

C++ Resources for All

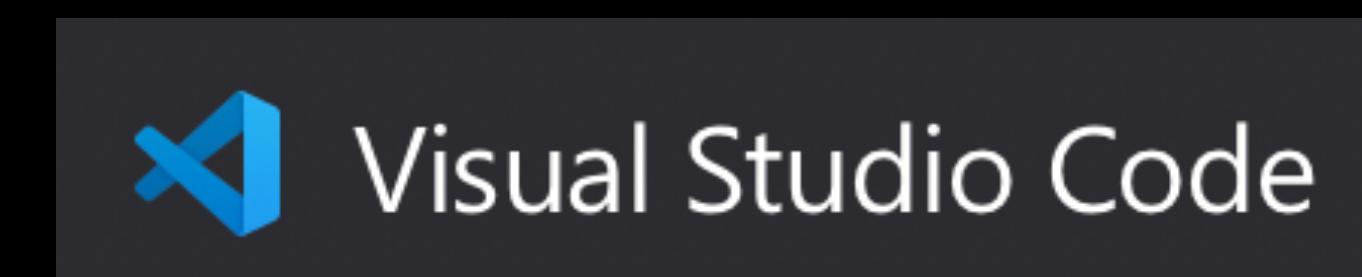
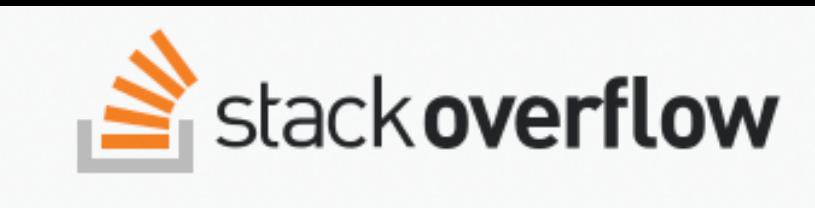
Richard Powell (rmpowell77@me.com), 6/15/2022



Algorithms +
Data
Structures =
Programs



cppreference.com







CMake Tutorial

<https://cmake.org/cmake-tutorial/>

Professional CMake: A Practical Guide

<https://crascit.com/professional-cmake/>



CGold

<https://cgold.readthedocs.io/en/latest/index.html>

Effective Modern CMake

<https://gist.github.com/mbinna/c61dbb39bca0e4fb7d1f73b0d66a4fd1>



Make sure I have cmake up to date



```
[~] brew install cmake
Warning: Treating cmake as a formula. For the cask, use homebrew/cask/cmake
cmake 3.21.0 is already installed but outdated (so it will be upgraded).
==> Downloading https://ghcr.io/v2/homebrew/core/cmake/manifests/3.23.1
#####
# 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/cmake/blobs/sha256:9fe99b75d9e37e3acd653f93d429af93f94c5460a50284e6c56cf99fb4f8
==> Downloading from https://pkg-containers.githubusercontent.com/ghcr1/blobs/sha256:9fe99b75d9e37e3acd653f93d429af93f94c5460a50
#####
# 100.0%
==> Upgrading cmake
3.21.0 -> 3.23.1

==> Pouring cmake--3.23.1.arm64_monterey.bottle.tar.gz
==> Caveats
To install the CMake documentation, run:
  brew install cmake-docs

Emacs Lisp files have been installed to:
  /opt/homebrew/share/emacs/site-lisp/cmake
==> Summary
🍺 /opt/homebrew/Cellar/cmake/3.23.1: 3,043 files, 42.1MB
==> `brew cleanup` has not been run in the last 30 days, running now...
Disable this behaviour by setting HOMEBREW_NO_INSTALL_CLEANUP.
```

Create the basic project files

```
test_fft_benchmark — richardpowell@Richards-MacBook-Air-2 — ..fft_b...
[→ test_fft_benchmark cat CMakeLists.txt ]]
cmake_minimum_required(VERSION 3.23)

set(CMAKE_CXX_STANDARD 20)

project(
    test_fft_benchmark
    VERSION 1.0
    LANGUAGES CXX)

add_executable(test_fft_benchmark test_fft_benchmark.cpp)

[→ test_fft_benchmark cat test_fft_benchmark.cpp ]]
#include <iostream>

int main() {
    std::cout<<"hello world\n";
}
→ test_fft_benchmark
```

Visual Studio Code

The screenshot shows the official Visual Studio Code website (`code.visualstudio.com`) displayed in a web browser. The page features a dark-themed header with the Visual Studio Code logo, navigation links (Docs, Updates, Blog, API, Extensions, FAQ, Learn), a search bar, and a prominent blue "Download" button. A banner at the top announces "Version 1.67 is now available! Read about the new features and fixes from April." Below this, a large headline reads "Code editing. Redefined." followed by the text "Free. Built on open source. Runs everywhere." A "Download Mac Universal" button is visible, along with links for "Web, Insiders edition, or other platforms". At the bottom, there's a note about agreeing to the license and privacy statement. The main content area displays a screenshot of the VS Code interface, showing multiple tabs (blog-post.js, index.js, utils.js) with code snippets, a sidebar with extension marketplace items like Python, GitLens, and ESLint, and a terminal window showing build logs.

code.visualstudio.com

Visual Studio Code Docs Updates Blog API Extensions FAQ Learn

Search Docs Download

Version 1.67 is now available! Read about the new features and fixes from April.

Code editing. Redefined.

Free. Built on open source. Runs everywhere.

Download Mac Universal

Stable Build

Web, Insiders edition, or other platforms

By using VS Code, you agree to its license and privacy statement.

EXTENSIONS: MARKETPLACE

- @sort:installs
- Python 2019.6.24221 54.9M ★4.5 Microsoft Install
- GitLens — Git supercharged 9.8.5 23.1M ★5 Eric Amodio Install
- C/C++ 0.24.0 23M ★3.5 Microsoft Install
- ESLint 1.9.0 21.9M ★4.5 Integrates ESLint JavaScript into V... Dirk Baeumer Install
- Debugger for C... 4.11.6 20.6M ★4.5 Debug your JavaScript code in the ... Microsoft Install
- Language Support 0.47.0 18.6M ★4.5 Java Linting, Intellisense, formattin... Red Hat Install
- vscode-icons 8.8.0 17.2M ★5 Icons for Visual Studio Code VSCode Icons Team Install
- Vetur 0.21.1 17M ★4.5 Vue tooling for VS Code Pine Wu Install

blog-post.js — gatsby-graphql-app

```
src > components > JS blog-post.js > <function> > blogPost
1 import { graphql } from 'gatsby'
2 import React from "react"
3 import Image from "gatsby-image"
4
5 export default ({ data }) => {
6   const blogPost = data.cms.blogPost
7   return (
8     <div>
9       {blogPost.debug}
10      blogPost.debugger
11      blogPost.decodeURI
12      <Image decodeURIComponent=>
13        <div>
14          <h1>{blogPost.defaultStatus}</h1>
15          <div>Post</div>
16          <div>dangerous</div>
17          </div>
18    )
19  }
20
21 export const query = graphql`
```

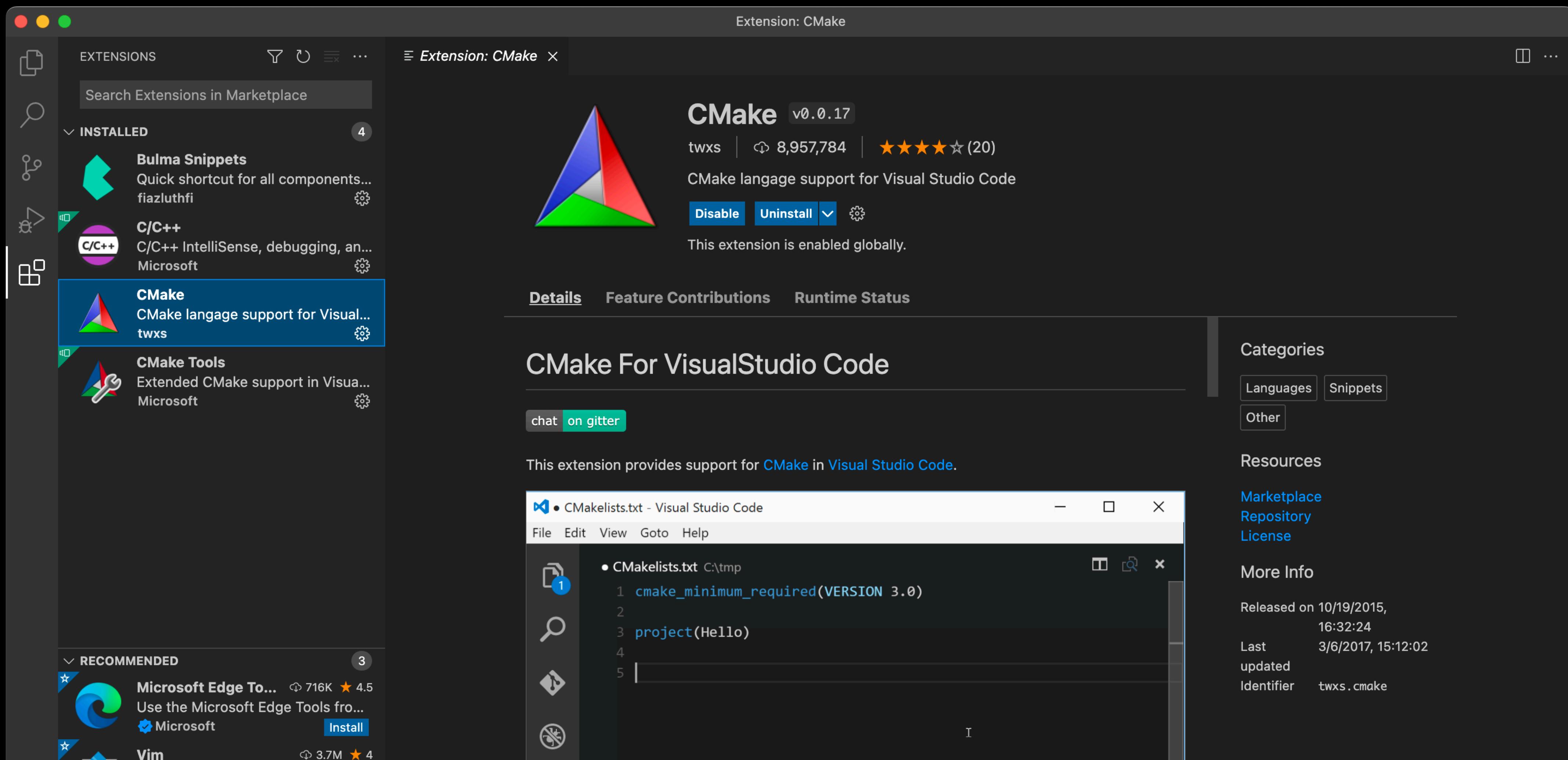
PROBLEMS TERMINAL ...

```
info i [wdm]: Compiling...
DONE Compiled successfully in 26ms
3:57:58 PM
```

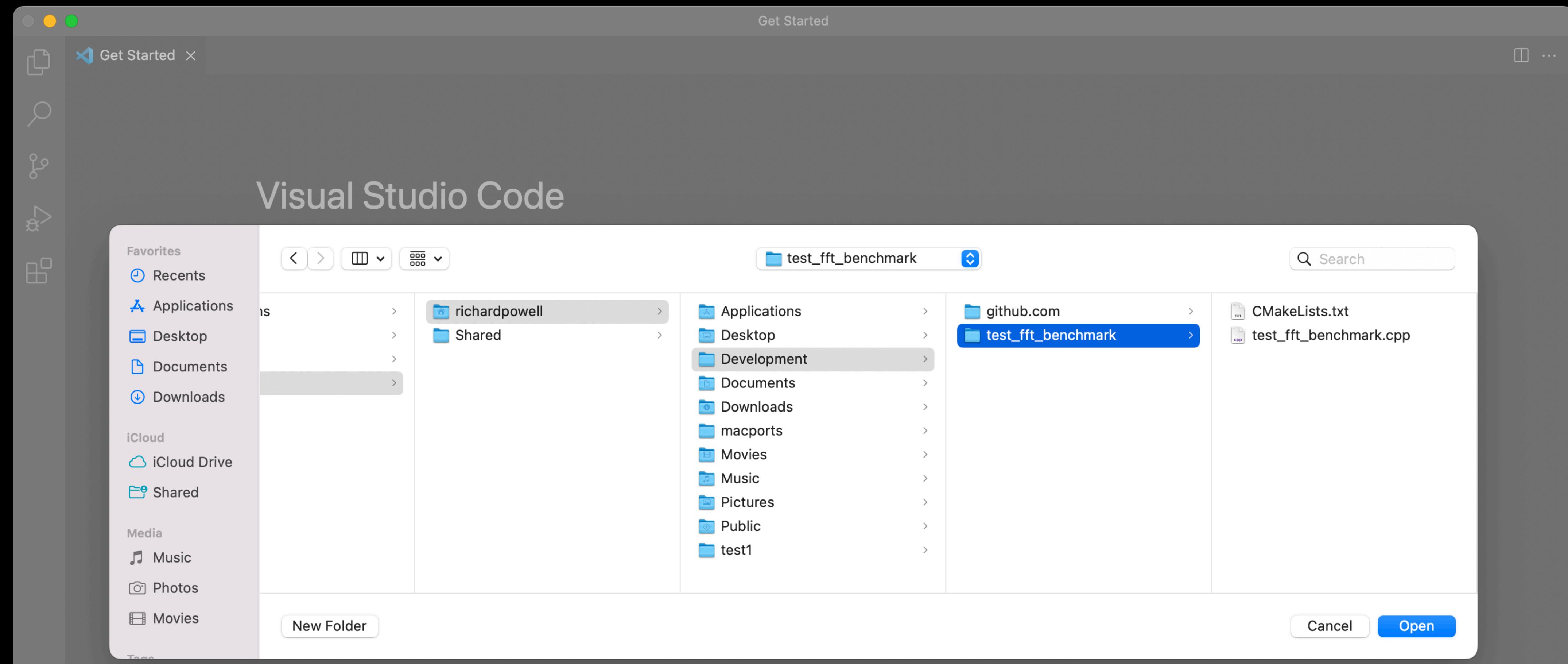
```
info i [wdm]:
info i [wdm]: Compiled successfully.
```

Ln 6, Col 21 Spaces: 2 UTF-8 LF JavaScript

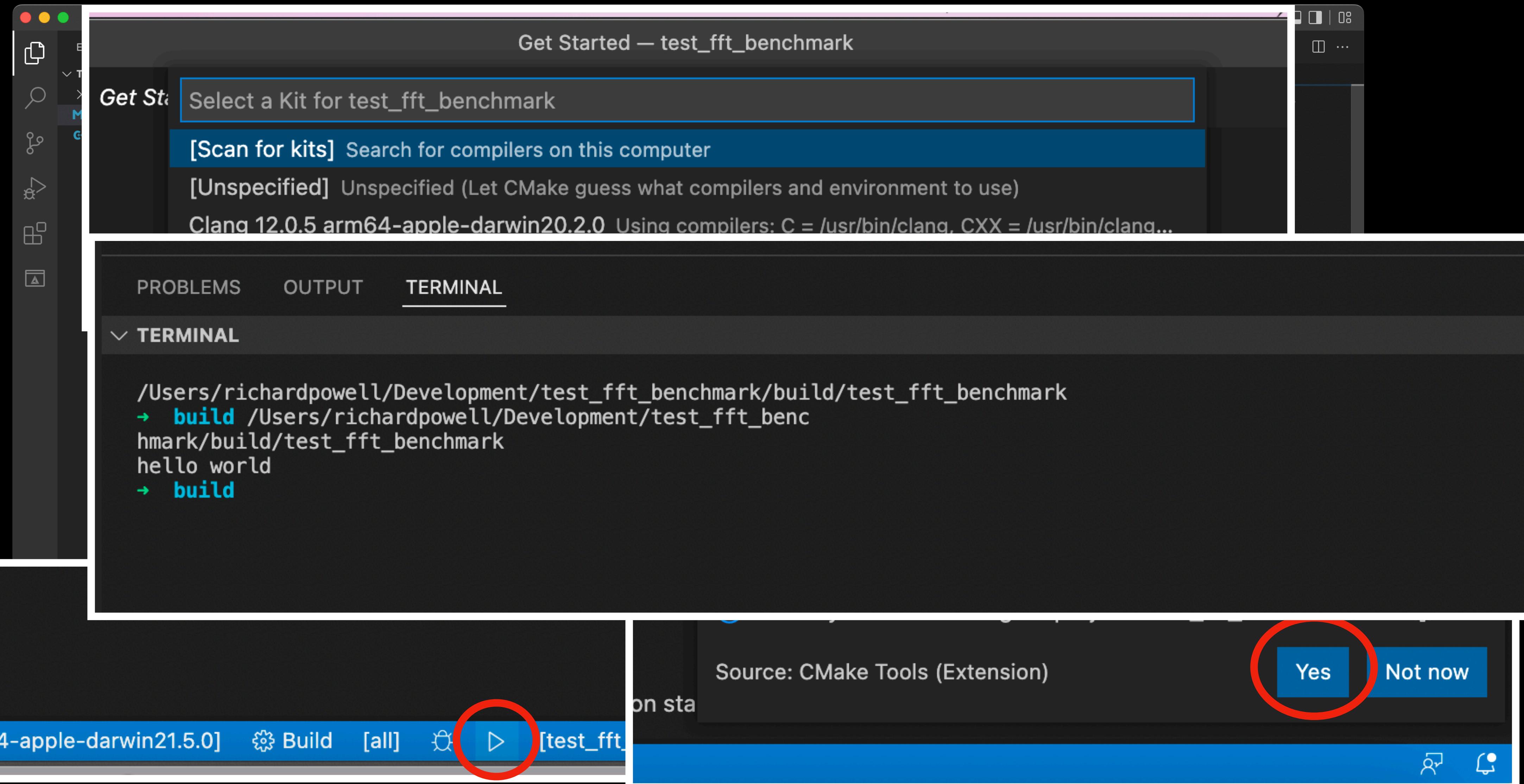
Power up VSC with Extensions



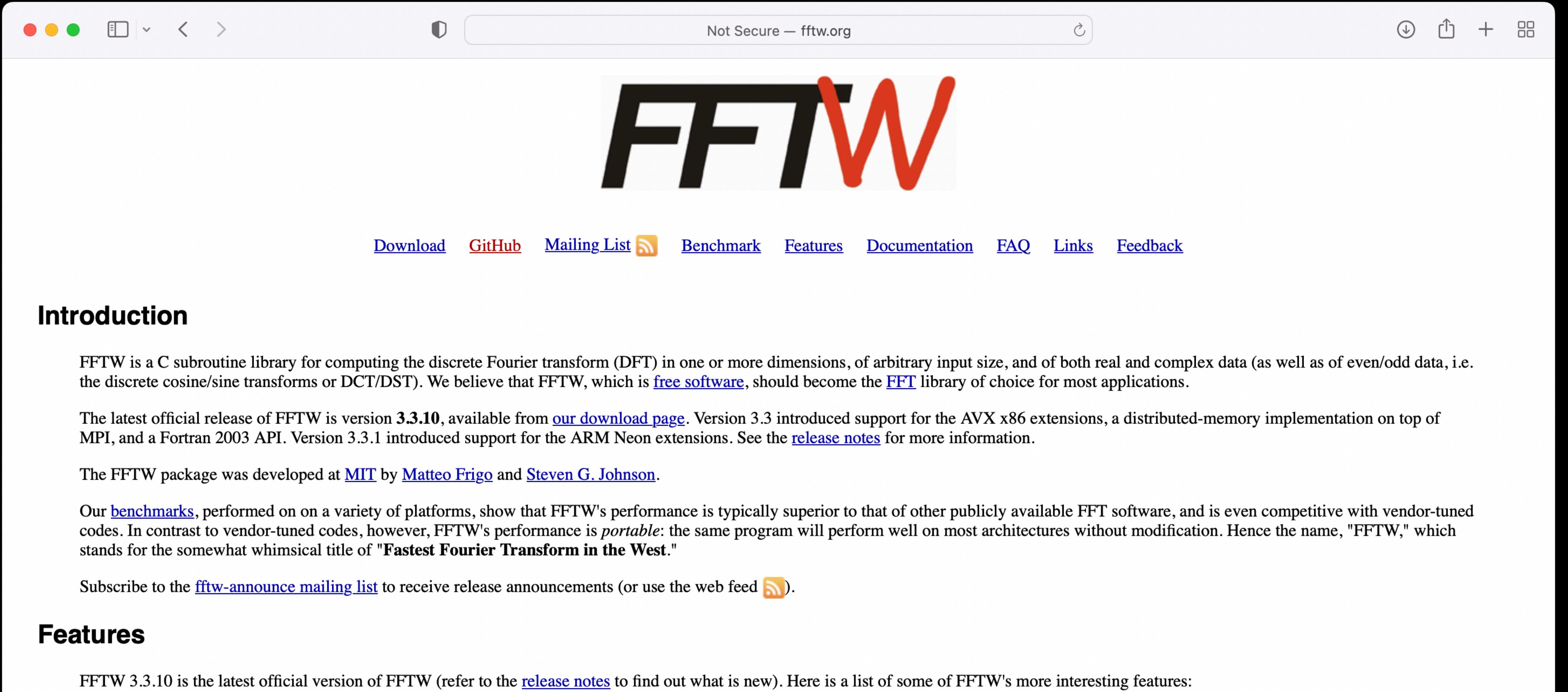
Open the directory



Configuring and build



FFTW (the Fastest Fourier Transform in the west!)

A screenshot of a web browser window showing the FFTW website. The title bar reads "Not Secure — fftw.org". The main content features the large FFTW logo (black F and T, red W) centered above a navigation menu with links: Download, GitHub, Mailing List (RSS), Benchmark, Features, Documentation, FAQ, Links, and Feedback. Below the menu is a section titled "Introduction".

Introduction

FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, i.e. the discrete cosine/sine transforms or DCT/DST). We believe that FFTW, which is [free software](#), should become the [FFT](#) library of choice for most applications.

The latest official release of FFTW is version **3.3.10**, available from [our download page](#). Version 3.3 introduced support for the AVX x86 extensions, a distributed-memory implementation on top of MPI, and a Fortran 2003 API. Version 3.3.1 introduced support for the ARM Neon extensions. See the [release notes](#) for more information.

The FFTW package was developed at [MIT](#) by [Matteo Frigo](#) and [Steven G. Johnson](#).

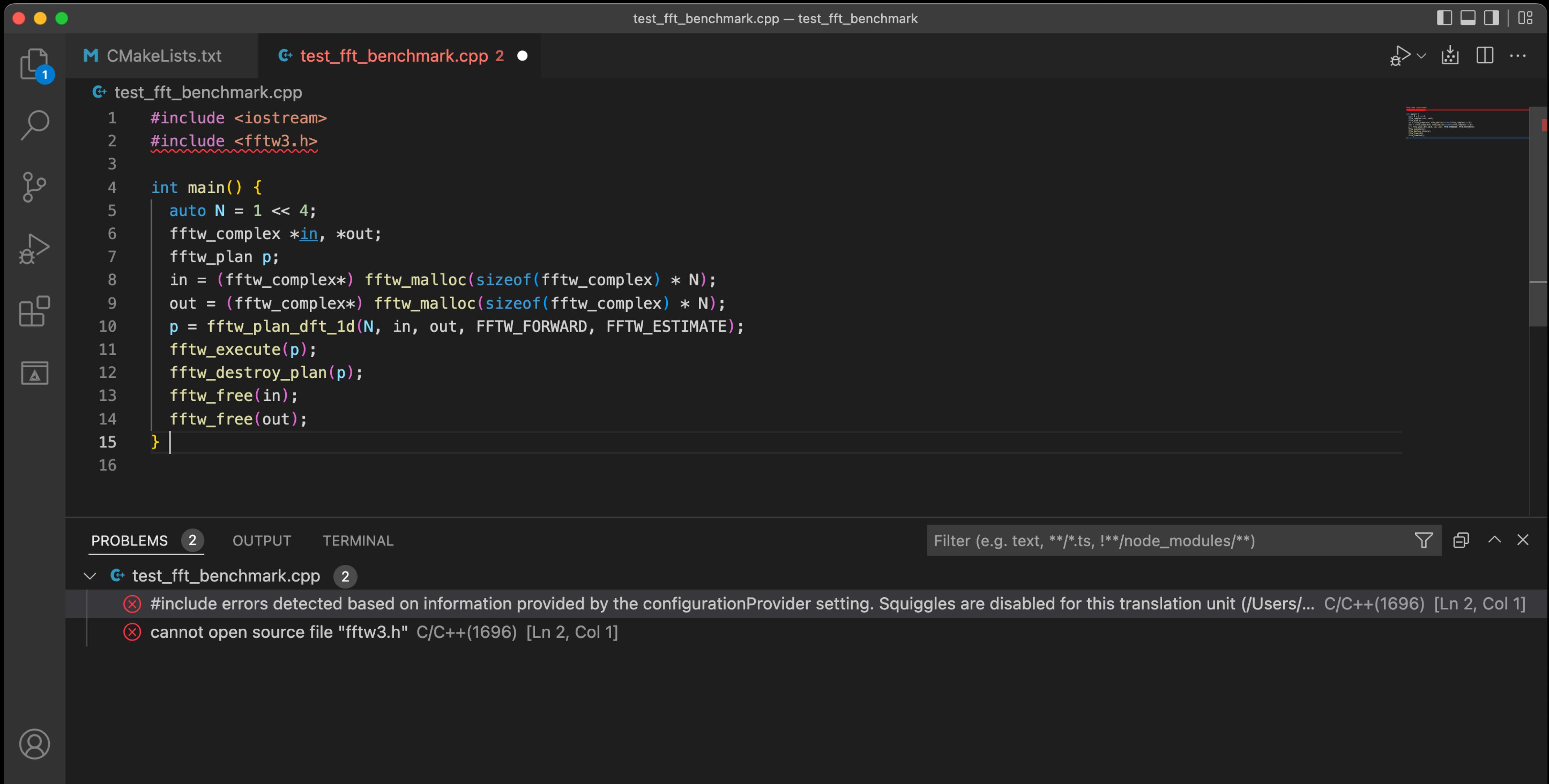
Our [benchmarks](#), performed on a variety of platforms, show that FFTW's performance is typically superior to that of other publicly available FFT software, and is even competitive with vendor-tuned codes. In contrast to vendor-tuned codes, however, FFTW's performance is *portable*: the same program will perform well on most architectures without modification. Hence the name, "FFTW," which stands for the somewhat whimsical title of "**F**astest **F**ourier **T**ransform **i**n the **W**est."

Subscribe to the [fftw-announce mailing list](#) to receive release announcements (or use the web feed).

Features

FFTW 3.3.10 is the latest official version of FFTW (refer to the [release notes](#) to find out what is new). Here is a list of some of FFTW's more interesting features:

Let's use it!



The screenshot shows a dark-themed code editor interface. At the top, there are three tabs: "CMakeLists.txt" (selected), "test_fft_benchmark.cpp" (with a red squiggle under "fftw3.h"), and "test_fft_benchmark.cpp 2". The main editor area contains the following code:

```
test_fft_benchmark.cpp — test_fft_benchmark
1 #include <iostream>
2 #include <fftw3.h>
3
4 int main() {
5     auto N = 1 << 4;
6     fftw_complex *in, *out;
7     fftw_plan p;
8     in = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
9     out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
10    p = fftw_plan_dft_1d(N, in, out, FFTW_FORWARD, FFTW_ESTIMATE);
11    fftw_execute(p);
12    fftw_destroy_plan(p);
13    fftw_free(in);
14    fftw_free(out);
15 }
16
```

Below the editor is a "PROBLEMS" tab with a count of 2. The list shows two errors:

- test_fft_benchmark.cpp:2: #include errors detected based on information provided by the configurationProvider setting. Squiggles are disabled for this translation unit (/Users/... C/C++(1696) [Ln 2, Col 1])
- test_fft_benchmark.cpp:2: cannot open source file "fftw3.h" C/C++(1696) [Ln 2, Col 1]

Stack Overflow to the rescue

A screenshot of a Google search results page. The search bar at the top contains the query "cmake and fftw". Below the search bar, there are navigation links for "All", "Videos", "News", "Shopping", "Images", and "More". A "Tools" link is also present. The search results indicate "About 73,400 results (0.35 seconds)". The first result is highlighted with a black border and contains the following text:

We delegate this to `pkg-config`:

```
3
find_package(PkgConfig REQUIRED)
pkg_search_module(FFTW REQUIRED fftw3 IMPORTED_TARGET)
include_directories(PkgConfig::FFTW)
link_libraries    (PkgConfig::FFTW)
```

This works with cmake 3.11 (at least, it may work with earlier versions too).

At the bottom of the page, there is a link to the GitHub issue: <https://github.com/xtensor-stack/xtensor-fftw/issues/52>.

CMakeLists.txt — test_fft_benchmark

CMakeLists.txt

```
1 cmake_minimum_required(VERSION 3.23)
2
3 set(CMAKE_CXX_STANDARD 20)

4
5 find_package(PkgConfig REQUIRED)
6 pkg_search_module(FFTW REQUIRED fftw3 IMPORTED_TARGET)
7 include_directories(PkgConfig::FFTW)
8 link_libraries(PkgConfig::FFTW)

9
10 project(
11     test_fft_benchmark
12     VERSION 1.0
13     LANGUAGES CXX)
14
15 add_executable(test_fft_benchmark test_fft_benchmark.cpp)
16
17
```

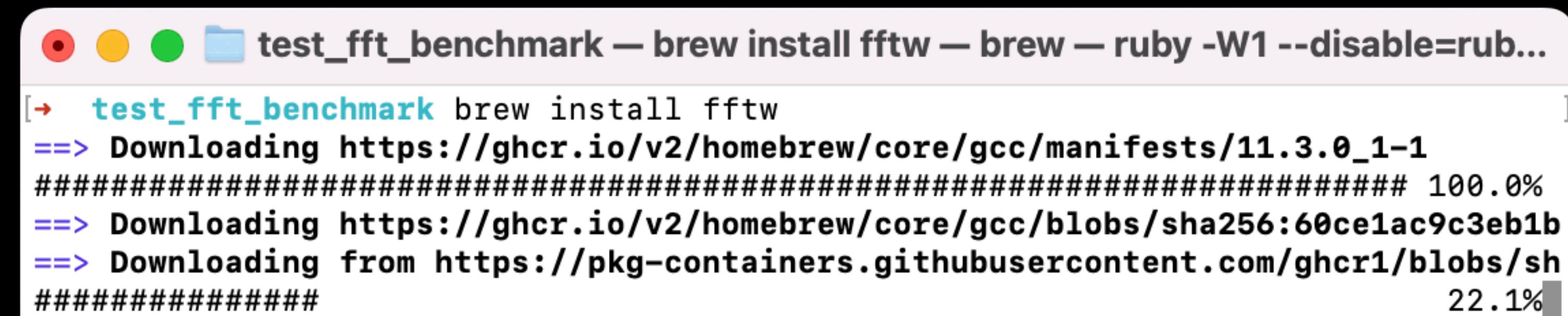
PROBLEMS 3 **OUTPUT** **TERMINAL**

[cmake] -- Found PkgConfig: /opt/homebrew/bin/pkg-config (found version "0.29.2")
[cmake] -- Checking for one of the modules 'fftw3'
[cmake] CMake Error at /opt/homebrew/Cellar/cmake/3.23.1/share/cmake/Modules/FindPkgConfig.cmake:890 (message):
[cmake] None of the required 'fftw3' found
[cmake] Call Stack (most recent call first):
[cmake] CMakeLists.txt:6 (pkg_search_module)
[cmake]
[cmake]
[cmake] -- Configuring incomplete, errors occurred!
[cmake] See also "/Users/richardpowell/Development/test_fft_benchmark/build/CMakeFiles/CMakeOutput.log".

Ln 10, Col 1 Spaces: 2 UTF-8 LF CMake ⚙️ 🔔

Oops, need to make sure I've got it installed...

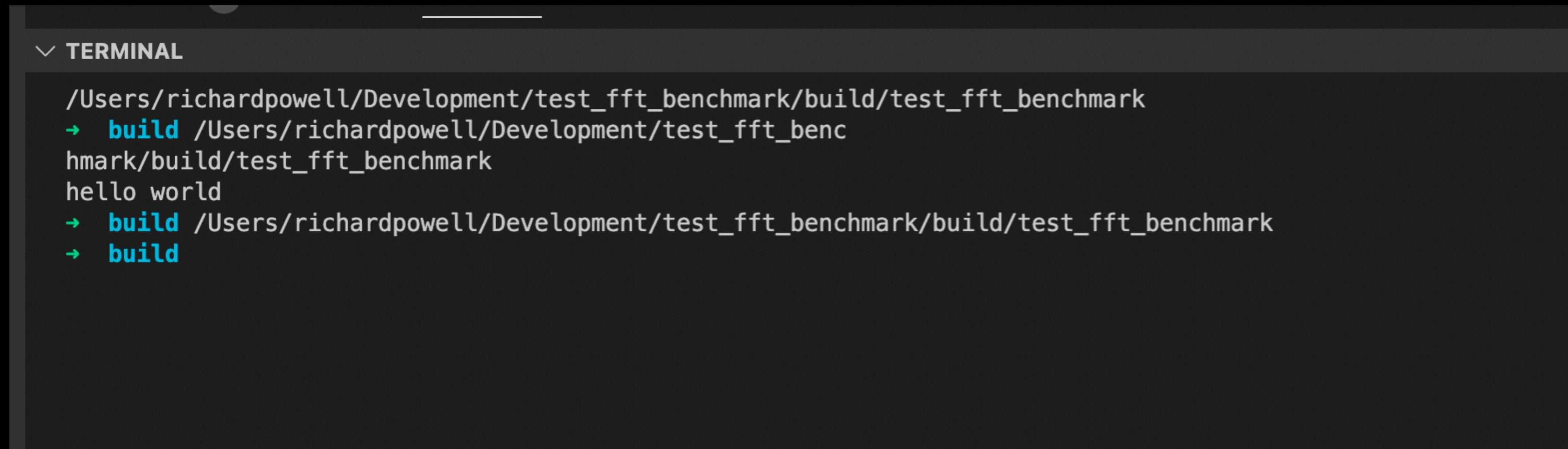
Oops, need to make sure I've got it installed...



A screenshot of a macOS terminal window titled "test_fft_benchmark — brew install fftw — brew — ruby -W1 --disable=rub...". The window shows the command "brew install fftw" being run. The output indicates that Homebrew is downloading the package from multiple sources. The progress bar at the bottom right shows the download is at 22.1% completion.

```
[→ test_fft_benchmark brew install fftw
==> Downloading https://ghcr.io/v2/homebrew/core/gcc/manifests/11.3.0_1-1
#####
# 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/gcc/blobs/sha256:60ce1ac9c3eb1b
==> Downloading from https://pkg-containers.githubusercontent.com/ghcr1/blobs/sh
#####
# 22.1%
```

Well, that's not interesting...



A screenshot of a terminal window titled "TERMINAL". The terminal output shows a build process for a project named "test_fft_benchmark". The path is "/Users/richardpowell/Development/test_fft_benchmark/build/test_fft_benchmark". The command "build" is run twice, resulting in the output "hello world".

```
/Users/richardpowell/Development/test_fft_benchmark/build/test_fft_benchmark
→ build /Users/richardpowell/Development/test_fft_benc
hmark/build/test_fft_benchmark
hello world
→ build /Users/richardpowell/Development/test_fft_benchmark/build/test_fft_benchmark
→ build
```

Cosine!

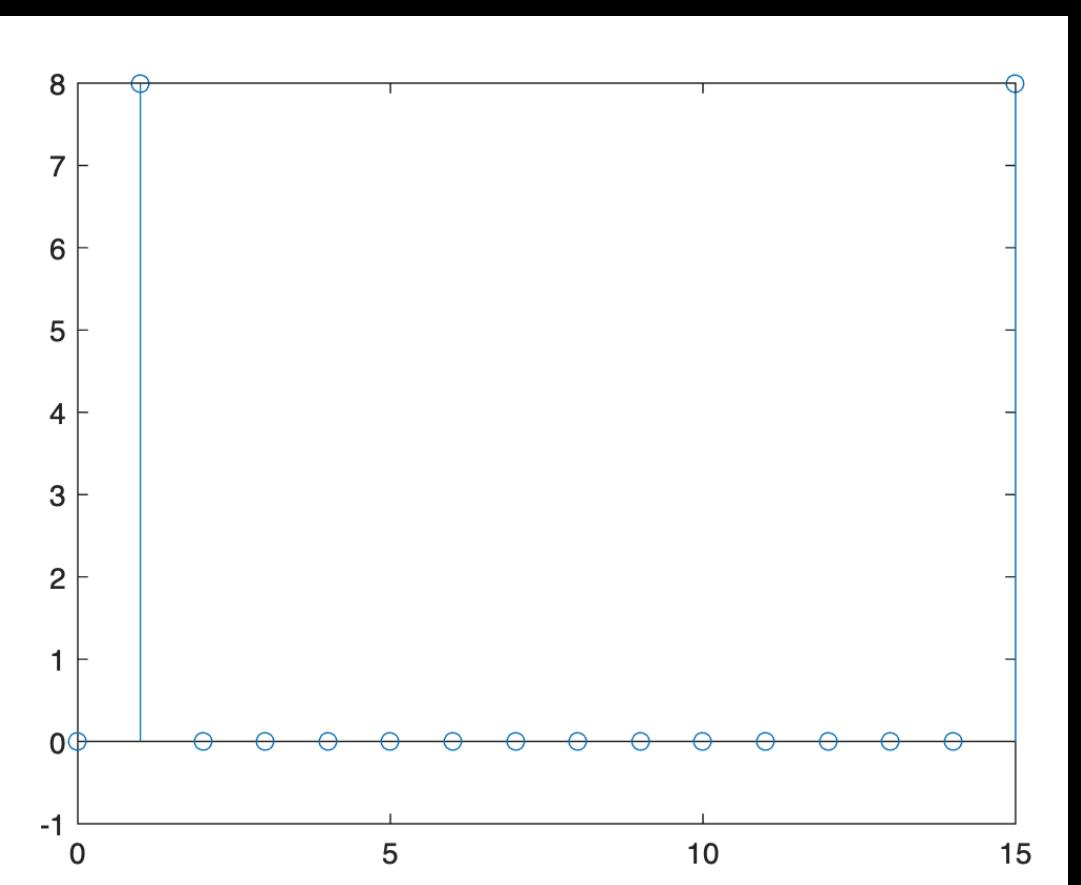
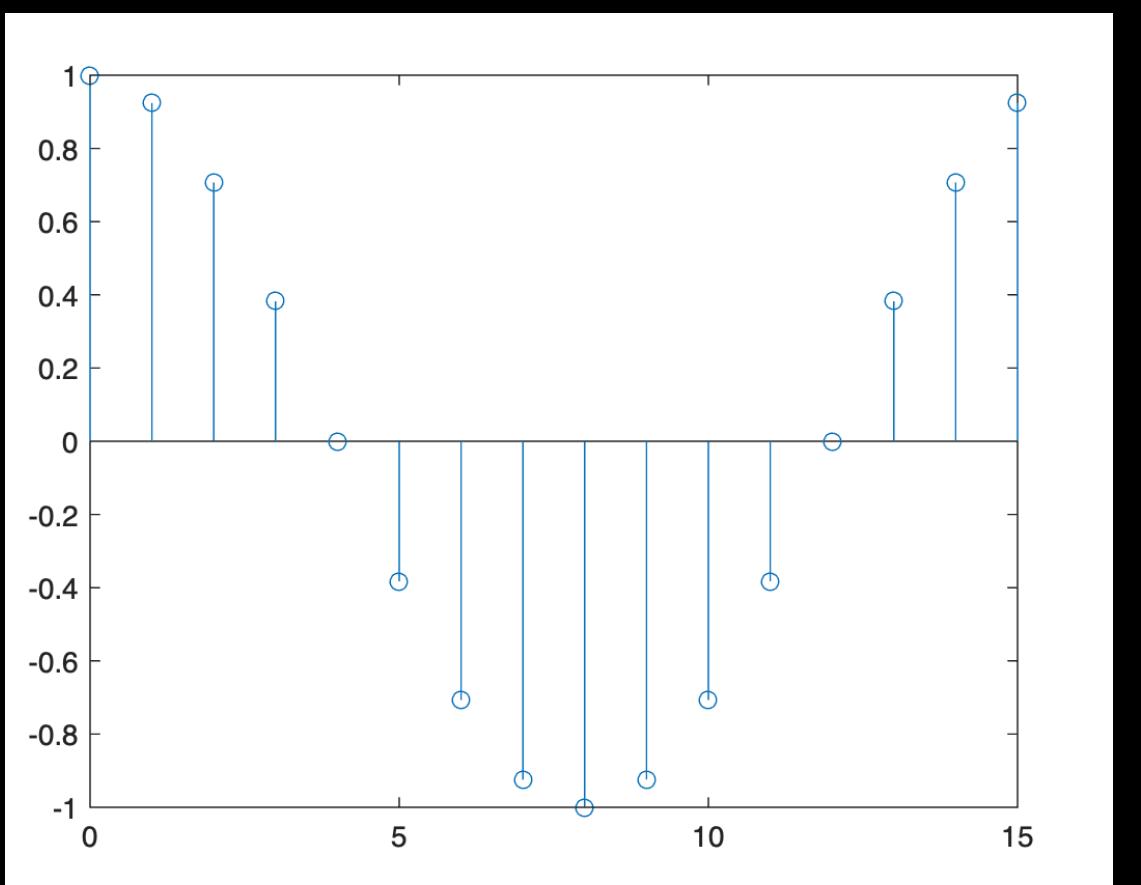
```
⌚ test_fft_benchmark.cpp > ⚡ main()
1  #include <iostream>
2  #include <numbers>
3  #include <algorithm>
4  #include <math.h>
5  #include <fftw3.h>
6
7  int main() {
8      auto N = 1 << 4;
9      fftw_complex *in, *out;
10     fftw_plan p;
11     in = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
12
13     for (auto i = 0; i < N; ++i) {
14         in[i][0] = cos(i * 2 * std::numbers::pi/N);
15     }
16
17     std::cout<<"Input: ";
18     for (auto i = 0; i < N; ++i) {
19         std::cout<<"("<<in[i][0]<<","<<in[i][1]<<")";
20     }
21     std::cout<<"\n";
22
23     out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
24     p = fftw_plan_dft_1d(N, in, out, FFTW_FORWARD, FFTW_ESTIMATE);
25
26     fftw_execute(p);
27
28     std::cout<<"\nOutput: ";
29     for (auto i = 0; i < N; ++i) {
30         std::cout<<"("<<out[i][0]<<","<<out[i][1]<<")";
31     }
32     std::cout<<"\n";
33
34     fftw_destroy_plan(p);
35     fftw_free(in);
36     fftw_free(out);
37 }
```

PROBLEMS OUTPUT TERMINAL

TERMINAL

```
/Users/richardpowell/Development/test_fft_benchmark/build/test_fft_benchmark
→ build /Users/richardpowell/Development/test_fft_benchmark/build/test_fft_benchmark
build/test_fft_benchmark
hello world
→ build /Users/richardpowell/Development/test_fft_benchmark/build/test_fft_benchmark
→ build /Users/richardpowell/Development/test_fft_benchmark/build/test_fft_benchmark
→ build /Users/richardpowell/Development/test_fft_benchmark/build/test_fft_benchmark
Input: (1,0)(0.92388,0)(0.707107,0)(0.382683,0)(6.12323e-17,0)(-0.382683,0)(-0.707107,0)(-0.92388,0)(-1,0)(-0.92388,0)(-0.707107,0)(-0.382683,0)(-1.83697e-16,0)(0.3
(0.707107,0)(0.92388,0)

Output: (-8.99621e-16,0)(8,-1.08026e-15)(4.39601e-17,4.71028e-16)(-9.05066e-16,-6.90891e-17)(9.95799e-17,-8.88178e-16)(3.66477e-16,6.90891e-17)(2.00969e-16,4.71028e
.59171e-17)(2.10602e-16,0)(0,-3.76406e-17)(2.00969e-16,-4.71028e-16)(3.58201e-16,-6.90891e-17)(9.95799e-17,8.88178e-16)(-9.13342e-16,6.90891e-17)(4.39601e-17,-4.71
8,1.07198e-15)
→ build
```



Hey, this is a C++ talk!

```
struct FFT {
    using complex = std::complex<double>;
    FFT(int log2n)
        : log2n(log2n)
        , N(1 << log2n)
        , in(static_cast<fftw_complex *>(fftw_malloc(sizeof(fftw_complex) * N)))
        , out(static_cast<fftw_complex *>(fftw_malloc(sizeof(fftw_complex) * N)))
        , p(fftw_plan_dft_1d(N, in, out, FFTW_FORWARD, FFTW_ESTIMATE)) {
    }

    ~FFT() {
        fftw_destroy_plan(p);
        fftw_free(in);
        fftw_free(out);
    }

    void fill_input(std::vector<complex> const& input) {
        std::copy(input.begin(), input.end(), reinterpret_cast<complex*>(in));
    }

    std::vector<complex> fetch_output() const {
        auto output = reinterpret_cast<complex*>(out);
        return { output, output + N };
    }

    void execute() {
        fftw_execute(p);
    }

    int log2n{};
    int N{};
    fftw_complex *in{};
    fftw_complex *out{};
    fftw_plan p{};
};
```

```
int main() {
    auto fft = FFT(4);

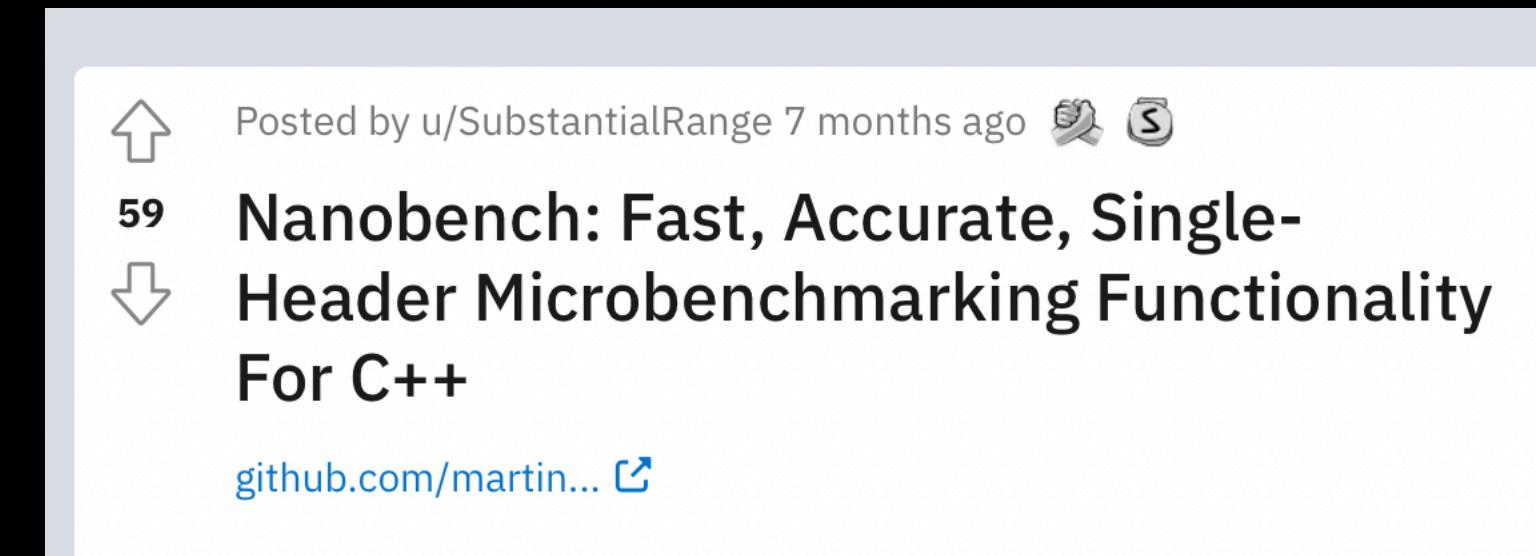
    auto in = std::vector<FFT::complex>(fft.N);
    for (auto i = 0; i < fft.N; ++i) {
        in[i] = cos(i * 2 * std::numbers::pi / fft.N);
    }

    std::cout << "Input: ";
    std::copy(in.begin(), in.end(), std::ostream_iterator<FFT::complex>(std::cout, ", "));
    std::cout << "\n";

    fft.fill_input(in);
    fft.execute();
    auto out = fft.fetch_output();

    std::cout << "\nOutput: ";
    std::copy(out.begin(), out.end(), std::ostream_iterator<FFT::complex>(std::cout, ", "));
    std::cout << "\n";
}
```

Now some benchmarks



A screenshot of a web browser showing the nanobench website and its GitHub page side-by-side. The left side shows the nanobench homepage with a blue header, a wooden chair icon, and the text "v4.3.7". It includes a search bar and a "GETTING STARTED" section with links for Installation, Usage, Examples, Comparing Results, and Asymptotic Complexity. The right side shows the GitHub repository page for nanobench, featuring the repository name, a "Edit on GitHub" button, and a summary card with status badges for release (v4.3.7), license (MIT), build (passing), and chat (on gitter). Both pages feature a small wooden chair icon at the bottom.

Integrate with nanobench

M CMakeLists.txt

```
1 cmake_minimum_required(VERSION 3.23)
2
3 set(CMAKE_CXX_STANDARD 20)
4
5 find_package(PkgConfig REQUIRED)
6 pkg_search_module(FFTW REQUIRED fftw3 IMPORTED_TARGET)
7 include_directories(PkgConfig::FFTW)
8 link_libraries(PkgConfig::FFTW)
9
10 project(
11     test_fft_benchmark
12     VERSION 1.0
13     LANGUAGES CXX)
14
15 add_executable(test_fft_benchmark test_fft_benchmark.cpp)
16
17
```

```
1 cmake_minimum_required(VERSION 3.23)
2
3 set(CMAKE_CXX_STANDARD 20)
4
5 find_package(PkgConfig REQUIRED)
6 pkg_search_module(FFTW REQUIRED fftw3 IMPORTED_TARGET)
7 include_directories(PkgConfig::FFTW)
8 link_libraries(PkgConfig::FFTW)
9
10 include(FetchContent)
11
12 FetchContent_Declare(
13     nanobench
14     GIT_REPOSITORY https://github.com/martinus/nanobench.git
15     GIT_TAG v4.1.0
16     GIT_SHALLOW TRUE)
17
18 FetchContent_MakeAvailable(nanobench)
19
20 project(
21     test_fft_benchmark
22     VERSION 1.0
23     LANGUAGES CXX)
24
25 add_executable(test_fft_benchmark test_fft_benchmark.cpp)
26 target_link_libraries(test_fft_benchmark PRIVATE nanobench)
```

Integrate with nanobench

```
int main() {
> { ...

    for (auto i : { 7, 8, 9, 10, 11, 12 }) {
        auto fft = FFT(i);
        auto in = std::vector<FFT::complex>(fft.N);
        for (auto i = 0; i < fft.N; ++i) {
            in[i] = cos(i * 2 * std::numbers::pi / fft.N);
        }
        ankerl::nanobench::Bench().run(std::string("fftw ") + std::to_string(fft.N) , [&] {
            fft.execute();
        });
    }
}
```

ns/op	op/s	err%	total	benchmark
245.86	4,067,377.93	0.1%	0.00	`fftw 128`
575.71	1,736,993.73	0.1%	0.00	`fftw 256`
1,304.05	766,839.38	0.1%	0.00	`fftw 512`
3,016.67	331,491.71	0.0%	0.00	`fftw 1024`
6,541.71	152,865.13	0.1%	0.00	`fftw 2048`
14,763.67	67,733.85	0.4%	0.00	`fftw 4096`

Ok, how about the other guy?

```
struct FFT {
    using complex = std::complex<double>;
    FFT(int log2n)
        : log2n(log2n)
        , N(1 << log2n)
        , in(static_cast<fftw_complex *>(fftw_malloc(sizeof(fftw_complex) * N)))
        , out(static_cast<fftw_complex *>(fftw_malloc(sizeof(fftw_complex) * N)))
        , p(fftw_plan_dft_1d(N, in, out, FFTW_FORWARD, FFTW_ESTIMATE)) {
    }

    ~FFT() {
        fftw_destroy_plan(p);
        fftw_free(in);
        fftw_free(out);
    }

    void fill_input(std::vector<complex> const& input) {
        std::copy(input.begin(), input.end(), reinterpret_cast<complex*>(in));
    }

    std::vector<complex> fetch_output() const {
        auto output = reinterpret_cast<complex*>(out);
        return { output, output + N };
    }

    void execute() {
        fftw_execute(p);
    }

    int log2n{};
    int N{};
    fftw_complex *in{};
    fftw_complex *out{};
    fftw_plan p{};
};
```

```
struct vDSP_FFT {
    using complex = DSPDoubleComplex;
    vDSP_FFT(int log2n)
        : log2n(log2n)
        , N(1 << log2n)
        , in(N)
        , out(N)
        , setup(vDSP_DFT_Interleaved_CreateSetupD(nullptr, N, vDSP_DFT_FORWARD, vDSP_DFT_Interleaved_ComplexToComplex))
    }

    ~vDSP_FFT() {
        vDSP_DFT_Interleaved_DestroySetupD(setup);
    }

    void fill_input(std::vector<complex> const& input) {
        std::copy(input.begin(), input.end(), in.begin());
    }

    std::vector<complex> fetch_output() const {
        return out;
    }

    void execute() {
        vDSP_DFT_Interleaved_ExecuteD(setup, in.data(), out.data());
    }

    int log2n{};
    int N{};
    std::vector<DSPDoubleComplex> in;
    std::vector<DSPDoubleComplex> out;
    vDSP_DFT_Interleaved_SetupD setup;
};
```

No workie...

The screenshot shows a dark-themed IDE interface. The main area displays a C++ file named `test_fft_benchmark.cpp`. The code contains several loops and function calls, including `#include <Accelerate/Accelerate.h>`, `FFT`, `vDSP_FFT`, and `ankerl::nanobench::Bench`. The terminal output at the bottom shows a linking process for an executable named `test_fft_benchmark`. A red box highlights the error message: `build] Undefined symbols for architecture arm64:` followed by a list of undefined symbols related to `_vDSP_DFT_Interleaved` functions.

```
test_fft_benchmark.cpp — test_fft_benchmark
CMakeLists.txt  test_fft_benchmark.cpp  2

1  #include <Accelerate/Accelerate.h>
2
3  int main() {
4      ...
5
6      for (auto i : { 7, 8, 9, 10, 11, 12}) {
7          auto fft = FFT(i);
8          auto in = std::vector<FFT::complex>(fft.N);
9          for (auto i = 0; i < fft.N; ++i) {
10              in[i] = cos(i * 2 * std::numbers::pi/fft.N);
11          }
12          ankerl::nanobench::Bench().run(std::string("fftw ") + std::to_string(fft.N) , [&] {
13              fft.execute();
14          });
15      }
16      for (auto i : { 7, 8, 9, 10, 11, 12 }) {
17          auto fft = vDSP_FFT(i);
18          auto in = std::vector<vDSP_FFT::complex>(fft.N);
19          for (auto i = 0; i < fft.N; ++i) {
20              in[i].real = cos(i * 2 * std::numbers::pi/fft.N);
21          }
22          ankerl::nanobench::Bench().run(std::string("vDSP ") + std::to_string(fft.N) , [&] {
23              fft.execute();
24          });
25      }
26  }

PROBLEMS  2  OUTPUT  TERMINAL
[build] [100%] Linking CXX executable test_fft_benchmark
[build] Undefined symbols for architecture arm64:
[build]   "_vDSP_DFT_Interleaved_CreateSetupD", referenced from:
[build]     _main in test_fft_benchmark.cpp.o
[build]     vDSP_FFT::vDSP_FFT(int) in test_fft_benchmark.cpp.o
[build]   "_vDSP_DFT_Interleaved_DestroySetupD", referenced from:
[build]     _main in test_fft_benchmark.cpp.o
[build]   "_vDSP_DFT_Interleaved_ExecuteD", referenced from:
[build]     main in test_fft_benchmark.cpp.o
```

Stack Overflow to the rescue again!

The screenshot shows a web browser window with the URL stackoverflow.com in the address bar. The search bar contains the query "AudioUnit". The results page displays 5 answers. A specific answer is highlighted with a black rectangular box. The answer starts with a heading "Here's some simple example with AudioUnit:" followed by CMake code:

```
find_library(AUDIO_UNIT AudioUnit)
if (NOT AUDIO_UNIT)
    message(FATAL_ERROR "AudioUnit not found")
endif()

add_executable(program ${program_SOURCES})
target_link_libraries(program ${AUDIO_UNIT})
```

Below the code, there is a comment from user Fraser: "Your if not(...) should be if(NOT ...) I think." and a note from user lef: "Still not working with CMake 2.8.12, if program is actually static library target. In Build Settings Xcode pane, Linker flags remain empty, framework search paths remain empty, in Build Phases pane, no dependency is added, no linkage is added. I observed that in empty project, it is sufficient to define linker flags and search paths." There are also several related questions listed on the right side of the page.

Add Accelerate to cmake

```
1  cmake_minimum_required(VERSION 3.23)
2
3  set (CMAKE_CXX_STANDARD 20)
4
5  find_package(PkgConfig REQUIRED)
6  pkg_search_module(FFTW REQUIRED fftw3 IMPORTED_TARGET)
7  include_directories(PkgConfig::FFTW)
8  link_libraries      (PkgConfig::FFTW)
9
10 include(FetchContent)
11
12 √ FetchContent_Declare(
13     |   nanobench
14     |   GIT_REPOSITORY https://github.com/martinus/nanobench.git
15     |   GIT_TAG v4.1.0
16     |   GIT_SHALLOW TRUE)
17
18 FetchContent_MakeAvailable(nanobench)
19
20 √ project(
21     |   test_fft_benchmark
22     |   VERSION 1.0
23     |   LANGUAGES CXX)
24
25 add_executable(test_fft_benchmark test_fft_benchmark.cpp)
26 target_link_libraries(test_fft_benchmark PRIVATE nanobench)
27
28
```

```
1  cmake_minimum_required(VERSION 3.23)
2
3  set (CMAKE_CXX_STANDARD 20)
4
5  find_package(PkgConfig REQUIRED)
6  pkg_search_module(FFTW REQUIRED fftw3 IMPORTED_TARGET)
7  include_directories(PkgConfig::FFTW)
8  link_libraries      (PkgConfig::FFTW)
9
10 include(FetchContent)
11
12 FetchContent_Declare(
13     |   nanobench
14     |   GIT_REPOSITORY https://github.com/martinus/nanobench.git
15     |   GIT_TAG v4.1.0
16     |   GIT_SHALLOW TRUE)
17
18 FetchContent_MakeAvailable(nanobench)
19
20 project(
21     |   test_fft_benchmark
22     |   VERSION 1.0
23     |   LANGUAGES CXX)
24
25 find_library(ACCELERATE_LIB Accelerate)
26 add_executable(test_fft_benchmark test_fft_benchmark.cpp)
27 target_link_libraries(test_fft_benchmark PRIVATE nanobench ${ACCELERATE}
28
```

And Viola!

```
1 cmake_minimum_required(VERSION 3.23)
2
3 set (CMAKE_CXX_STANDARD 20)
4
5 find_package(PkgConfig REQUIRED)
6 pkg_search_module(FFTW REQUIRED fftw3 IMPORTED_TARGET)
7 include_directories(PkgConfig::FFTW)
8 link_libraries      (PkgConfig::FFTW)
9
10 include(FetchContent)
11
12 FetchContent_Declare(
13   nanobench
14   GIT_REPOSITORY https://github.com/martinus/nanobench.git
15   GIT_TAG v4.1.0
16   GIT_SHALLOW TRUE)
17
18 FetchContent_MakeAvailable(nanobench)
19
20 project(
21   test_fft_benchmark
22   VERSION 1.0
23   LANGUAGES CXX)
24
25 find_library(ACCELERATE_LIB Accelerate)
26 add_executable(test_fft_benchmark test_fft_benchmark.cpp)
27 target_link_libraries(test_fft_benchmark PRIVATE nanobench ${ACCELERATE_LIB})
28
```

	ns/op	op/s	err%	total	benchmark
	246.35	4,059,256.78	0.3%	0.00	`fftw 128`
	582.77	1,715,933.01	1.1%	0.00	`fftw 256`
	1,383.33	722,891.57	2.3%	0.00	`fftw 512`
	3,097.22	322,869.96	2.5%	0.00	`fftw 1024`
	6,553.57	152,588.56	1.9%	0.00	`fftw 2048`
	14,722.33	67,924.02	1.1%	0.00	`fftw 4096`
	199.68	5,007,957.56	0.4%	0.00	`accelerate 128`
	459.70	2,175,316.14	0.4%	0.00	`accelerate 256`
	1,251.24	799,210.19	0.6%	0.00	`accelerate 512`
	2,816.67	355,029.59	1.0%	0.00	`accelerate 1024`
	5,785.71	172,839.51	0.2%	0.00	`accelerate 2048`
	13,291.75	75,234.64	1.0%	0.00	`accelerate 4096`

ns/op	op/s	err%	total	benchmark
246.35	4,059,256.78	0.3%	0.00	`fftw 128`
582.77	1,715,933.01	1.1%	0.00	`fftw 256`
1,383.33	722,891.57	2.3%	0.00	`fftw 512`
3,097.22	322,869.96	2.5%	0.00	`fftw 1024`
6,553.57	152,588.56	1.9%	0.00	`fftw 2048`
14,722.33	67,924.02	1.1%	0.00	`fftw 4096`
199.68	5,007,957.56	0.4%	0.00	`accelerate 128`
459.70	2,175,316.14	0.4%	0.00	`accelerate 256`
1,251.24	799,210.19	0.6%	0.00	`accelerate 512`
2,816.67	355,029.59	1.0%	0.00	`accelerate 1024`
5,785.71	172,839.51	0.2%	0.00	`accelerate 2048`
13,291.75	75,234.64	1.0%	0.00	`accelerate 4096`

