

ONLINE TOOLS EVERY C++ DEVELOPER SHOULD KNOW



Thamara Andrade



HELLO!

I am Thamara Andrade (she/her)
Lead Software Engineer @ Cadence



@thamyk



<https://thamara.dev>



VERY IMPATIENT

HELLO!

I am Thamara Andrade (she/her)
Lead Software Engineer @ Cadence



@thamyk



<https://thamara.dev>

C++ shell

cpp.sh
online C++ compiler
[about cpp.sh](#)

```
1 // Example program
2 #include <iostream>
3 #include <string>
4
5 int main()
6 {
7     std::string name;
8     std::cout << "What is your name? ";
9     getline(std::cin, name);
10    std::cout << "Hello, " << name << "!\n";
11 }
12
```

Short URL: cpp.sh/

Run

options compilation execution

Standard

- ☐ C++98
☐ C++11
☒ C++14

Warnings

- ☒ Many (-Wall)
☐ Extra (-Wextra)
☐ Pedantic (-Wpedantic)

Optimization level

- ☐ None (-O0)
☐ Moderate (-O1)
☒ Full (-O2)
☐ Maximum (-O3)

Standard Input

- ☐ None
☒ Interactive
☐ Text:

⚡

main.cpp

```
1 - /*****
2
3 Welcome to GDB Online.
4 GDB online is an online compiler and debugger tool for C, C++, Python, Java, PHP, Ruby, Perl,
5 C#, VB, Swift, Pascal, Fortran, Haskell, Objective-C, Assembly, HTML, CSS, JS, SQLite, Prolog.
6 Code, Compile, Run and Debug online from anywhere in world.
7
8 *****/
9 #include <iostream>
10
11 int main()
12 {
13     int n = 3;
14     int nSquare = n * n;
15     std::cout << nSquare << std::endl;
16     return 0;
17 }
18
```

input

Debug Console

▶ start

⏸ pause

▶ continue

▶ step over

▶ step into

▶ step out

📘 help

Breakpoint 1, main () at main.cpp:14
14 int nSquare = n * n;
(gdb) step
15 std::cout << nSquare << std::endl;
(gdb)

Call Stack

#	Function	File:Line
0	main	main.cpp:15

Local Variables

Variable	Value
n	3
nSquare	9

Display Expressions

Expression	Value
Enter expression to watch	

Breakpoints and Watchpoints

	#	Description	
<input checked="" type="checkbox"/>	1	in main() at main.cpp:14	✕

Quick C++ Benchmark

<http://quick-bench.com>



Quick C++ Benchmark

Run Quick Bench locally

Support Quick Bench Suite ▾ More ▾

```
1 static void StringCreation(benchmark::State& state) {  
2     // Code inside this loop is measured repeatedly  
3     for (auto _ : state) {  
4         std::string created_string("hello");  
5         // Make sure the variable is not optimized away by compiler  
6         benchmark::DoNotOptimize(created_string);  
7     }  
8 }  
9 // Register the function as a benchmark  
10 BENCHMARK(StringCreation);  
11  
12 static void StringCopy(benchmark::State& state) {  
13     // Code before the loop is not measured  
14     std::string x = "hello";  
15     for (auto _ : state) {  
16         std::string copy(x);  
17     }  
18 }  
19 BENCHMARK(StringCopy);  
20
```

compiler = Clang 12.0 ▾

std = c++20 ▾

optim = O3 ▾

STL = libstdc++(GNU) ▾

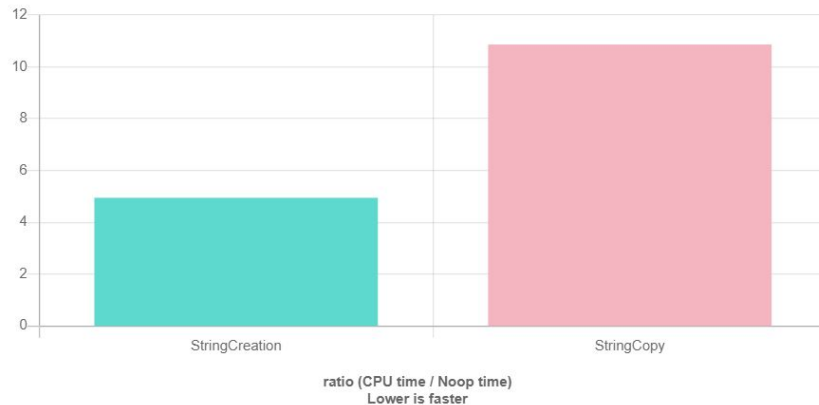
🔄 Run Benchmark

☒ Record disassembly ☐ Clear cached results



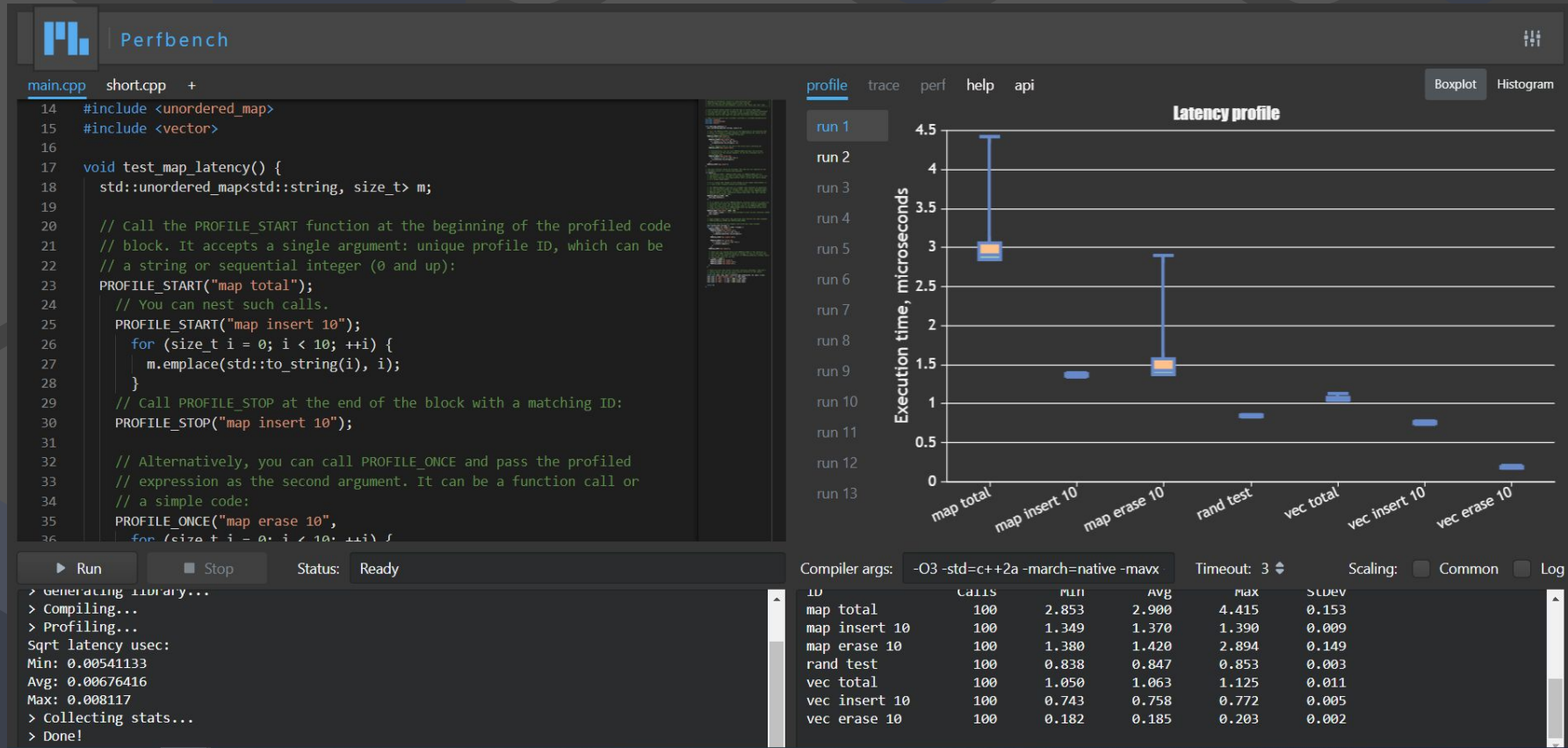
Charts

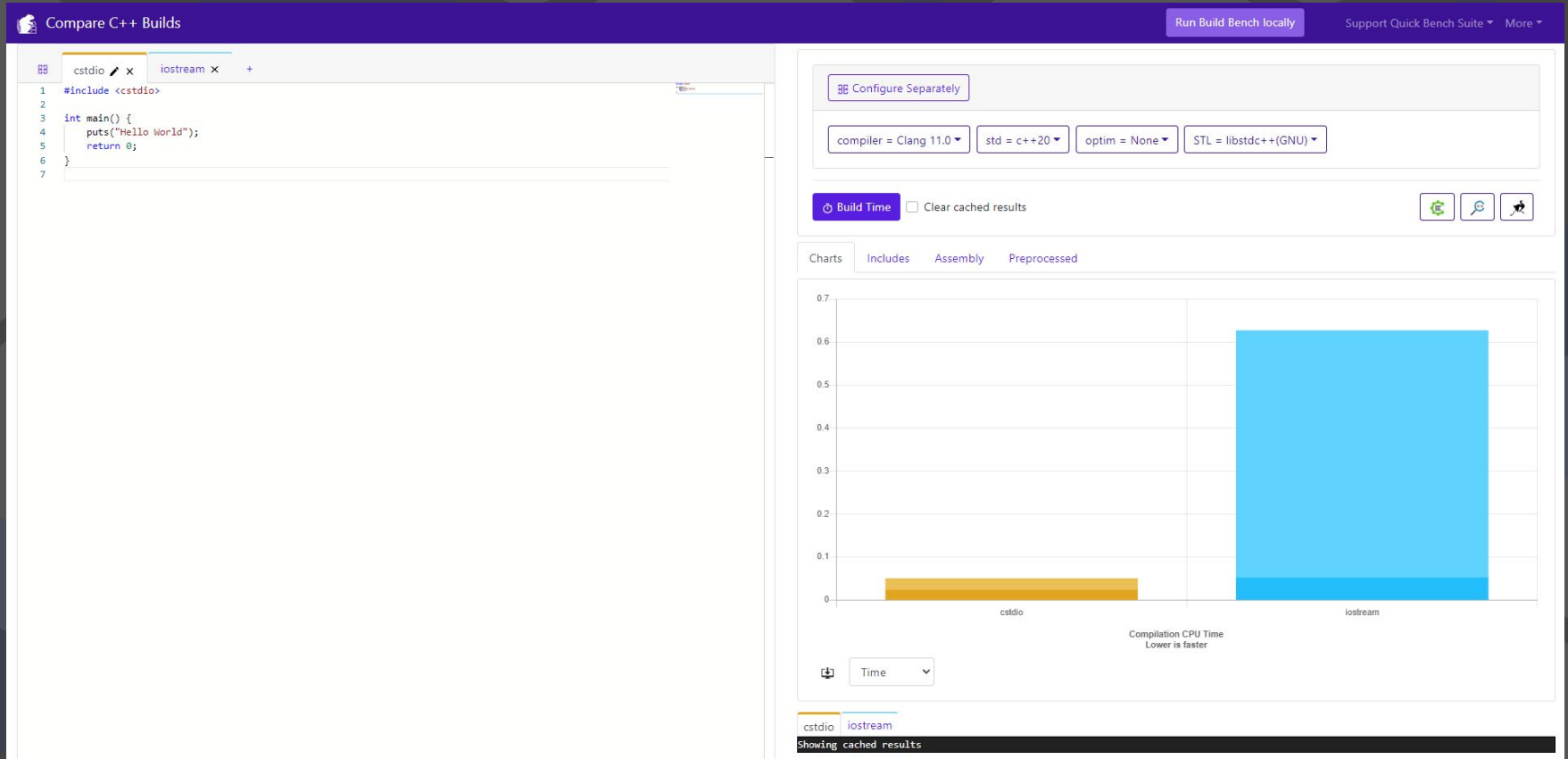
Assembly










☐ Show Noop bar

Showing cached results







[C++ Standard: C++ 17] ▾ D... ▾ More

New C++ Insights Episode ×

Made by [Andreas Fertig](#)
Powered by [Flask](#) and [CodeMirror](#)

Source:

```
1 int main()
2 {
3     // Generic lambdas have a method template call operator.
4     auto x = [](auto x) { return x * x; };
5
6     x(2); // int
7     x(3.0); // double
8 }
```

Insight:

```
1 int main()
2 {
3
4     class __lambda_4_12
5     {
6     public:
7         template<class type_parameter_0_0>
8         inline /*constexpr */ auto operator()(type_parameter_0_0 x) const
9         {
10             return x * x;
11         }
12
13         #ifdef INSIGHTS_USE_TEMPLATE
14         template<>
15         inline /*constexpr */ int operator()(int x) const
16         {
17             return x * x;
18         }
19         #endif
20
21
22         #ifdef INSIGHTS_USE_TEMPLATE
23         template<>
24         inline /*constexpr */ double operator()(double x) const
25         {
26             return x * x;
```

Console: ▮

Insights exited with result code: 0

Compiler Explorer (Godbolt)

<http://godbolt.org>

The screenshot displays the Compiler Explorer (Godbolt) interface. The top bar includes the logo, navigation links (Add..., More...), a promotional banner for the Compiler Explorer shop, sponsor logos (Intel, PC-lint, Solid), and utility links (Share, Policies, Other).

The main workspace is divided into four panes:

- Source Code:** Shows C++ source code for a function `square(int num)` that returns `num * num`.
- Compiler #1 (x86-64 gcc 7.2):** Displays the assembly output for the first compiler. The assembly includes stack frame setup, pushing `rbp`, moving `rsp` to `rbp`, storing the argument `num` at `[rbp-4]`, loading it into `eax`, multiplying `eax` by itself (`imul eax, DWORD PTR [rbp-4]`), and returning.
- Diff Viewer:** Compares the assembly of gcc 7.2 (Left) with clang 12.0.0 (Right). The diff shows differences in stack frame setup and argument passing, with clang using `dword ptr` for the stack slot.
- Compiler #2 (x86-64 clang 12.0.0):** Displays the assembly output for the second compiler, showing a different stack frame setup and argument passing compared to gcc.

The bottom status bar shows the output of the compilers, indicating successful compilation for both.



THANKS!



@thamyk



<https://thamara.dev>