

Godbolt Compiler Explorer

December 2024

Overview

- What is Compiler Explorer?
- Features
- Other Tools
- Summary

What is **Compiler Explorer**?

What is Compiler Explorer?



github.com/mattgodbolt

- Created by Matt Godbolt
 - Former Google developer
 - Worked in the video game and Quant Trading industry
 - Unix Geek

IDEA?

- Real time C/C++ Disassembly
 - Start in 2012
 - Just for C and C++ code
 - Simple Terminal commands
 - Only GCC on X86 machine
 - Grew in years

```

1 #include <stdio.h>
2
3 int main(void) {
4     printf("hello\n");
5     for (int i = 0; i < 5; i++) {
6         printf("counter: %d\n", i);
7     }
8 }
9

```

● Compile, Assembly, Print

watch gcc -S run.c -o -

When Started In 2012

```

Every 2.0s: gcc -S run.c -o -

.file   "run.c"
.text
.section .rodata
.LC0:
.string "hello"
.LC1:
.string "counter: %d\n"
.text
.globl main
.type   main, @function
main:
.LFB8:
.cfi_startproc
pushq   %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq    %rsp, %rbp
.cfi_def_cfa_register 6
subq    $16, %rsp
leaq    .LC0(%rip), %rax
movq    %rax, %rdi
call    puts@PLT
movl    $0, -4(%rbp)
jmp     .L2

.L3:
movl    -4(%rbp), %eax
movl    %eax, %esi
leaq    .LC1(%rip), %rax
movq    %rax, %rdi
movl    $0, %eax
call    printf@PLT
addl    $1, -4(%rbp)

.L2:
cmpl    $4, -4(%rbp)
jle     .L3
movl    $0, %eax
leave
.cfi_def_cfa 7, 8
ret
.cfi_endproc
.LFE8:
.size   main, .-main
.ident  "GCC: (Debian 14.2.0-8) 14.2.0"
.section .note.GNU-stack,"",@progbits

```

New Version



- Web Application
 - Godbolt.org
 - Many Features
 - Extensibility
 - Compiler Testing Platform
 - Education Playground

Now In 2024

godbolt.org

The screenshot displays the Godbolt Compiler Explorer interface. The top bar includes the 'COMPILER EXPLORER' logo, navigation buttons ('Add...', 'More...'), and a 'Compiler Explorer' dropdown menu. The main area is split into two panes. The left pane, titled 'C++ source (1)', shows a C++ program with a function 'square' that takes an integer and returns its square. The right pane, titled 'x86-64 gcc 10.2 (Compiler Explorer)', shows the corresponding assembly code for the 'square' function. The assembly code is as follows:

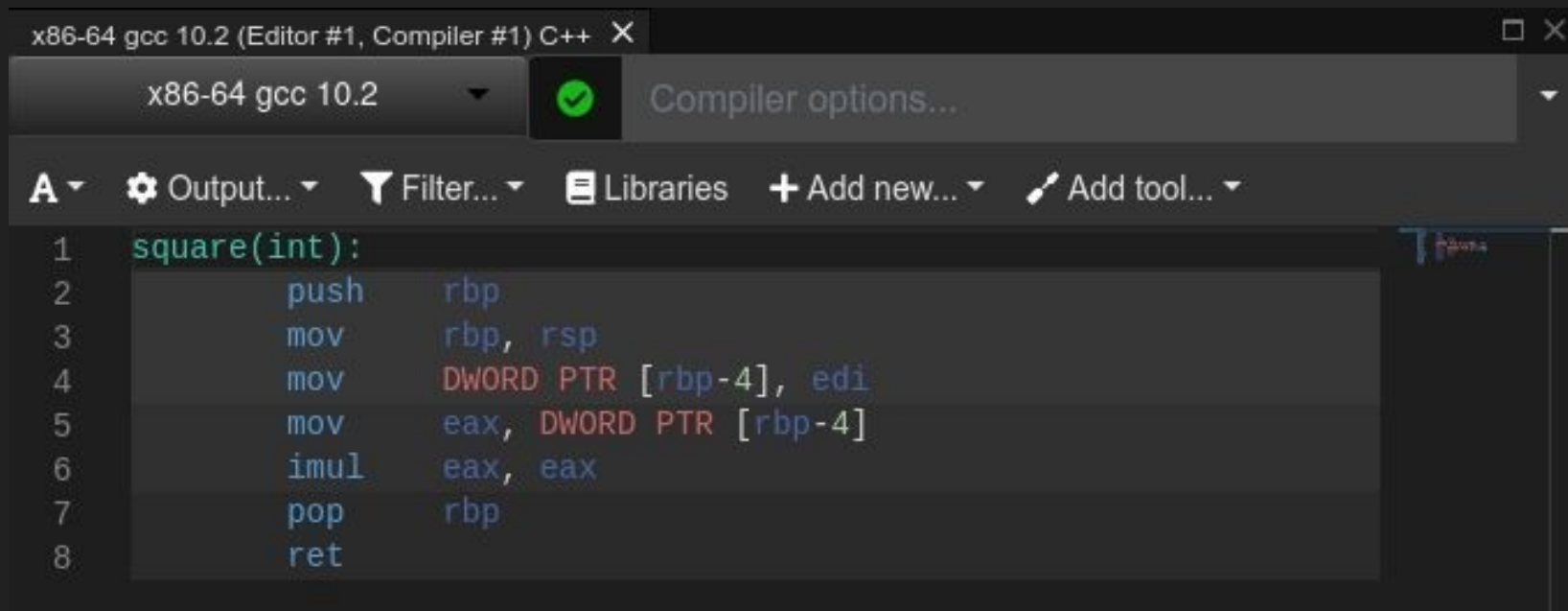
```
1 square(int):  
2     push    rbp  
3     mov     rbp, rbp  
4     mov     %rdi, PTR [rbp+8]  
5     mov     eax, QWORD PTR [rbp+8]  
6     imul    eax, eax  
7     pop     rbp  
8     ret
```

Below the assembly code, the 'Output' pane shows the compiled binary code for the 'square' function.

C++ Source

Disassembly

Right side: Disassembly



The screenshot shows the x86-64 gcc 10.2 compiler interface. The window title is "x86-64 gcc 10.2 (Editor #1, Compiler #1) C++". The toolbar includes a dropdown menu for "x86-64 gcc 10.2", a green checkmark icon, and a "Compiler options..." button. Below the toolbar is a row of icons: a list icon, a gear icon for "Output...", a funnel icon for "Filter...", a document icon for "Libraries", a plus icon for "Add new...", and a wrench icon for "Add tool...". The main area displays the disassembly of a function named "square(int):". The disassembly code is as follows:

```
1 square(int):  
2     push    rbp  
3     mov     rbp, rsp  
4     mov     DWORD PTR [rbp-4], edi  
5     mov     eax, DWORD PTR [rbp-4]  
6     imul    eax, eax  
7     pop     rbp  
8     ret
```

X86-64 gcc 10.2 compiler

Right side: Disassembly

The image displays two side-by-side disassembly windows from a debugger. The left window is titled 'armv7-a clang 19.1.0 (Editor #1)' and shows the assembly for a function named 'square(int):'. The assembly uses registers: 'sub sp, sp, #4', 'str r0, [sp]', 'ldr r0, [sp]', 'ldr r1, [sp]', 'mul r0, r0, r1', 'add sp, sp, #4', and 'bx lr'. The right window is titled 'mips gcc 14.2.0 (Editor #1)' and shows the assembly for the same function. It uses stack frames and registers: 'addiu \$sp, \$sp, -8', 'sw \$fp, 4(\$sp)', 'move \$fp, \$sp', 'sw \$4, 8(\$fp)', 'lw \$2, 8(\$fp)', 'nop', 'mult \$2, \$2', 'mflo \$2', 'move \$sp, \$fp', 'lw \$fp, 4(\$sp)', 'addiu \$sp, \$sp, 8', 'jr \$31', and 'nop'.

```
armv7-a clang 19.1.0 (Editor #1)
armv7-a clang 19.1.0
Compiler options...

1 square(int):
2     sub    sp, sp, #4
3     str    r0, [sp]
4     ldr    r0, [sp]
5     ldr    r1, [sp]
6     mul    r0, r0, r1
7     add    sp, sp, #4
8     bx     lr

mips gcc 14.2.0 (Editor #1)
mips gcc 14.2.0
Compiler options...

1 square(int):
2     addiu   $sp, $sp, -8
3     sw      $fp, 4($sp)
4     move    $fp, $sp
5     sw      $4, 8($fp)
6     lw      $2, 8($fp)
7     nop
8     mult    $2, $2
9     mflo    $2
10    move    $sp, $fp
11    lw      $fp, 4($sp)
12    addiu   $sp, $sp, 8
13    jr      $31
14    nop
```

arm7 and MIPS

Customization

source

gcc

clang



The image displays three side-by-side screenshots of a code editor, illustrating the customization of the compilation environment. Each window shows the same source code for a C++ function named `square`, which takes an integer `n` and returns `n * n`.

The first window, titled "C++ source #1", shows the source code:

```
1 // Type your code here, or
2 int square(int n) {
3     return n * n;
4 }
```

The second window, titled "x86_64 gcc 10.2 (Editor #1, Compiler #1) C++", shows the assembly output generated by GCC 10.2. The assembly code is as follows:

```
1 square(int):
2     push    rbp
3     mov     rbp, rsp
4     mov     QWORD PTR [rbp-4], n
5     mov     eax, QWORD PTR [rbp-4]
6     imul    eax, eax
7     pop     rbp
8     ret
```

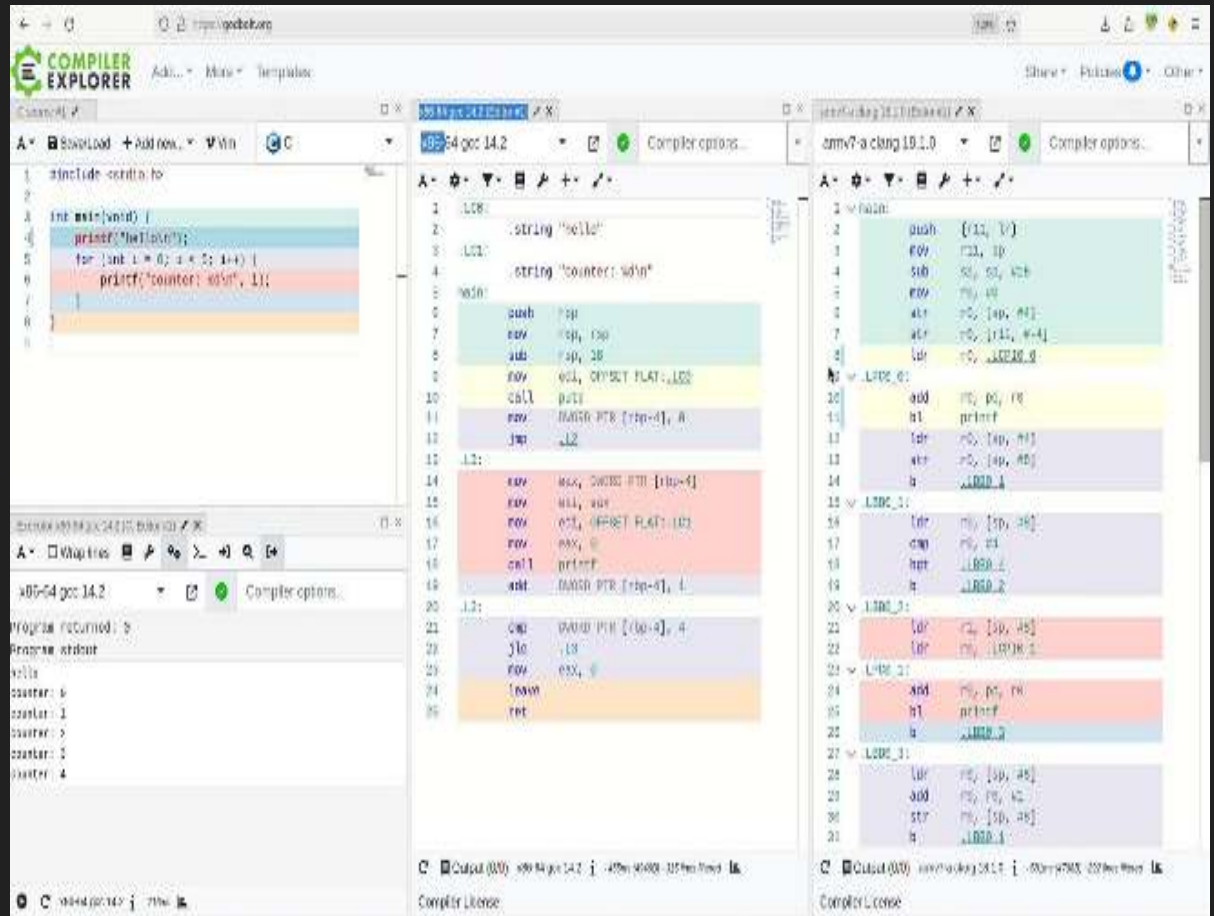
The third window, titled "x86_64 clang 11.0.1 (Editor #1, Compiler #2) C++", shows the assembly output generated by Clang 11.0.1. The assembly code is as follows:

```
1 square(int):
2     push    rbp
3     mov     rbp, rsp
4     mov     QWORD PTR [rbp-4], n
5     mov     eax, QWORD PTR [rbp-4]
6     imul    eax, QWORD PTR [rbp-4]
7     pop     rbp
8     ret
```

Editor, Compiler

Workbench

- Editor
- Execution
- Compiler
- Tool chain



Lots of options...

ARM GCC

- ARM gcc 9.2.1 (none)
- ARM gcc 8.3.1 (none)
- ARM64 gcc 8.2
- ARM gcc 8.2 (WinCE)
- ARM gcc 8.2
- ARM64 gcc 7.3
- ARM gcc 7.3
- ARM gcc 7.2.1 (none)
- ARM64 gcc 6.4
- ARM gcc 6.4
- ARM64 gcc 6.3.0 (linux)
- ARM gcc 6.3.0 (linux)
- ARM gcc 5.4.1 (none)
- ARM64 gcc 5.4 (linux)

MSVC X64

- x64 msvc v19.28
- x64 msvc v19.27
- x64 msvc v19.25
- x64 msvc v19.24
- x64 msvc v19.23
- x64 msvc v19.22
- x64 msvc v19.21
- x64 msvc v19.20
- x64 msvc v19.16
- x64 msvc v19.15
- x64 msvc v19.14

MIPS GCC

- MIPS64 gcc 5.4 (el)
- MIPS64 gcc 5.4
- MIPS gcc 5.4 (el)
- MIPS gcc 5.4

MSP GCC

- MSP430 gcc 6.2.1
- MSP430 gcc 5.3.0
- MSP430 gcc 4.5.3

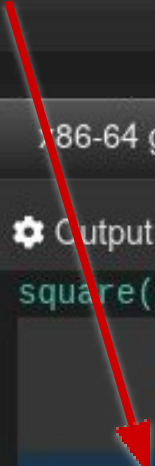
Compiler Explorers Features

Code Highlighting

```
A ▾ Save/Load + Add new... ▾ Vim CppInsights ▾
1 // Type your code here, or load an example.
2 int square(int num) {
3     return num * num;
4 }
```

Match Code with
assembly instruction

```
x86-64 gcc 10.2 ▾ ✓ Compiler options...
A ▾ ⚙ Output... ▾ Filter... ▾ Libraries + Add new... ▾ Add tool... ▾
1 square(int):
2     push    rbp
3     mov     rbp, rsp
4     mov     DWORD PTR [rbp-4], edi
5     mov     eax, DWORD PTR [rbp-4]
6     imul    eax, eax
7     pop     rbp
8     ret
```



Documentation

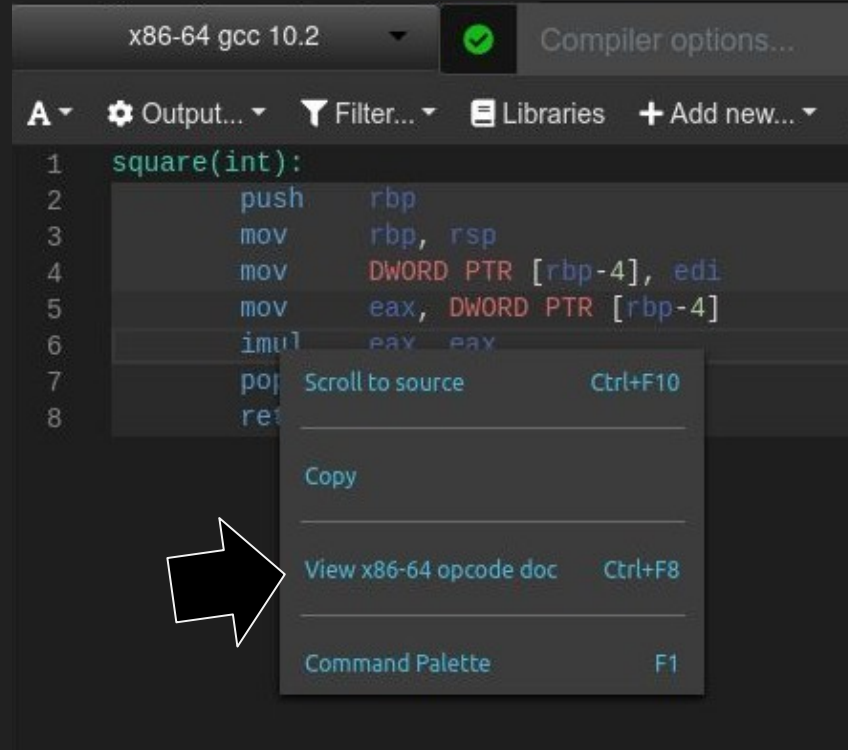
x86-64 gcc 10.2 Compiler options...

A Output... Filter... Libraries Add new... Add tool...


```
1 square(int):  
2     push    rbp  
3  
4     Performs a signed multiplication of two operands. This instruction has three  
5     forms, depending on the number of operands.  
6     More information available in the context menu.  
7     imul    eax, eax  
8     pop     bp  
9     ret
```

A white arrow points to the `ret` instruction on line 8.

Documentation



Documentation

 COMPILER

Add → More →

Document loaded 100% 2024-08-08 10:00:00

Show → Other → Profiles →

IMUL help

Performs a signed multiplication of two operands. This instruction has three forms, depending on the number of operands.

When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format.

The CF and OF flags are set when the signed integer value of the intermediate product differs from the sign-extended operand-size-truncated product, otherwise the CF and OF flags are cleared.

The three forms of the IMUL instruction are similar in that the length of the product is calculated to twice the length of the operands. With the one-operand form, the product is stored exactly in the destination. With the two- and three-operand forms, however, the result is truncated to the length of the destination before it is stored in the destination register. Because of this truncation, the CF or OF flag should be tested to ensure that no significant bits are lost.

The two- and three-operand forms may also be used with unsigned operands because the lower half of the product is the same regardless if the operands are signed or unsigned. The CF and OF flags, however, cannot be used to determine if the upper half of the result is non-zero.

For more information, visit the [IMUL documentation](#).

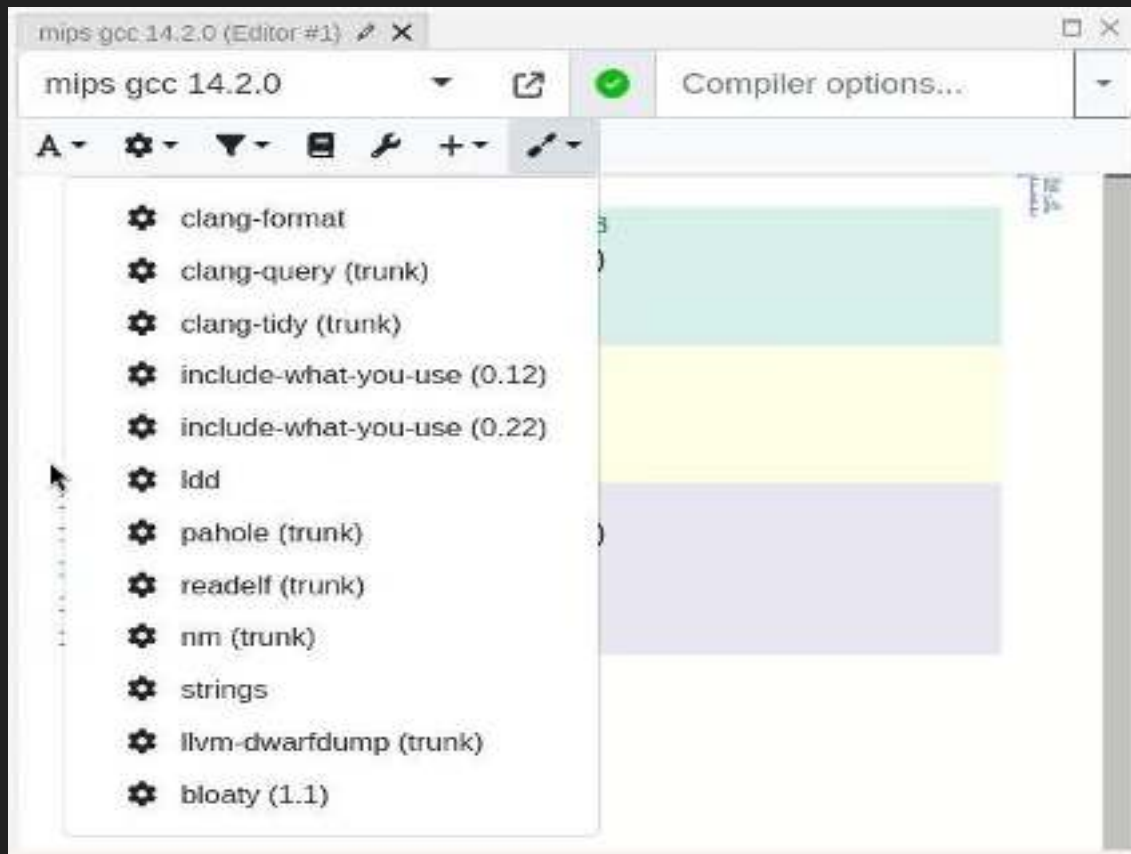
If the documentation for this opcode is wrong or broken in some way, please feel free to [open an issue on GitHub](#).

Close

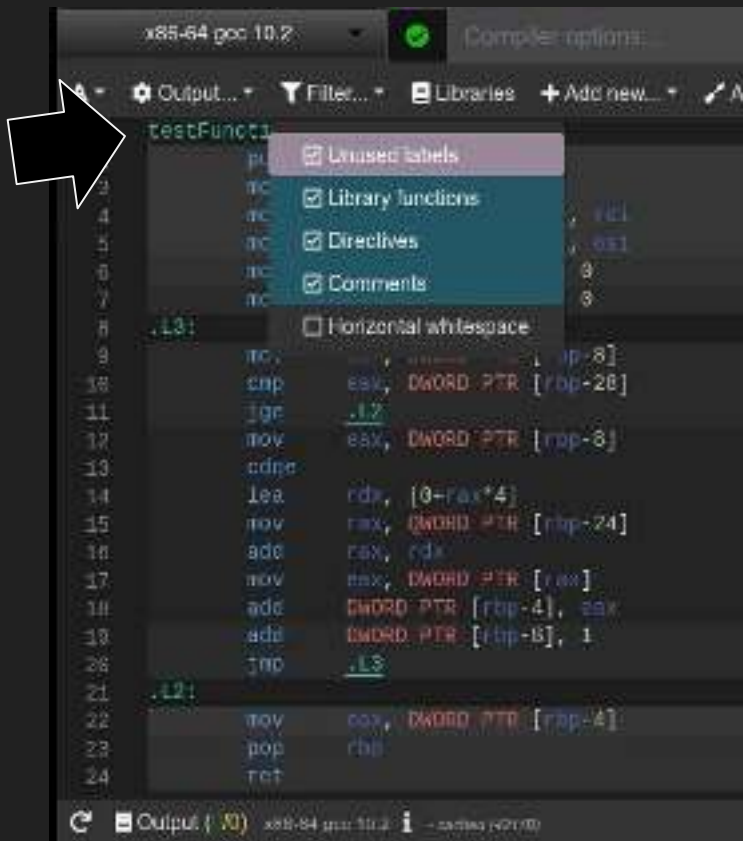
Even more probing tools...

CLI toolbox

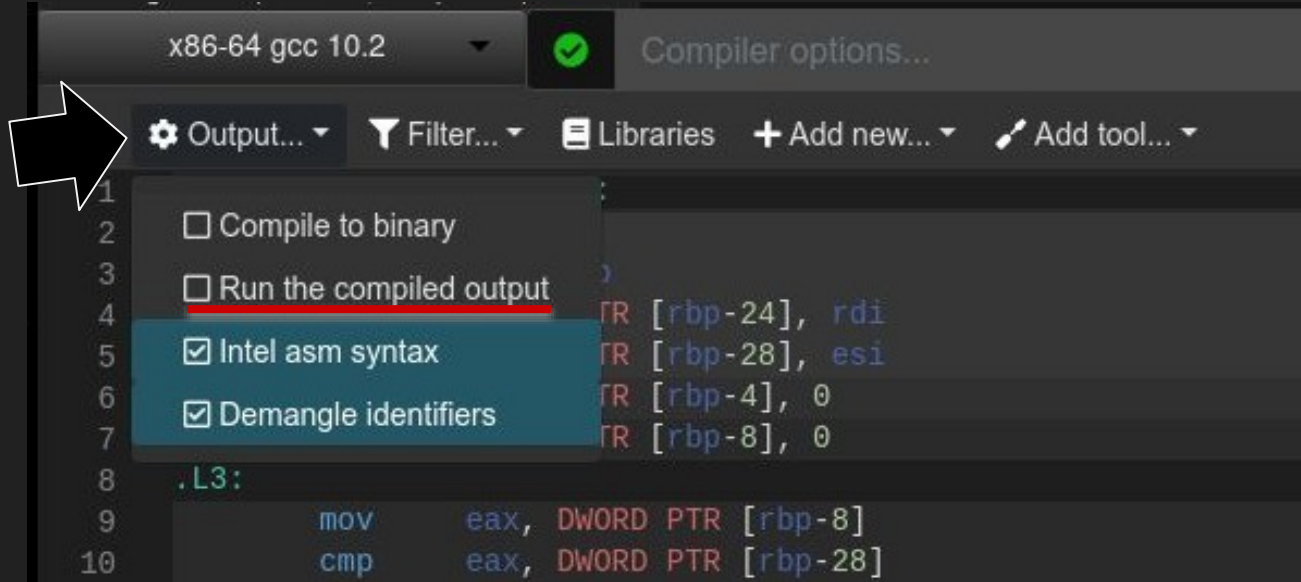
- ldd
- strings
- readelf



Filtering options



Output options



Code execution

```
C++ sources #1 X
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << "Hello world!" << std::endl;
6     return 0;
7 }
8
```

```
x86-64 gcc 10.2 (Editor #1, Compiler #1) C++ X
x86-64 gcc 10.2 -O2
Output... Filter... Libraries + Add new...
1 .LC0:
2     .string "Hello world:"
3 main:
4     push    r12
5     mov     esi, OFFSET FLAT:.LC0
6     mov     edi, OFFSET FLAT:_ZSt4cout
7     push    rbp
8     sub     rsp, 8
9     call    std::basic_ostream<char, st
Output (0) x86-64 gcc 10.2 i 1479ms (116640B)
#1 with x86-64 gcc 10.2 X
Wrap lines
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 0
Hello world!
```

Compiler options

```
C++ source #1 X
1 int getInteger(int x)
2 {
3     return x;
4 }
5
```

```
x86-64 gcc 10.2 (Editor #1, Compiler #1) C++ X
x86-64 gcc 10.2 -O0
A - [icon] [icon] [icon] [icon] [icon] [icon]
1 getInteger(int):
2     push    rbp
3     mov     rbp, rsp
4     mov     DWORD PTR [rbp-4], edi
5     mov     eax, DWORD PTR [rbp-4]
6     pop     rbp
7     ret
```

```
x86-64 gcc 10.2 (Editor #1, Compiler #2) C++ X
x86-64 gcc 10.2 -O2
A - [icon] [icon] [icon] [icon] [icon] [icon]
1 getInteger(int):
2     mov     eax, edi
3     ret
```

Compiler options

The screenshot displays the Visual Studio Code interface with a C source file on the left and three assembly output windows on the right, generated by GCC 14.2 for x86_64.

Source Code (left):

```
1 short foo(short a, short b) {  
2     short result;  
3     result = b;  
4     while(a > 0) {  
5         result ^= a;  
6         b -= a;  
7     }  
8     return result;  
9 }
```

Assembly Output 1 (top right): Compiled with `-O1`. The assembly shows instructions in order: `movl $0x1, %eax`, `movl %eax, %ecx`, `testl %ecx, %ecx`, `jle .L3`, `imull %ecx, %ecx`, `subl %ecx, %ecx`, `testl %ecx, %ecx`, `je .L3`, and `ret`.

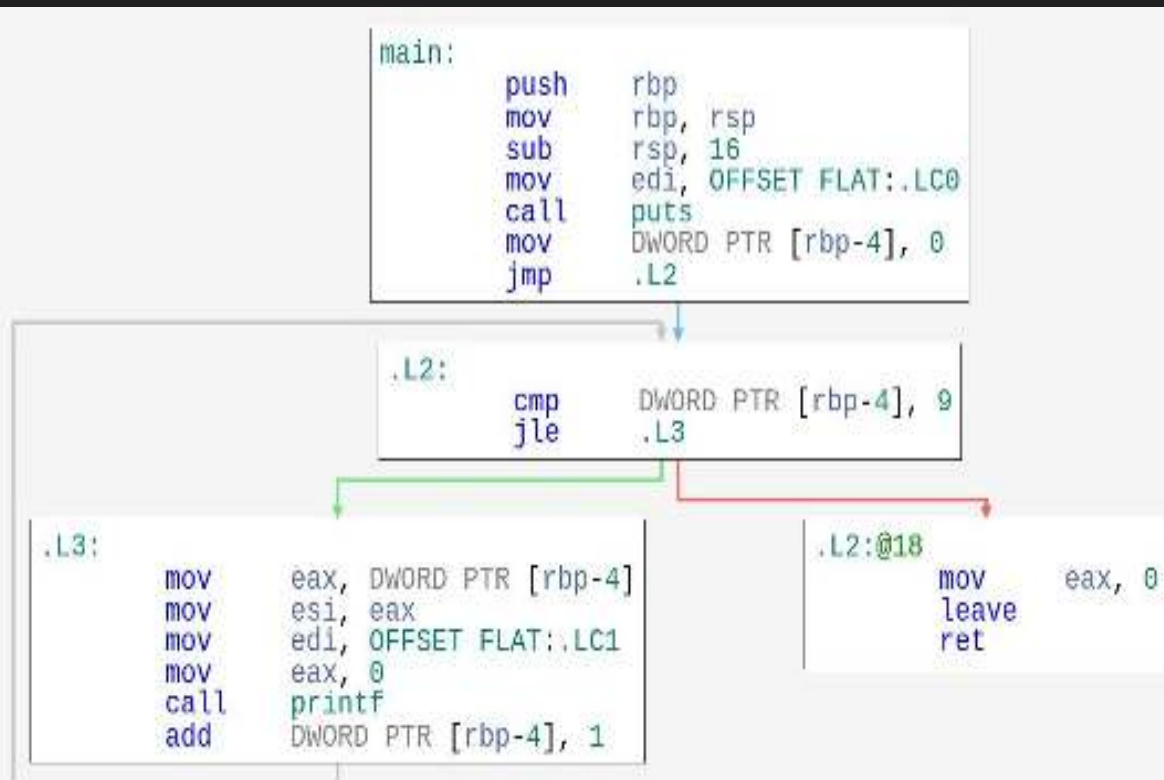
Assembly Output 2 (middle right): Compiled with `-O2`. The assembly shows instructions in order: `movl $0x1, %eax`, `movl %eax, %ecx`, `testl %ecx, %ecx`, `jle .L3`, `subl %ecx, %ecx`, `imull %ecx, %ecx`, `testl %ecx, %ecx`, `je .L3`, and `ret`.

Assembly Output 3 (bottom right): Compiled with `-O1 -fschedule-insns2`. The assembly shows instructions in order: `movl $0x1, %eax`, `movl %eax, %ecx`, `testl %ecx, %ecx`, `jle .L3`, `subl %ecx, %ecx`, `imull %ecx, %ecx`, `testl %ecx, %ecx`, `je .L3`, and `ret`.

Instruction Order changed

- `imul`
- `Subl`
- Gcc O1, O2 optimization level

Control Flow Graph



Other Tools

C++ Insights

C++ Insights



The image shows a screenshot of the Compiler Explorer web application. At the top left is the 'COMPILER EXPLORER' logo. To its right are two buttons: 'Add...' and 'More'. Below the logo, a tab labeled 'C++ source #1' is visible. A toolbar contains several icons and labels: a compiler selection dropdown (showing 'A'), a 'Save/Load' button, an 'Add new...' button, a 'Vim' button, a 'CppInsights' button with a magnifying glass icon, and a 'Quick-bench' button with a speedometer icon. A white arrow points to the 'CppInsights' button. The main area displays C++ code with line numbers 17 through 24. The code defines a double array, initializes it, and starts a main function.

```
17 double result[N];  
18  
19 void initialize();  
20 void vm_multiply();  
21 void output();  
22  
23 int main() {  
24     int i;
```

C++ Insights

The screenshot displays the C++ Insights web interface. The top navigation bar includes a search icon, a dropdown menu set to 'C++ - standard c++17', and buttons for 'default' and 'None'. On the right, it says 'New C++ Insights episode' and 'Made by Andreas Fertig, Powered by Black and CodeMoose'.

The interface is split into two main panels: 'Source' on the left and 'Insights' on the right. Red arrows indicate the mapping from source code to the generated insights.

Source Code:

```
1 auto Best() -> int {
2     {}
3     return 1;
4 }
5
6 constexpr auto cBest() -> int {
7     {}
8     return 1;
9 }
10
11 decltype(auto) best() {
12     {}
13     return 'c';
14 }
15
16 constexpr decltype(auto) cBest() {
17     {}
18     return 'C';
19 }
20
21 [maybe_unused] inline constexpr decltype(auto) WithDefault() {
22     {}
23     return 'a';
24 }
```

Insights:

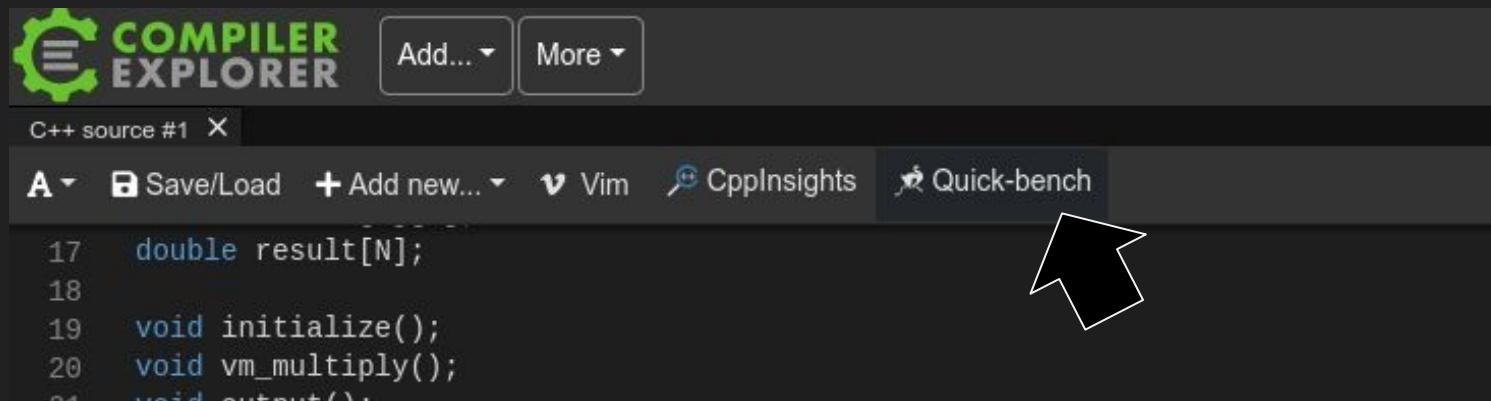
```
1 auto Best() {
2     {}
3     return 1;
4 }
5
6 inline constexpr int cBest() {
7     {}
8     return 1;
9 }
10
11 char best() {
12     {}
13     return 'c';
14 }
15
16 inline constexpr char cBest() {
17     {}
18     return 'C';
19 }
20
21 [maybe_unused] inline constexpr decltype(auto) WithDefault() {
22     {}
23     return 'a';
24 }
```

Console:

```
Insights exited with result code: 0
```

Quick-bench

Quick-bench



Quick C++ Benchmark

The screenshot displays the Quick C++ Benchmark web interface. On the left, a code editor shows C++ source code. A red arrow points from the `BM_SomeFunction` benchmark registration line in the code to the corresponding entry in the benchmark results table on the right.

Quick C++ Benchmark Discover Build Bench! Support Quick Bench Auto? More?

compiler = GCC 10.1 std = c++11 optim = O2
STL = libstdc++ (GNU)

☒ Record disassembly ☐ Clear cached results

Charts Assembly

| | |
|-----------|--|
| 300000000 | |
| 250000000 | |
| 200000000 | |
| 150000000 | |
| 100000000 | |
| 50000000 | |
| 0 | |

BM_SomeFunction
cpu time: 313229.5363, 2767824

BM_SomeFunction

```
80     if (i%10 == 0)
81         printf("v\n");
82         printf(" %6.2f", result[i]);
83     }
84     printf("\n");
85     return
86     result;
87 }
88
89 static void BM_SomeFunction(benchmark::State& state) {
90     // Perform setup here
91     for (auto _ : state) {
92         // This code gets timed
93         run();
94     }
95 }
96
97 // Register the function as a benchmark
98 BENCHMARK(BM_SomeFunction);
99
```


Quick C++ Benchmark

- Relies on Google Benchmark
 - github.com/google/benchmark

Summary

- It's in the browser!
- Edit, Compile, View Disassembly, benchmarking and execution.
- Many Languages, Compilers, architectures and configuration options
- C, C++, Rust, C#, GO, FORTRAN, Python, Ruby, Java, ...
- Armv7, AVR, X86, MIPS, RISC-V, SPARC, VAX, and many more.

Questions?

Thanks