windows.
  - Can build binaries and deployable packages for multiple platforms, including desktop, server, and mobile, from the same project.
- **Bazel scales.** Bazel maintains agility while handling builds with 100k+ source files. It works with multiple repositories and user bases in the tens of thousands.
- **Bazel is extensible.** Many languages are supported, and you can extend Bazel to support any other language or framework.

Bazel WORKSPACE and Package

- A workspace is a **directory** that holds your project's source files and Bazel's build outputs. It contains:
  - The **WORKSPACE** file: which identifies the directory and its contents as a Bazel workspace and lives at the **root** of the project's directory structure.
    - All inputs and dependencies must be in the same workspace. Files residing in different workspaces are independent of one another unless linked (we don't cover linking)
  - One or more **BUILD** files, which tell Bazel how to build different parts of the project.
- **Package**: A directory within the workspace that contains a BUILD file is a *package*.



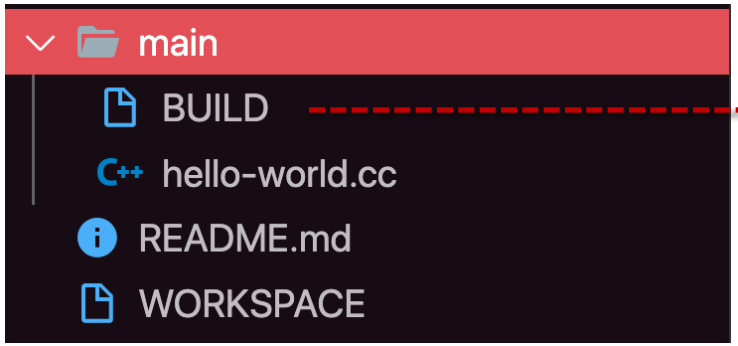
The BUILD file

- A BUILD file contains several different types of instructions for Bazel.
  - ***build rule***: tells Bazel how to build the desired outputs, such as executable binaries or libraries.
  - Each instance of a build rule in the BUILD file is called a *target* and points to a specific set of source files and dependencies. A target can also point to other targets.
  - In our example, the hello-world target instantiates Bazel's built-in cc_binary rule. The rule tells Bazel to build a self-contained executable binary from the hello-world.cc source file with no dependencies.
  - **Attributes**:
    - **name**: name of the target
    - **srcs**: specifies the source file(s) from which Bazel builds the target.

Attributes

```
cc_binary(
    name = "hello-world",
    srcs = ["hello-world.cc"],
)
```

How to build?

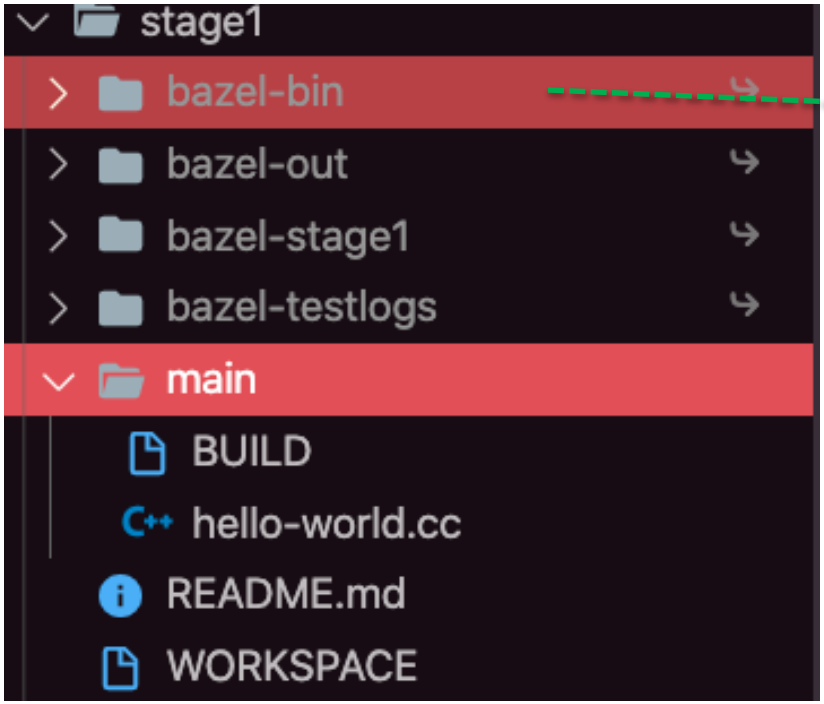



```
cc_binary(
    name = "hello-world",
    srcs = ["hello-world.cc"],
)
```

```
bazel build //main:hello-world
```

```
Starting local Bazel server and connecting to it...
INFO: Analyzed target //main:hello-world (14 packages loaded, 105 targets configured).
INFO: Found 1 target...
Target //main:hello-world up-to-date:
  bazel-bin/main/hello-world
INFO: Elapsed time: 16.762s, Critical Path: 0.99s
INFO: 2 processes: 2 darwin-sandbox.
INFO: Build completed successfully, 5 total actions
```

How to run executable?



Your binary output goes here

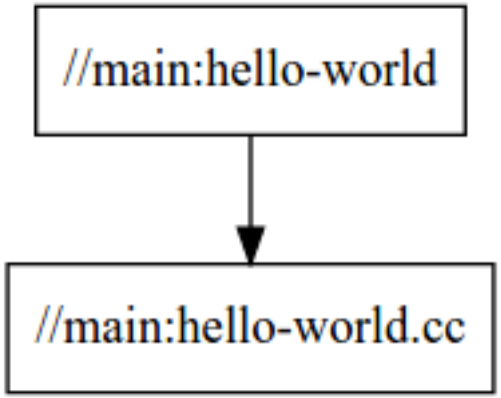
Run your executable:
bazel-bin/main/hello-world

Dependency Grapy

- A great feature of Bazel is generating the dependency graph:

```
> bazel query --notool_deps --noimplicit_deps "deps(//main:hello-world)"   --output graph
```

- This will generate a graph in dot format:




```
digraph mygraph {
  node [shape=box];
  "//main:hello-world"
  "//main:hello-world" -> "//main:hello-world.cc"
  "//main:hello-world.cc"
}
```

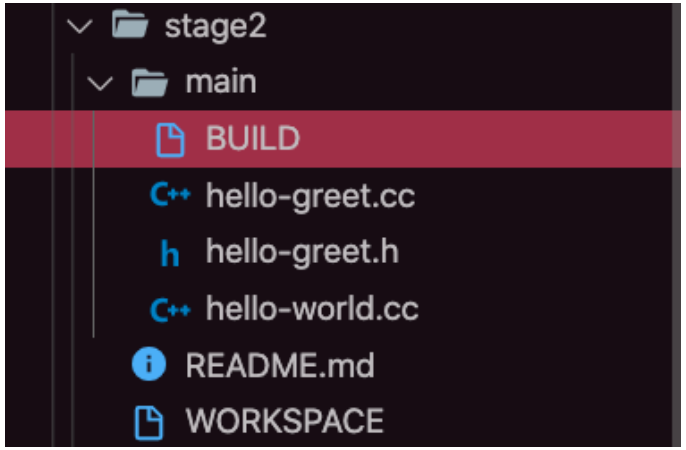
Library Files

- Let's say you want to put some of your library classes and functions in .cc and .c files:

```
cc_library(
  name = "hello-greet",
  srcs = ["hello-greet.cc"],
  hdrs = ["hello-greet.h"],
)

cc_binary(
  name = "hello-world",
  srcs = ["hello-world.cc"],
  deps = [
   ":hello-greet",
  ],
)
```

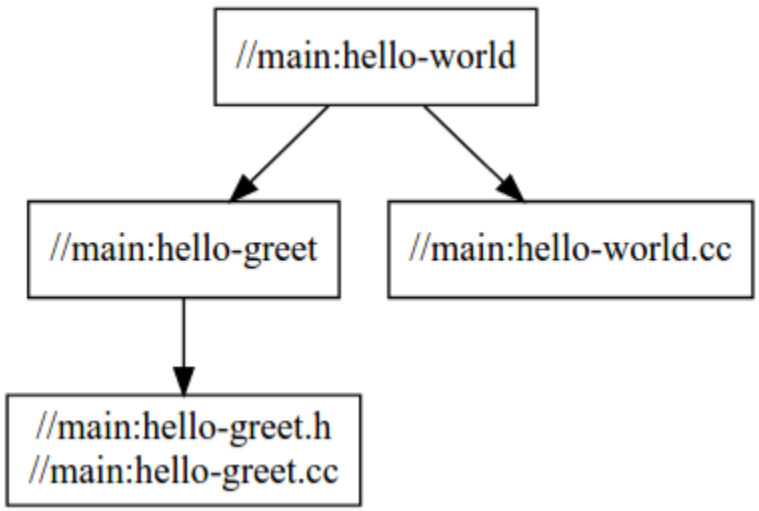
Your library target

Executable that depends on the library



The Dependency Graph

```
cc_library(
  name = "hello-greet",
  srcs = ["hello-greet.cc"],
  hdrs = ["hello-greet.h"],
)

cc_binary(
  name = "hello-world",
  srcs = ["hello-world.cc"],
  deps = [
   ":hello-greet",
  ],
)
```

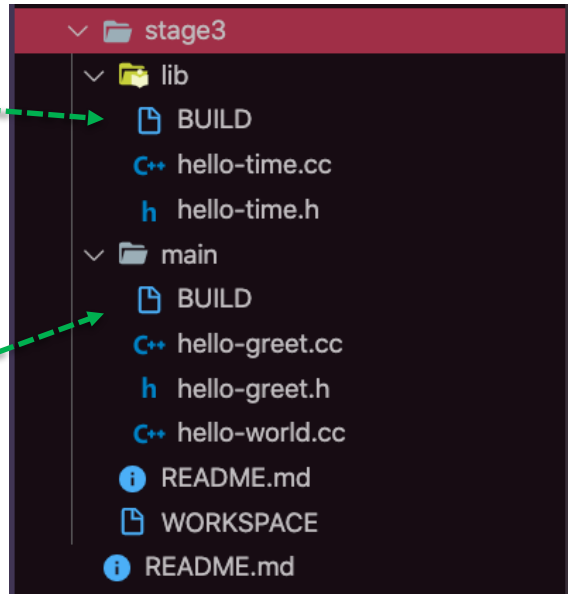


Multiple Packages

Package: A directory within the workspace that contains a BUILD file is a *package*.

```
cc_library(
  name = "hello-time",
  srcs = ["hello-time.cc"],
  hdrs = ["hello-time.h"],
  visibility = ["//main:__pkg__"],
)
```

```
cc_library(
  name = "hello-greet",
  srcs = ["hello-greet.cc"],
  hdrs = ["hello-greet.h"],
)

cc_binary(
  name = "hello-world",
  srcs = ["hello-world.cc"],
  deps = [
    ":hello-greet",
    "//lib:hello-time",
  ],
)
```

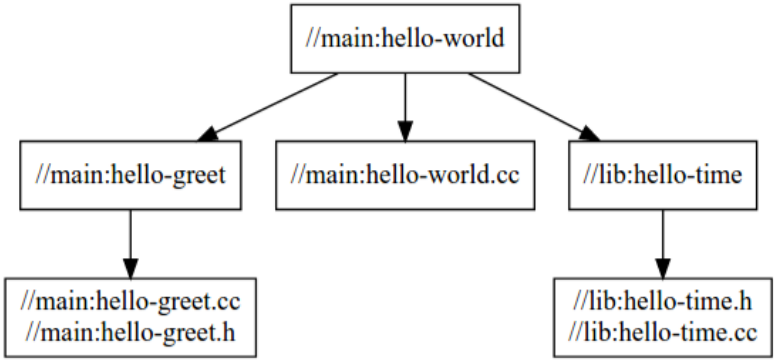


Multiple Packages

Package: A directory within the workspace that contains a BUILD file is a *package*.

```
cc_library(
  name = "hello-time",
  srcs = ["hello-time.cc"],
  hdrs = ["hello-time.h"],
  visibility = ["//main:__pkg__"],
)
```

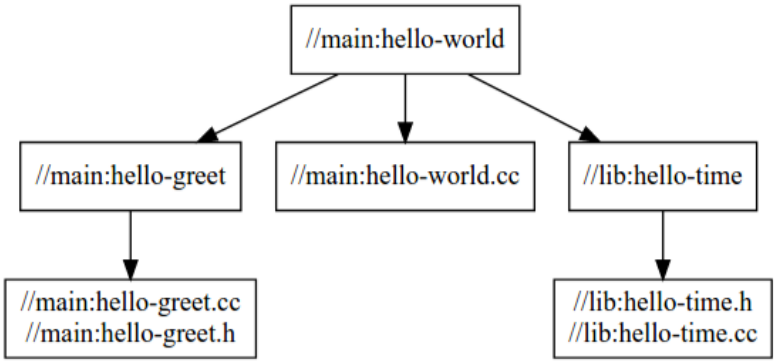
```
cc_library(
  name = "hello-greet",
  srcs = ["hello-greet.cc"],
  hdrs = ["hello-greet.h"],
)

cc_binary(
  name = "hello-world",
  srcs = ["hello-world.cc"],
  deps = [
    ":hello-greet",
    "//lib:hello-time",
  ],
)
```



Visibility

- By default targets are only visible to other targets in the **same BUILD file**.
- Bazel uses target visibility to prevent issues such as libraries containing implementation details leaking into public APIs (Encapsulation)

```
cc_library(
  name = "hello-time",
  srcs = ["hello-time.cc"],
  hdrs = ["hello-time.h"],
  visibility = ["//main:__pkg__"],
)
```

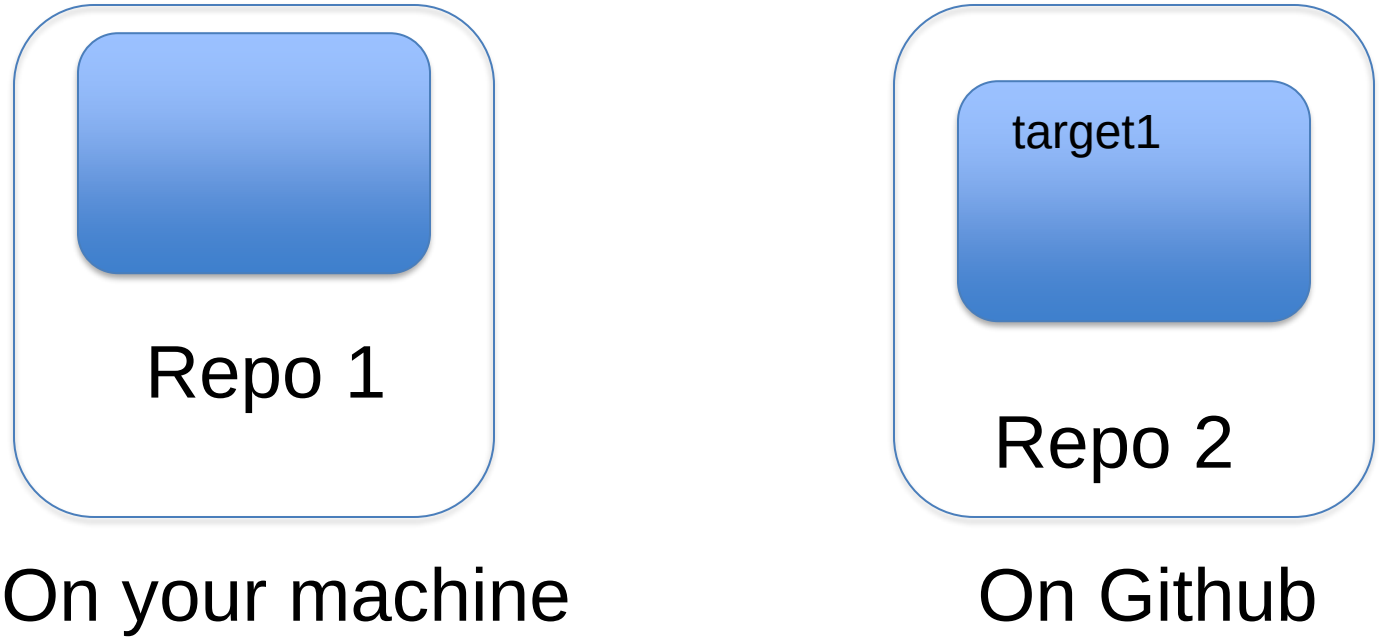


make the //lib:hello-time target in lib/BUILD explicitly visible to targets in main/BUILD

Bazel: Automatic Downloading of Git Repositories

git_repository rule

- Another great feature of Bazel is the support for unit testing
- What if you create Repo 1 which uses a target from Repo 2 that is on github? You have two options:
  - Bruin way: clone Repo 2 and include it in Repo 1
  - Trojan way: Make Bazel clone Repo 2 automatically



WORKSPACE File Rule

- Use git_repository rule in WORKSPACE file
  - Clone an external git repository.
- Here is an example:

```
git_repository(
  name = "com_google_absl",
  remote = "https://github.com/abseil/abseil-cpp.git",
  tag = "20200225.2",
)

git_repository(
  name = "com_google_benchmark",
  remote = "https://github.com/google/benchmark.git",
  tag = "v1.5.1",
)
```

Your WORKSPACE File

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

# GoogleTest/GoogleMock framework. Used by most unit-tests.
http_archive(
    name = "com_google_googletest",
    # Keep this URL in sync with ABSL_GOOGLETEST_COMMIT in ci/cmak
    urls = ["https://github.com/google/googletest/archive/8567b092
    strip_prefix = "googletest-8567b09290fe402cf01923e2131c5635b8e
    sha256 = "9a8a166eb6a56c7b3d7b19dc2c946fe4778fd6f21c7a12368ad3
)
```

Remote repo's WORKSPACE File

WORKSPACE File Rule

- Now we can reference targets of the remote in our repo
- Congrats! You are using com_google_absl without including it in your repo!

```
cc_binary(
  name = "main_flags_absl",
  srcs = ["main_flags_absl.cc"],
  deps = [
   "@com_google_absl//absl/flags:flag",
   "@com_google_absl//absl/flags:parse",
   "@com_google_absl//absl/flags:usage",
   "@glog",
],
)
```

Your BUILD File

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

# GoogleTest/GoogleMock framework. Used by most unit-tests.
http_archive(
    name = "com_google_googletest",
    # Keep this URL in sync with ABSL_GOOGLETEST_COMMIT in ci/cmak
    urls = ["https://github.com/google/googletest/archive/8567b092
    strip_prefix = "googletest-8567b09290fe402cf01923e2131c5635b8e
    sha256 = "9a8a166eb6a56c7b3d7b19dc2c946fe4778fd6f21c7a12368ad3
)
```

Remote repo's WORKSPACE File

Bazel for Unit Testing

Bazel for Unit Testing

- Another great feature of Bazel is the support for unit testing
- First, we ask Bazel to clone google test repo:

```
git_repository(
  name = "googletest",
  remote = "https://github.com/google/googletest",
  tag = "release-1.8.1",
)
```

Your WORKSPACE File

- Google Test Platform:
  - A testing framework for C++ code
  - Automates various tasks:
    - Creates a main function
    - Calls our function under test
    - Applies inputs
    - Provides various functions for testing

Bazel for Unit Testing

- Next, create a folder called tests in the root of your repo

```
git_repository(
  name = "com_google_googletest",
  remote = "https://github.com/google/googletest",
  tag = "release-1.8.1",
)
```

Your WORKSPACE File

```
cc_test(
  name = "cpplib_test",
  srcs = [”cpplib_test.cc"],
  deps = [
    "//src/lib:CPPLib",
  "@com_google_googletest//:gtest_main",
  ],
)
```

Your tests/BUILD File

A sample test file

- Next, create a folder called tests in the root of your repo
- The test doesn't require a main function.

```
#include "src/lib/cpplib.h"

#include <map>
#include <vector>

#include "gtest/gtest.h"

TEST(CPPLibTest, ReturnHelloWorld) {
  CPPLib cpplib;
  std::string actual = cpplib.PrintHelloWorld();
  std::string expected = "**** Hello World ****";
  EXPECT_EQ(expected, actual);
}
```

Your WORKSPACE File

```
cc_test(
  name = "cpplib_test",
  srcs = [”cpplib_test.cc"],
  deps = [
    "//src/lib:CPPLib",
    "@com_google_googletest//:gtest_main",
  ],
)
```

Your tests/BUILD File

```
> bazel test tests/cpplib_test
```

Command line to run the test

Google Test Macros

```
#include "src/lib/cpplib.h"

#include <map>
#include <vector>

#include "gtest/gtest.h"

TEST(CPPLibTest, ReturnHelloWorld) {
  CPPLib cpplib;
  std::string actual = cpplib.PrintHelloWorld();
  std::string expected = "**** Hello World ****";
  EXPECT_EQ(expected, actual);
}
```

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

- `ASSERT_*` yields a fatal failure and returns from the current function.
- `EXPECT_*` yields a nonfatal failure, allowing the function to continue running.

Google Test Macros

Fatal assertion	Nonfatal assertion	Verifies
`ASSERT_EQ(val1, val2);`	`EXPECT_EQ(val1, val2);`	`val1 == val2`
`ASSERT_NE(val1, val2);`	`EXPECT_NE(val1, val2);`	`val1 != val2`
`ASSERT_LT(val1, val2);`	`EXPECT_LT(val1, val2);`	`val1 < val2`
`ASSERT_LE(val1, val2);`	`EXPECT_LE(val1, val2);`	`val1 <= val2`
`ASSERT_GT(val1, val2);`	`EXPECT_GT(val1, val2);`	`val1 > val2`
`ASSERT_GE(val1, val2);`	`EXPECT_GE(val1, val2);`	`val1 >= val2`

Google Test provides more features:

https://github.com/google/googletest

Test Fixtures

Test Fixtures

- Used to do common actions in one place.
  - SetUp(): runs at the beginning of each TEST_F
  - TearDown() (): runs at the end of each TEST_F
- See **tests/test_fixture.cc** in cpp-template repo for the complete example.

```
template <typename E>
class Queue {
 private:
  std::vector<E> _v;

 public:
  Queue() {}
  void Enqueue(const E& element);
  // Throws the queue is empty.
  E Dequeue;
  size_t size() const ;
  bool IsEmpty();
};
```

Class we want to test

```
class QueueTest : public ::testing::Test {
protected:
void SetUp() override {
  q1_.Enqueue(1);
  q2_.Enqueue(2);
  q2_.Enqueue(3);
}

void TearDown() override {
  std::cout << "Test ended!" << std::endl;
}

  Queue<int> q0_;
  Queue<int> q1_;
  Queue<int> q2_;
};
```

Class for Test Fixture

```
TEST_F(QueueTest, IsEmptyInitially) {
  EXPECT_EQ(q0_.size(), 0);
  EXPECT_EQ(q0_.IsEmpty(), true);
}
```

Use TEST_F instead of TEST

Google Mock (GMOCK)

Full Documentation:
https://google.github.io/googletest/

GMOCK

```
class Turtle {
public:
virtual ~Turtle() {}
virtual void PenUp() = 0;
virtual void PenDown() = 0;
virtual void Forward(int distance) = 0;
virtual void Turn(int degrees) = 0;
virtual void GoTo(int x, int y) = 0;
virtual int GetX() const = 0;
virtual int GetY() const = 0;
};
```

```
#include <platform_dependant_file.h>
class TurtleMac1080 : public Turtle {

 // Actual implementation of Turtle APIs
};
```

```
class Painter {
  Turtle* turtle;

 public:
  Painter(Turtle* turtle) : turtle(turtle) {}

  bool DrawCircle(int x, int y, int r) {
    int next_y;
    turtle->GoTo(x, y + r);
    turtle->PenDown();
    turtle->PenUp();
    next_y = turtle->GetY();
    ...
    return true;
  }
};
TurtleMac1080 turtle;

Painter painter(&turtle);
```

- Problem:
  - How can we test Painter class without depending on `TurtleMac1080`?
    - Ideally, we should not need to include `platform_dependant_file.h` in the test file for Painter and we should not worry about any linking all library files that `TurtleMac1080` uses for testing.
  - Ideally when we test class A that depends on and calls class B's API, we want the call to B's API's to be simple and cheap. For example, we should avoid:
    - RPC calls, database calls, credit card transaction calls, etc.
- Solution: Pass a mock class to Painter when we test it.

GMOCK – gmock1.cc

```
class MockTurtle : public Turtle {
public:
MOCK_METHOD0(PenUp, void());
MOCK_METHOD0(PenDown, void());
MOCK_METHOD1(Forward, void(int
distance));
MOCK_METHOD1(Turn, void(int degrees));
MOCK_METHOD2(GoTo, void(int x, int y));
MOCK_CONST_METHOD0(GetX, int());
MOCK_CONST_METHOD0(GetY, int());
};
```

```
TEST(PainterTest, CanDrawCircle) {
MockTurtle turtle; // #2
EXPECT_CALL(turtle, PenDown()) // #3
.Times(AtLeast(1));

EXPECT_CALL(turtle, GoTo(0, 10)) // #3
.Times(Exactly(1));

EXPECT_CALL(turtle, GoTo(0, -10)) // #3
.Times(Exactly(1));

EXPECT_CALL(turtle, GetY())
.Times(2)
.WillOnce(Return(100))
.WillOnce(Return(150))
.WillRepeatedly(Return(200));

Painter painter(&turtle); // #4

EXPECT_TRUE(painter.DrawCircle(0, 0, 10)); // #5
}
```

- We inherit a class from Turtle and use GMOCK macros to mock each API:
  - 0 in **MOCK_METHOD0** means the API takes 0 arguments.
  - We pass this mock object to Painter's constructor.
  - See tests/gmock/gmock1.cc in [cpp-template](#) repo for the complete example.

GMOCK

List expectations

Exercise some code and check the result

```
TEST(PainterTest, CanDrawCircle) {
// Wrap in NiceMock to suppress uninteresting warnings from GMock.
// These warnings happen when an API is called, but there is no
expectation
// for it.
NiceMock<MockTurtle> turtle; // #2
EXPECT_CALL(turtle, PenDown()) // #3
.Times(AtLeast(1));

EXPECT_CALL(turtle, GoTo(0, 10)) // #3
.Times(1);

EXPECT_CALL(turtle, GoTo(0, -10)) // #3
.Times(1);


Painter painter(&turtle); // #4

EXPECT_TRUE(painter.DrawCircle(0, 0, 10)); // #5
}
```

- As the code is exercised GMOCK monitors API calls and checks them against the list of expectations in reverse order.
  - See tests/gmock/gmock1.cc in cpp-template repo for the complete example.

GMOCK – gmock2.cc

```
class MockTurtle : public Turtle {
public:
MOCK_METHOD0(PenUp, void());
MOCK_METHOD0(PenDown, void());
MOCK_METHOD1(Forward, void(int
distance));
MOCK_METHOD1(Turn, void(int degrees));
MOCK_METHOD2(GoTo, void(int x, int y));
MOCK_CONST_METHOD0(GetX, int());
MOCK_CONST_METHOD0(GetY, int());
};
```

```
class Turtle {
public:
MOCK_METHOD0(PenUp, void());
MOCK_METHOD0(PenDown, void());
MOCK_METHOD1(Forward, void(int distance));
MOCK_METHOD1(Turn, void(int degrees));
MOCK_METHOD2(GoTo, void(int x, int y));
MOCK_CONST_METHOD0(GetX, int());
MOCK_CONST_METHOD0(GetY, int());
};
```

```
TEST(PainterTest, CanDrawCircle) {
NiceMock<MockTurtle> turtle; // #2
...
}
```

```
TEST(PainterTest, CanDrawCircle) {
NiceMock<Turtle> turtle; // #2
...
}
```

- MockTurtle is inheriting from Turtle, that means there is still dependency on Turtle.
- We can remove that by completely replacing Turtle class with its mock version

GMOCK Quirks – gmock3.cc

```
TEST(PainterTest,
CanDrawCircleGeneralRuleOnTheTop) {
NiceMock<MockTurtle> turtle;

EXPECT_CALL(turtle, GoTo(_, _)).Times(1);
EXPECT_CALL(turtle, GoTo(0, -10)).Times(1);

Painter painter(&turtle);

EXPECT_TRUE(painter.DrawCircle(0, 0, 10));
}
```

```
TEST(PainterTest,
CanDrawCircleGeneralRuleOnTheBottom) {
NiceMock<MockTurtle> turtle;

EXPECT_CALL(turtle, GoTo(0, -10)).Times(1);
EXPECT_CALL(turtle, GoTo(_, _)).Times(1);

Painter painter(&turtle);

EXPECT_TRUE(painter.DrawCircle(0, 0, 10));
}
```

1. Rules are sticky! If a rule matches to the first call of GoTo. It is still active for the second call to GoTo.
2. Rules match in reverse order.
3. Each rule should be satisfied.
4. Using '_' for parameters means match to any value.

Example:

- The left one passes:
  - The first GoTo(0,10) is matched with the top rule and the second GoTo(0,-10) is matched to the bottom rule. Both rules are now matched and saturated.
- The right one does not pass
  - The first GoTo(0,10) is matched with the bottom. The second GoTo(0,-10) is matched to the bottom rule too, but it says GoTo is called only once. So it fails. The top rule never matches so that fails too!

GMOCK Actions – gmock4.cc

```
TEST(PainterTest, CanDrawCircle) {
NiceMock<Turtle> turtle;
EXPECT_CALL(turtle,
PenDown()).Times(AtLeast(2));

EXPECT_CALL(turtle, GoTo(0, 10)).Times(1);

EXPECT_CALL(turtle, GoTo(0, -10)).Times(1);

ON_CALL(turtle, SomeRandomFunction
  (_, _, _))
  .WillByDefault(Return(10));

Painter painter(&turtle);

EXPECT_TRUE(painter.DrawCircle(0, 0, 10));
}
```

```
bool DrawCircle(int x, int y, int r) {
  auto res = turtle->SomeRandomFunction(3,
                                        2, 1);
  std::cout << "res: " <<
    res << std::endl;

  res = turtle->SomeRandomFunction(2,
                                        3, 1);
  std::cout << "res: " << res << std::endl;
...
}
```

```
Output:
res: 10
res: 10
```

- ON_CALL specifies what to do when a certain API is called.
- Each time SomeRandomFunction() is called, with any parameter value, it returns 10.

GMOCK Actions – gmock5.cc

```
TEST(PainterTest, CanDrawCircle) {
NiceMock<Turtle> turtle;
EXPECT_CALL(turtle,
PenDown()).Times(AtLeast(2));

EXPECT_CALL(turtle, GoTo(0, 10)).Times(1);

EXPECT_CALL(turtle, GoTo(0, -10)).Times(1);

EXPECT_CALL(turtle, SomeRandomFunction(_, _, _))
.Times(AtLeast(1))
.WillRepeatedly(Return(10));

Painter painter(&turtle);

EXPECT_TRUE(painter.DrawCircle(0, 0, 10));
}
```

```
bool DrawCircle(int x, int y, int r) {
  auto res = turtle->SomeRandomFunction(3,
                                        2, 1);
  std::cout << "res: " <<
    res << std::endl;

  res = turtle->SomeRandomFunction(2,
                                        3, 1);
  std::cout << "res: " << res << std::endl;
...
}
```

```
Output:
res: 10
res: 10
```

- Similar thing with EXPECT_CALL instead. In this case, it not only answers, but also fails the test if the function is not called.
- AtLeast(1) matches to values 1 and higher.

GMOCK Actions – gmock5.cc

```
TEST(PainterTest, CanDrawCircle2) {
NiceMock<Turtle> turtle;
EXPECT_CALL(turtle,
PenDown()).Times(AtLeast(2));

EXPECT_CALL(turtle, GoTo(0, 10)).Times(1);

EXPECT_CALL(turtle, GoTo(0, -10)).Times(1);

EXPECT_CALL(turtle, SomeRandomFunction(_, _, _))
.Times(2)
.WillOnce(Return(10))
.WillOnce(Return(20));

Painter painter(&turtle);

EXPECT_TRUE(painter.DrawCircle(0, 0, 10));
}
```

```
bool DrawCircle(int x, int y, int r) {
  auto res = turtle->SomeRandomFunction(3,
                                        2, 1);
  std::cout << "res: " <<
    res << std::endl;

  res = turtle->SomeRandomFunction(2,
                                        3, 1);
  std::cout << "res: " << res << std::endl;
...
}
```

```
Output:
res: 10
res: 20
```

- Similar thing with EXPECT_CALL. Once we return 10, once we return 20.

GMOCK Actions – gmock6.cc

```
int MyGetY() {
std::cout << "Hello from MyGetY!" << std::endl;
return 0;
}

TEST(PainterTest, CanDrawCircle) {
NiceMock<Turtle> turtle; // #2

EXPECT_CALL(turtle, GetY())
.Times(5)
.WillOnce(Invoke(MyGetY))
.WillRepeatedly(Return(200));

Painter painter(&turtle); // #4

EXPECT_TRUE(painter.DrawCircle(0, 0, 10)); // #5
}
```

```
bool DrawCircle(int x, int y, int r) {
int next_y;
turtle->GoTo(x, y + r);
turtle->PenDown();
turtle->PenUp();
next_y = turtle->GetY();
std::cout << "next_y: " << next_y <<
std::endl;
...
}
```

```
Output:
Hello from MyGetY!
next_y: 0
next_y: 200
next_y: 200
next_y: 200
next_y: 200
```

- We can call another function in response to an API call instead of return using Invoke.

Other Actions (Invoke Instead of Return)

Using a Function, Functor, or Lambda as an Action

In the following, by "callable" we mean a free function, `std::function`, functor, or lambda.

`f`	Invoke f with the arguments passed to the mock function, where f is a callable.
`Invoke(f)`	Invoke `f` with the arguments passed to the mock function, where `f` can be a global/static function or a functor.
`Invoke(object_pointer, &class::method)`	Invoke the method on the object with the arguments passed to the mock function.
`InvokeWithoutArgs(f)`	Invoke `f`, which can be a global/static function or a functor. `f` must take no arguments.
`InvokeWithoutArgs(object_pointer, &class::method)`	Invoke the method on the object, which takes no arguments.
`InvokeArgument<N>(arg1, arg2, ..., argk)`	Invoke the mock function's `N`-th (0-based) argument, which must be a function or a functor, with the `k` arguments.

The return value of the invoked function is used as the return value of the action.

When defining a callable to be used with `Invoke*()`, you can declare any unused parameters as `Unused`:

```
using ::testing::Invoke;
double Distance(Unused, double x, double y) { return sqrt(x*x + y*y); }
...
EXPECT_CALL(mock, Foo("Hi", _, _)).WillOnce(Invoke(Distance));
```

Other Matchers

Generic Comparison

Matcher	Description
`Eq(value)` or `value`	`argument == value`
`Ge(value)`	`argument >= value`
`Gt(value)`	`argument > value`
`Le(value)`	`argument <= value`
`Lt(value)`	`argument < value`
`Ne(value)`	`argument != value`
`IsFalse()`	`argument` evaluates to `false` in a Boolean context.
`IsTrue()`	`argument` evaluates to `true` in a Boolean context.
`IsNull()`	`argument` is a `NULL` pointer (raw or smart).
`NotNull()`	`argument` is a non-null pointer (raw or smart).
`Optional(m)`	`argument` is `optional<>` that contains a value matching `m`. (For testing whether an `optional<>` is set, check for equality with `nullopt`. You may need to use `Eq(nullopt)` if the inner type doesn't have `==`.)
`VariantWith<T>(m)`	`argument` is `variant<>` that holds the alternative of type T with a value matching `m`.
`Ref(variable)`	`argument` is a reference to `variable`.
`TypedEq<type>(value)`	`argument` has type `type` and is equal to `value`. You may need to use this instead of `Eq(value)` when the mock function is overloaded.

Other Matchers

String Matchers

The `argument` can be either a C string or a C++ string object:

Matcher	Description
`ContainsRegex(string)`	`argument` matches the given regular expression.
`EndsWith(suffix)`	`argument` ends with string `suffix`.
`HasSubstr(string)`	`argument` contains `string` as a sub-string.
`IsEmpty()`	`argument` is an empty string.
`MatchesRegex(string)`	`argument` matches the given regular expression with the match starting at the first character and ending at the last character.
`StartsWith(prefix)`	`argument` starts with string `prefix`.
`StrCaseEq(string)`	`argument` is equal to `string`, ignoring case.
`StrCaseNe(string)`	`argument` is not equal to `string`, ignoring case.
`StrEq(string)`	`argument` is equal to `string`.
`StrNe(string)`	`argument` is not equal to `string`.

Tons of other matchers and actions:

- https://google.github.io/googletest/gmock_cheat_sheet.html
- https://google.github.io/googletest/gmock_cook_book.html

How to use Google Benchmark for C++

By Ari Saif
https://www.youtube.com/c/arisaif

How Long Does My Function Take?

```
void SomeFunction(int a, int b){
  // ...
}
```

How long does this take?

```
static void BM_SomeFunction(benchmark::State& state) {
// Perform setup here
  for (auto _ : state) {
    // This code gets timed
    SomeFunction()
  }
}
```

Why Google Benchmark?

Why not simply write this?

```
int main() {

  auto start = std::chrono::high_resolution_clock::now();
  SomeFunction(100, 200);
  auto stop = std::chrono::high_resolution_clock::now();

  auto duration = std::chrono::duration_cast<microseconds>(stop - start);
}
```

Because don't reinvent the wheel!

Why Google Benchmark?

- We could use a timer and measure the time ourselves.

- Some convenient features from Google Benchmark:
  - Running the measurement multiple times and report the time once the result was stable.
  - Some approximate runtime complexity
  - Various report formats (JSON, CSV, Text)
  - Sweep the parameters and repeat the measurement

Sweep Parameters

Sweep from 256 to 2^18

```
void BinarySearch(int size, ...){
  // ...
}
```

```
-----------------------------------------------------------------------------------------
Benchmark                          Time             CPU   Iterations UserCounters...
-----------------------------------------------------------------------------------------
BM_BinarySearch/256             1122 ns         1121 ns       629655 items_per_second=228.371M/s
BM_BinarySearch/512             1114 ns         1113 ns       630687 items_per_second=460.137M/s
BM_BinarySearch/1024            1135 ns         1131 ns       617840 items_per_second=905.102M/s
BM_BinarySearch/2048            1168 ns         1163 ns       601034 items_per_second=1.7611G/s
BM_BinarySearch/4096            1198 ns         1188 ns       595623 items_per_second=3.4477G/s
BM_BinarySearch/8192            1611 ns         1599 ns       424567 items_per_second=5.12458G/s
BM_BinarySearch/16384           1670 ns         1640 ns       431880 items_per_second=9.99125G/s
BM_BinarySearch/32768           2305 ns         2199 ns       335561 items_per_second=14.9041G/s
BM_BinarySearch/65536           2361 ns         2253 ns       309598 items_per_second=29.088G/s
BM_BinarySearch/131072          2419 ns         2287 ns       306363 items_per_second=57.3102G/s
BM_BinarySearch/262144          2354 ns         2218 ns       313731 items_per_second=118.17G/s
```

Runtime Complexity

```
void BinarySearch(int size, ...){
  // ...
}
```

Runtime Complexity

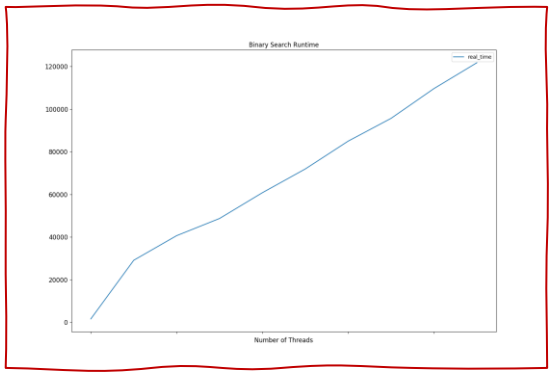
```
BM_BinarySearch_BigO          130.92 lgN
```

Various Output Formats

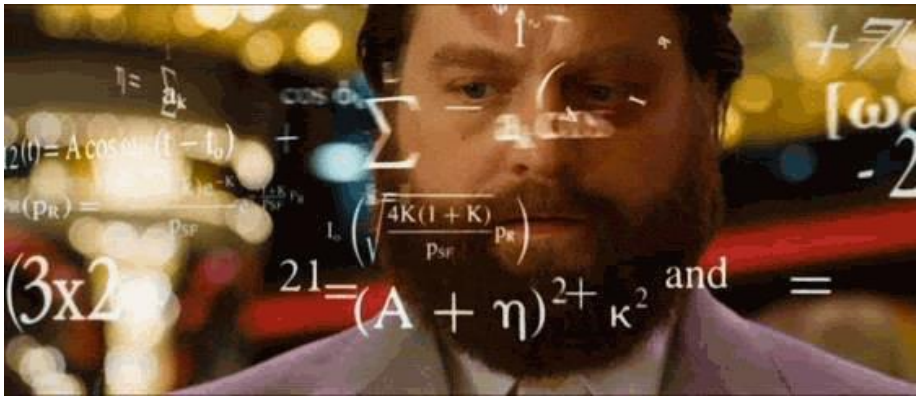

```
{
"context": {
"date": "2020-06-23 14:58:29",
"host_name": "myhost",
"executable": ".../execroot/__main__/bazel-out/darwin-
opt/bin/src/benchmark/main_benchmark_bad_example",
"num_cpus": 12,
"mhz_per_cpu": 2600,
"cpu_scaling_enabled": false,
"load_avg": [2.57812,2.3916,2.11328],
"library_build_type": "release"
},
"benchmarks": [
{
"name": "BM_Increment/256/1",
"run_name": "BM_Increment/256/1",
"run_type": "iteration",
"repetitions": 0,
"repetition_index": 0,
"threads": 1,
"iterations": 1000000000,
"real_time": 1.1369702406227589e-06,
"cpu_time": 1.0000000000010001e-06,
"time_unit": "ns"
},
{
"name": "BM_Increment/512/1",
"run_name": "BM_Increment/512/1",
"run_type": "iteration",
"repetitions": 0,
"repetition_index": 0,
"threads": 1,
"iterations": 1000000000,
"real_time": 1.1190422810614109e-06,
"cpu_time": 1.0000000000010001e-06,
"time_unit": "ns"
},
```



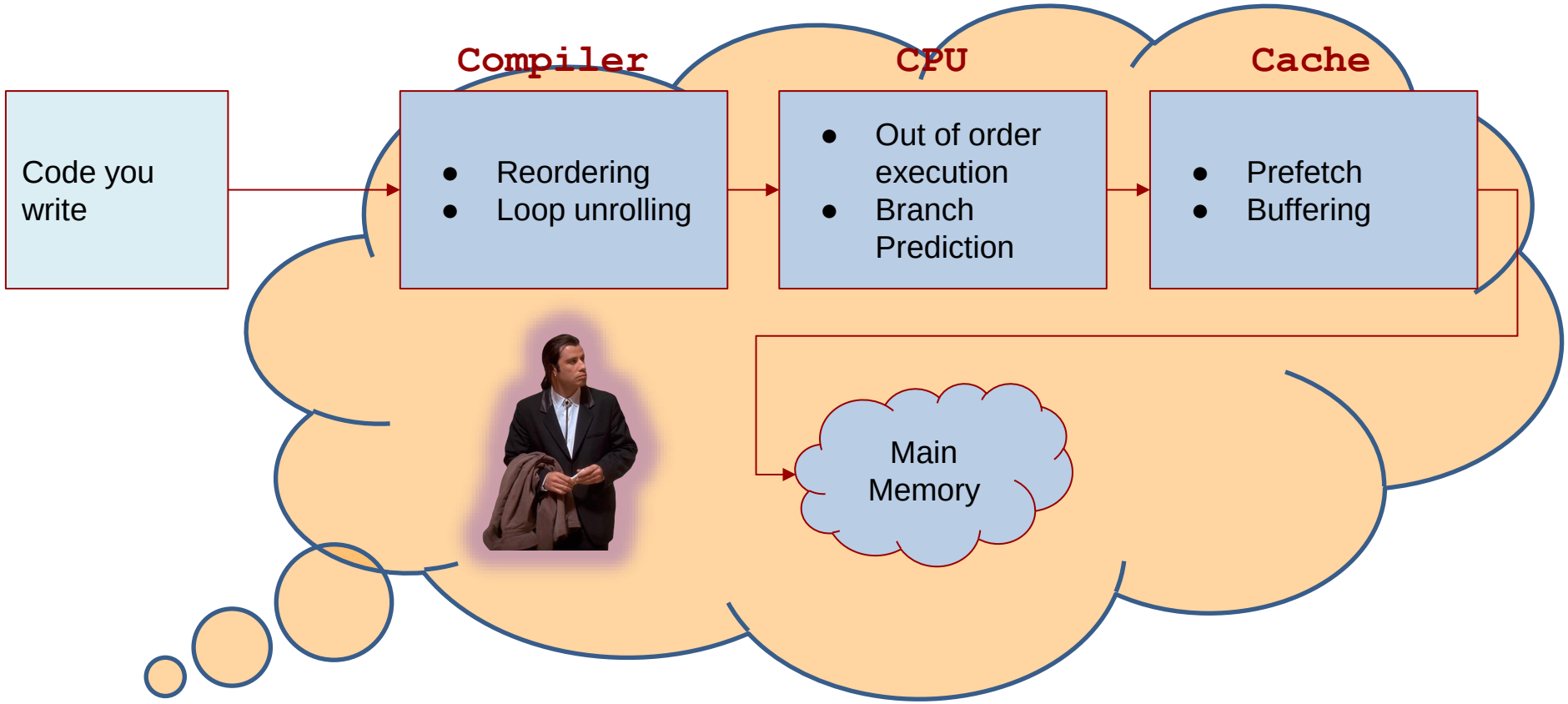
```
name,iterations,real_time,cpu_time,time_unit,bytes_per_second,items_per_second,label,error_occurred,error_message
"BM_Increment/256/1",1000000000,1.507e-06,2e-06,ns,,,,,
"BM_Increment/512/1",1000000000,1.60706e-06,1e-06,ns,,,,,
"BM_Increment/1024/1",1000000000,1.58802e-06,1e-06,ns,,,,,
"BM_Increment/256/2",1000000000,1.94e-06,2e-06,ns,,,,,
"BM_Increment/512/2",1000000000,1.66398e-06,2e-06,ns,,,,,
"BM_Increment/1024/2",1000000000,1.85601e-06,2e-06,ns,,,,,
"BM_Increment/256/4",1000000000,1.662e-06,2e-06,ns,,,,,
"BM_Increment/512/4",1000000000,1.54896e-06,2e-06,ns,,,,,
"BM_Increment/1024/4",1000000000,1.59396e-06,2e-06,ns,,,,,
"BM_Increment/256/8",1000000000,1.74e-06,1e-06,ns,,,,,
"BM_Increment/512/8",1000000000,1.53901e-06,2e-06,ns,,,,,
"BM_Increment/1024/8",1000000000,1.68901e-06,1e-06,ns,,,,,
```

Why is Benchmarking Hard?



Why is Benchmarking Hard?



"The runtime of your program depends on many factors…"

"What you measure is probably accurate only on your machine."

Google Benchmark

google / benchmark

Watch 218 | Unstar 4.5k | Fork 860

Code | Issues 107 | Pull requests 26 | Actions | Projects 0 | Security 0 | Insights

Branch: master | benchmark / README.md | Find file | Copy path

chfast Relax CHECK condition in benchmark_runner.cc (#938) c078337 on Feb 21

49 contributors and others

1300 lines (1026 sloc) | 44 KB | Raw | Blame | History

Benchmark

build passing | build passing | coverage 92% | slack 43

A library to benchmark code snippets, similar to unit tests. Example:

```
#include <benchmark/benchmark.h>

static void BM_SomeFunction(benchmark::State& state) {
  // Perform setup here
  for (auto _ : state) {
    // This code gets timed
    SomeFunction();
  }
}
// Register the function as a benchmark
BENCHMARK(BM_SomeFunction);
// Run the benchmark
BENCHMARK_MAIN();
```

https://github.com/google/benchmark

Prerequisites

Prerequisite: Installing Bazel

This repo uses `Bazel` for building C++ files. You can install Bazel using this link.

Cloning this repo

```
git clone https://github.com/ourarash/cpp-template.git
```

https://github.com/ourarash/cpp-template

***It is also available for CMake**

Structure of Benchmarks

Function to benchmark

```
unsigned long Increment(unsigned long n) {
  unsigned long sum = 0;
  for (unsigned long i = 0; i < n; i++) {
    sum++;
  }
  return sum;
}
```

A wrapper used by the framework

```
static void BM_Increment(benchmark::State& state) {
  // Perform setup here
  for (auto _ : state) {
    // This code gets timed
    Increment(state.range(0));
  }
}
```

Registering and running the benchmark

```
// Register the function as a benchmark
BENCHMARK(BM_Increment);

BENCHMARK_MAIN();
```

Passing an Argument

```
static void BM_Increment(benchmark::State& state) {
  // Perform setup here
  for (auto _ : state) {
    // This code gets timed
    Increment(state.range(0));
  }
}
```

```
BENCHMARK(BM_Increment)->Arg(1000);
BENCHMARK(BM_Increment)->Arg(2000);
```

Passing Multiple Arguments

```
static void BM_AddByValue(benchmark::State& state) {
// Perform setup here
for (auto _ : state) {
// This code gets timed
AddByValue(state.range(0), state.range(1));
}
}
```

```
BENCHMARK(BM_AddByValue)->Args({2000, 2});
```

Multiple Runs – Sweep Input

```
BENCHMARK(BM_Increment)->Arg(1 << 8)->Arg(1 << 9)->Arg(1 << 10);
```

```
BENCHMARK(BM_Increment)->RangeMultiplier(2)->Range(1 << 8, 1 << 10);
```

Sweep the first argument from 2^8 to 2^10,
each time multiply by 2

Multiple Runs – Sweep Multiple Inputs

```
BENCHMARK(BM_Increment)
  ->RangeMultiplier(2)
  ->Ranges({{1 << 8, 1 << 10}, {1, 5}});
```

```
-----------------------------------------------------------------
Benchmark                      Time             CPU   Iterations
-----------------------------------------------------------------
BM_Increment/256/1           395 ns          394 ns      1511399
BM_Increment/512/1           740 ns          734 ns      1040490
BM_Increment/1024/1         1347 ns         1341 ns       484315
BM_Increment/256/2           347 ns          346 ns      2002758
BM_Increment/512/2           664 ns          664 ns      1038853
BM_Increment/1024/2         1381 ns         1375 ns       513769
BM_Increment/256/4           355 ns          354 ns      1810334
BM_Increment/512/4           684 ns          681 ns      1035626
BM_Increment/1024/4         1430 ns         1424 ns       532583
BM_Increment/256/8           429 ns          427 ns      1596231
BM_Increment/512/8           688 ns          686 ns      1041667
BM_Increment/1024/8         1332 ns         1329 ns       525980
```

Compile Optimization Flags

gcc optimization flags

option	optimization level
-O0	optimization for compilation time (default)
-O1 or -O	the compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time.
-O2	Optimize even more. GCC performs nearly all supported optimizations that do not involve a space-speed tradeoff.
-O3	Optimize yet more. -O3 turns on all optimizations specified by -O2 and more.
-Os	optimization for code size
-Ofast	O3 with fast none accurate math calculations
-Og	Optimize debugging experience

Bazel Optimization Flags

`--compilation_mode or -c`

option	optimization level
fastbuild	build as fast as possible: generate minimal debugging information (`-gmlt -Wl,-S`), and don't optimize. This is the default. Note: -DNDEBUG will not be set.
dbg	build with debugging enabled (-g), so that you can use gdb (or another debugger).
opt	build with optimization enabled and with assert() calls disabled (`-O2 -DNDEBUG`). Debugging information will not be generated in opt mode unless you also pass --copt -g.

`--copt`

Takes an argument to be passed to the compiler.

```
bazel run --cxxopt='-std=c++17' src/benchmark/main_benchmark_bad_example -c opt
```

```
bazel run --cxxopt='-std=c++17' src/benchmark/main_benchmark_bad_example -c opt --copt=-O3
```

Preventing Optimization

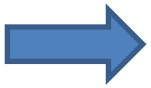
```
unsigned long Increment(unsigned long n) {
  unsigned long sum = 0;
  for (unsigned long i = 0; i < n; i++) {
    benchmark::DoNotOptimize(sum++);
    benchmark::ClobberMemory();
  }
  return sum;
}
```

forces the result to be stored in either memory or a register

forces the compiler to perform all pending writes to global memory

Preventing Optimization

```
int foo(int x) { return x + 42; }
while (...) DoNotOptimize(foo(0));
```



```
int foo(int x) { return x + 42; }
while (...) DoNotOptimize(42);
```

```
static void BM_vector_push_back(benchmark::State& state) {
  for (auto _ : state) {
    std::vector<int> v;
    v.reserve(1);
    // Allow v.data() to be clobbered.
    benchmark::DoNotOptimize(v.data());

    v.push_back(42);
    benchmark::ClobberMemory(); // Force 42 to be written to memory.
  }
}
```

Pause and Resume the Timer

```
static void BM_TernarySearch(benchmark::State& state) {
  for (auto _ : state) {
    state.PauseTiming();

    auto d = init(state.range(0));

    state.ResumeTiming();

    Search<unsigned long>::TernarySearch(d.v, d.v[d.v.size() - 1]);
  }
}
```

The runtime of this line will be ignored.

In-class Activity

Measuring Runtime Complexity

Measuring Complexity

Add this at the end where state.range(0) is the size of the problem (i.e. n)

```
static void BM_TernarySearch(benchmark::State& state) {
  for (auto _ : state) {
    state.PauseTiming();
    auto d = init(state.range(0));
    state.ResumeTiming();
    Search<unsigned long>::TernarySearch(d.v, d.v[d.v.size() - 1]);
  }
  state.SetComplexityN(state.range(0));
}
```

```
BENCHMARK(BM_TernarySearch)
->RangeMultiplier(2)
->Range(1 << 8, 1 << 18)
->Complexity();
```

Add this at the end

Measuring Complexity

Add this at the end where state.range(0) is the size of the problem (i.e. n)

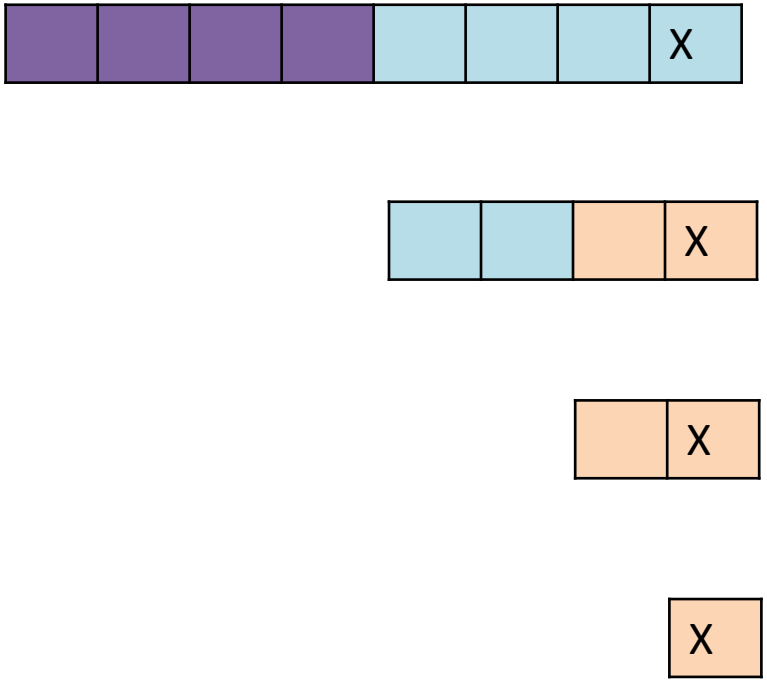
```
static void BM_TernarySearch(benchmark::State& state) {
  for (auto _ : state) {
    state.PauseTiming();
    auto d = init(state.range(0));
    state.ResumeTiming();
    Search<unsigned long>::TernarySearch(d.v, d.v[d.v.size() - 1]);
  }
  state.SetComplexityN(state.range(0));
}
```

```
BENCHMARK(BM_TernarySearch)
->RangeMultiplier(2)
->Range(1 << 8, 1 << 18)
->Complexity(benchmark::oLogN);
```

Add this at the end

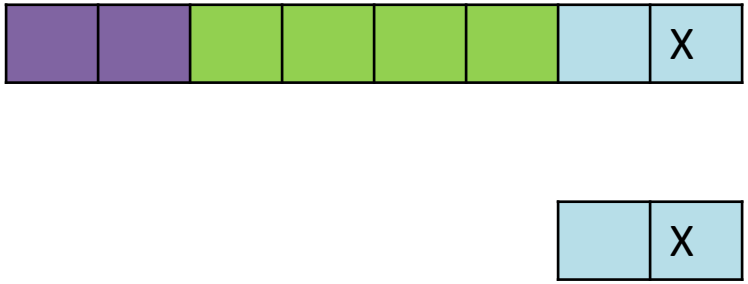
Binary or Ternary

Binary Search



O(Log n / Log 2)

Ternary Search



O(Log n / Log 3)

```
Benchmark                          Time             CPU   Iterations UserCounters...
-----------------------------------------------------------------------------------------
BM_BinarySearch/32768            1934 ns         1875 ns       371793 items_per_second=17.4747G/s
BM_BinarySearch/65536            2188 ns         2095 ns       335178 items_per_second=31.2838G/s
BM_BinarySearch/131072           2276 ns         2146 ns       326738 items_per_second=61.0796G/s
BM_BinarySearch/262144           2338 ns         2189 ns       319956 items_per_second=119.751G/s
BM_BinarySearch/524288           2585 ns         2395 ns       294375 items_per_second=218.887G/s
BM_BinarySearch_BigO           133.27 lgN      125.80 lgN
BM_BinarySearch_RMS                 2 %             2 %
BM_TernarySearch/32768           1938 ns         1880 ns       371374 items_per_second=17.4317G/s
BM_TernarySearch/65536           2179 ns         2083 ns       335358 items_per_second=31.4622G/s
BM_TernarySearch/131072          2236 ns         2113 ns       332568 items_per_second=62.0399G/s
BM_TernarySearch/262144          2305 ns         2165 ns       325388 items_per_second=121.07G/s
BM_TernarySearch/524288          2574 ns         2381 ns       294146 items_per_second=220.171G/s
BM_TernarySearch_BigO          132.20 lgN      124.85 lgN
BM_TernarySearch_RMS                2 %             3 %
BM_ExponentialSearch/32768       1970 ns         1910 ns       366235 items_per_second=17.1567G/s
BM_ExponentialSearch/65536       2212 ns         2119 ns       333371 items_per_second=30.9247G/s
BM_ExponentialSearch/131072      2271 ns         2148 ns       326088 items_per_second=61.0144G/s
BM_ExponentialSearch/262144      2357 ns         2215 ns       317270 items_per_second=118.35G/s
BM_ExponentialSearch/524288      2609 ns         2428 ns       287544 items_per_second=215.9G/s
BM_ExponentialSearch_BigO      134.40 lgN      127.21 lgN
BM_ExponentialSearch_RMS            2 %             2 %
BM_BinarySearchPar/32768/2      30104 ns        28324 ns        24729 items_per_second=1.15689G/s
BM_BinarySearchPar/65536/2      30377 ns        28582 ns        24496 items_per_second=2.29288G/s
BM_BinarySearchPar/131072/2     30420 ns        28692 ns        24800 items_per_second=4.5683G/s
BM_BinarySearchPar/262144/2     30634 ns        28658 ns        24478 items_per_second=9.14736G/s
BM_BinarySearchPar/524288/2     35537 ns        26913 ns        26014 items_per_second=19.4805G/s
BM_BinarySearchPar_BigO       1842.86 lgN     1647.52 lgN
BM_BinarySearchPar_RMS              5 %            10 %
```

Pay Attention to Numbers…

```
CPU Caches:
  L1 Data 32 KiB (x6)
  L1 Instruction 32 KiB (x6)
  L2 Unified 256 KiB (x6)
  L3 Unified 9216 KiB (x1)
Load Average: 2.97, 2.64, 2.53
---------------------------------------------------------------------------
Benchmark                              Time             CPU   Iterations Use
---------------------------------------------------------------------------
BM_BinarySearch/32768               2139 ns         2074 ns       257496 ite          ond=15.7973G/s
BM_BinarySearch/65536               2430 ns         2317 ns       328503 it           ond=28.2893G/s
BM_BinarySearch/131072              2394 ns         2255 ns       244561 i            ond=58.1245G/s
BM_BinarySearch/262144              2559 ns         2388 ns       309382 items_per_second=109.798G/s
BM_BinarySearch/524288              2455 ns         2293 ns       303165 items_per_second=228.659G/s
BM_BinarySearch/1048576             7964 ns         6761 ns        98694 items_per_second=155.092G/s
BM_BinarySearch/2097152          1088699 ns      1086450 ns          656 items_per_second=1.93028G/s
BM_BinarySearch/4194304          2623366 ns      2611351 ns          322 items_per_second=1.60618G/s
BM_BinarySearch/8388608          4390794 ns      4380928 ns          166 items_per_second=1.9148G/s
BM_BinarySearch_BigO            54979.95 lgN    54806.52 lgN
```

?

Summary

- Google Benchmark
  - **Sweeping** parameters
  - **Pause/resume** the timer
  - Runtime **complexity**
- Benchmarking:
  - Careful understanding and analysis of your **entire system**
- For critical programs
  - **Always benchmark**!

GLog
& Abseil in C++

By Ari Saif

Prerequisite: Installing Bazel

This repo uses `Bazel` for building C++ files. You can install Bazel using this link.

Cloning this repo

```
git clone https://github.com/ourarash/cpp-template.git
```

Examples:

Hello World Example:

You can run this using `bazel`:

```
bazel run src/main:main
```

https://github.com/ourarash/cpp-template

Abseil

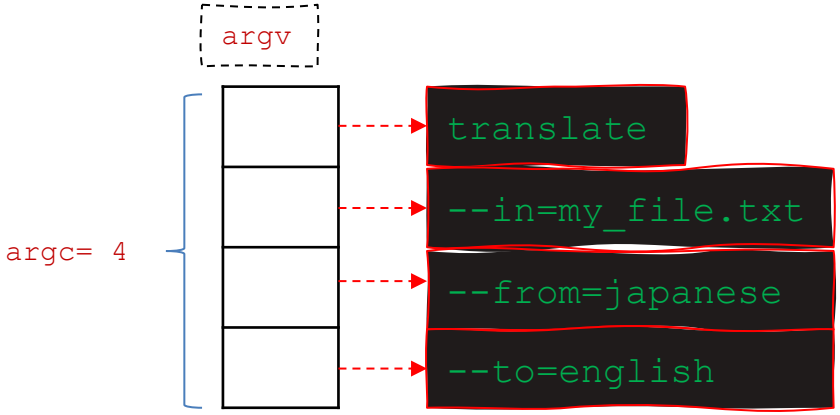
Abseil

- An open-source library augmenting the C++ standard library
  - **Flags**: Parsing **flag** values passed on the **command-line** to binaries
  - **Time**: Holding **time values**, both in terms of **absolute** time and **civil** time

Why Flags?

```
> translate --in=my_file.txt --from=japanese  --to=english
```

```
int main(int argc, char *argv[])
```



Abseil Flags

Define a Flag

```
ABSL_FLAG(type, name, default, help-text)
```

```
ABSL_FLAG(bool, verbose, false, ”Enable verbose mode");

ABSL_FLAG(std::string, message, "Hello world!", "Message to print");

ABSL_FLAG(std::vector<std::string>, names,
  std::vector<std::string>({"jack", "jim", "jamal"}),
  "comma-separated list of names the program accepts");
```

Command Line Arguments (Flags) Using Abseil

```
// Define the flag
ABSL_FLAG(bool, verbose, false, ”Enable verbose mode");

// Get the flag value
absl::GetFlag(FLAGS_verbose);
```

Command Line Arguments (Flags) Using Abseil

```
#include "absl/flags/flag.h”

ABSL_FLAG(bool, verbose, false, ”Enable verbose mode");
ABSL_FLAG(std::string, message, "Hello world!", "Message to print");

int main(int argc, char *argv[]) {
  absl::ParseCommandLine(argc, argv);

  if (absl::GetFlag(FLAGS_verbose)) {
   std::cout << "Verbose ” << ": ";
  }

  std::cout << absl::GetFlag(FLAGS_message) << std::endl;
  return 0;
}
```

Call this in the beginning

```
> main_flags_absl --verbose=true --message="hello world"
```

Supported Types

```
bool
int16_t
uint16_t
int32_t
uint32_t
int64_t
uint64_t
float
double
std::string
std::vector<std::string>
```

Special Usage Flags

```
--help            show help on important flags for this binary
--helpfull        shows all flags from all files, sorted by file and then
                  by name; shows the flagname, its default value, and its
                  help string
--helpshort       shows only flags for the file with the same name as the
                  executable (usually the one containing main())
--helpon=FILE     shows only flags defined in FILE.*
--helpmatch=S     shows only flags defined in *S*.*
--helppackage     shows flags defined in files in same directory as main()
--version         prints version info for the executable
```

https://abseil.io/docs/cpp/guides/flags

Bonus: Logging

Without a Logger

```
int main(int argc, char* argv[]) {
  std::vector<int> my_vector = {1, 2, 3, 4};
  std::map<int, int> my_map = {{1, 2}, {2, 3}};

  std::cout << "INFO: "
   << "This is an info message" << std::endl;
  std::cout << "WARNING: "
  << "This is a warning message" << std::endl;
  std::cout << "ERROR: "
  << "This is an error message" << std::endl;

  std::cout << "Printing my_vector: " << my_vector << std::endl;
  std::cout << "Printing my_map: " << my_map << std::endl;
  if (g_cond == true) {
    std::cout << "g_cond is true!" << std::endl;
  }
  return 0;
}
```

Logging using GLOG

```
#include <glog/logging.h>
#include <glog/stl_logging.h>

void MyFunction() {
std::vector<int> my_vector = {1, 2, 3, 4};
  std::map<int, int> my_map = {{1, 2}, {2, 3}};

  LOG(INFO) << "Printing my_vector: "
    << "{" << my_vector << "}";

  LOG(INFO) << "Printing a my_map " << my_map;

  LOG(WARNING) << "This is a warning message";
  LOG(INFO) << "Hello, world again!";
  LOG(ERROR) << "This is an error message";
  LOG_IF(INFO, g_cond == true) << "g_cond is true!";
  CHECK(5 == 4) << "Check failed!";
}
```

Debugging


DEBUGGING
I DON'T KNOW WHERE YOU ARE,
BUT I WILL FIND YOU AND I WILL FIX YOU!
imgflip.com

Google Logs

LOG Macros	Check Macros	Flags (Use them with FLAG_*)
LOG_IF(INFO, condition)	CHECK(condition)	alsologtostderr
LOG_EVERY_N(INFO, 10)	CHECK_NE(a,b)	logtostderr
LOG_IF_EVERY_N(INFO, condition, 10)	CHECK_EQ(a, b);	minloglevel
LOG_FIRST_N(INFO, 20)	CHECK_NOTNULL(pointer)	stderrthreshold
		log_dir

https://github.com/google/glog/