C++ Insights

Peek behind the curtains of your C++ compiler

Presentation Material

CppNorth, Toronto, 2024-07-23

Andreas Fertig

Write unique code!

Version: v1.0

Planning, typesetting and cover design: Andreas Fertig
Cover art and illustrations: Franziska Panter https://franziskapanter.com
Production and publishing: Andreas Fertig

Style and conventions

The following shows the execution of a program. I used the Linux way here and skipped supplying the desired output name, resulting in `a.out` as the program name.

```
$ ./a.out
Hello, C++!
```

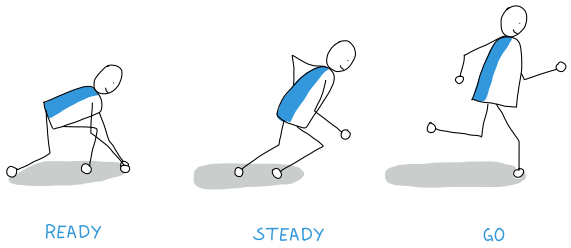
- `<string>` stands for a header file with the name `string`
- `[[xyz]]` marks a C++ attribute with the name `xyz`.

https://AndreasFertig.com
post@AndreasFertig.com

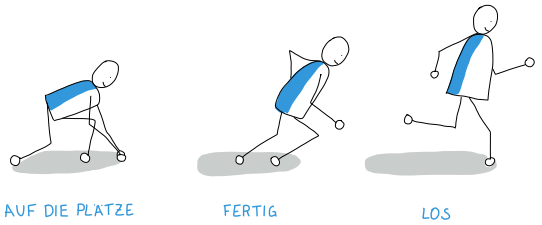
fertig

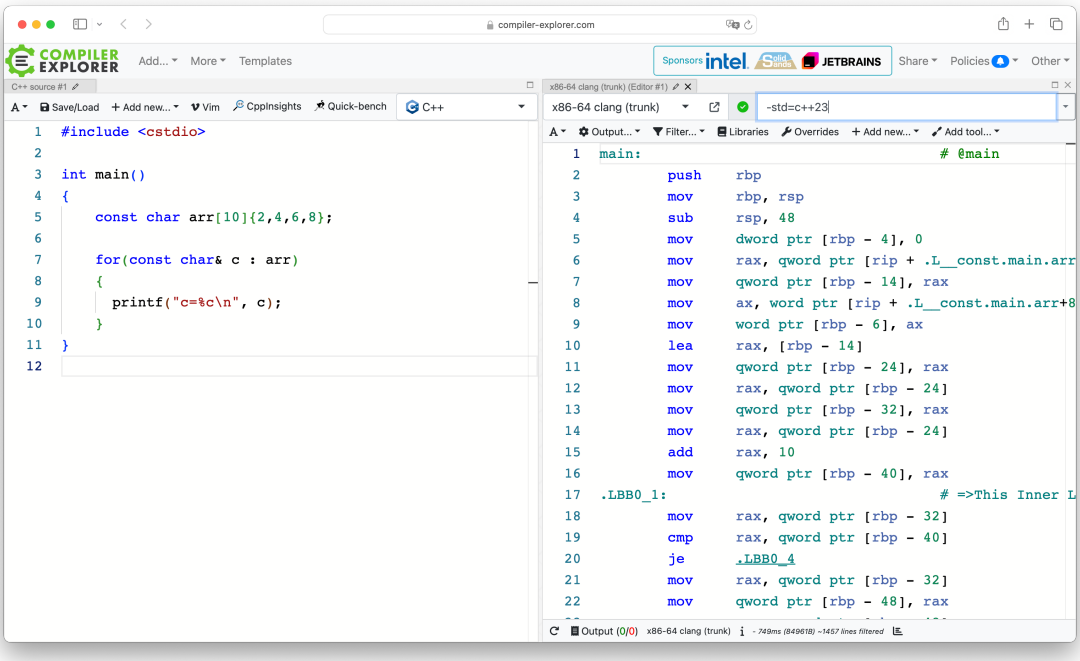
adjective /ˈfɛrtɪç/

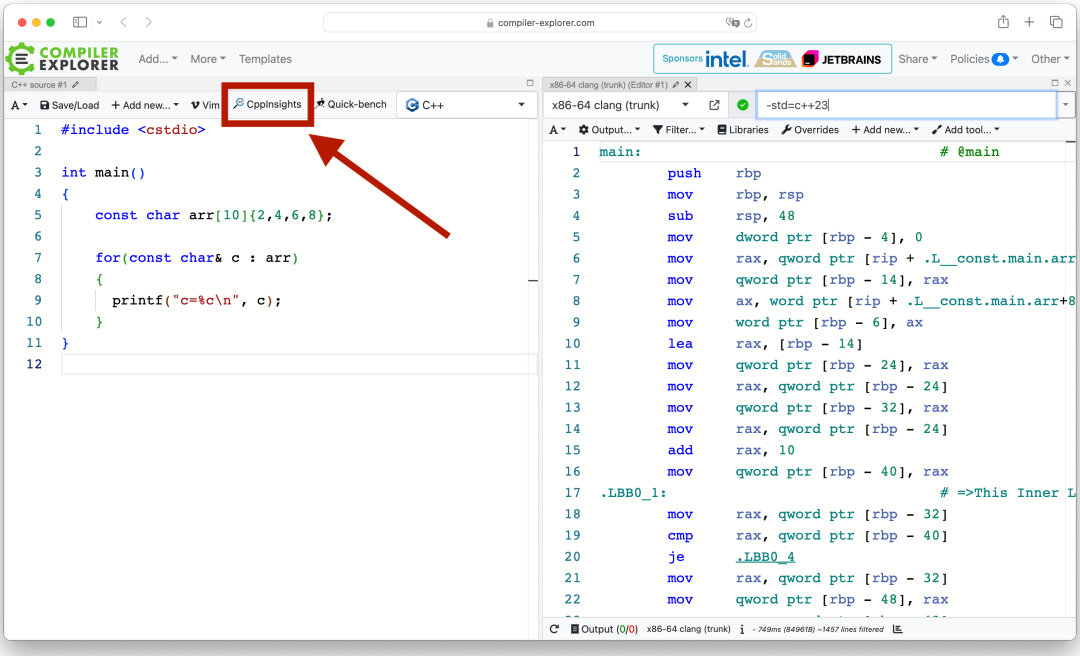
finished
ready
complete
completed

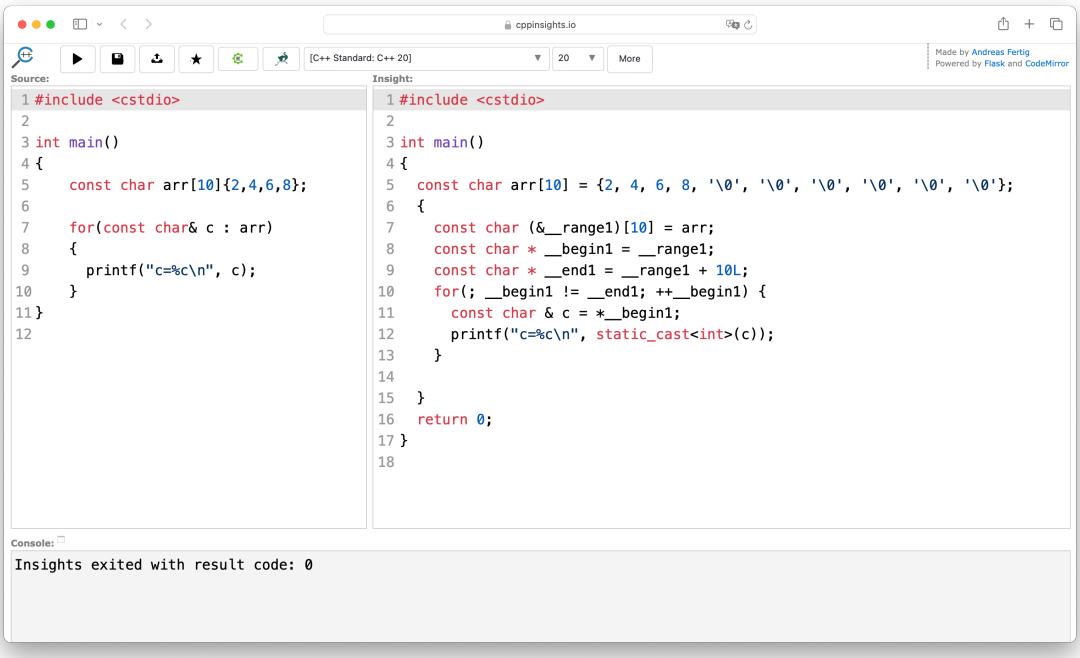


https://AndreasFertig.com
post@AndreasFertig.com


AUF DIE PLÄTZE
FERTIG
LOS


compiler-explorer.com
COMPILER EXPLORER
Add... More Templates
Sponsors intel JETBRAINS
Share Policies Other
C++ source #1
Save/Load + Add new... Vim CppInsights Quick-bench C++
x86-64 clang (trunk) (Editor #1)
x86-64 clang (trunk)
-std=c++23
Output... Filter... Libraries Overrides + Add new... Add tool...
1 #include <cstdio>
2
3 int main()
4 {
5 const char arr[10]{2,4,6,8};
6
7 for(const char& c : arr)
8 {
9 printf("c=%c\n", c);
10 }
11 }
12
1 main: # @main
2 push rbp
3 mov rbp, rsp
4 sub rsp, 48
5 mov dword ptr [rbp - 4], 0
6 mov rax, qword ptr [rip + .L__const.main.arr
7 mov qword ptr [rbp - 14], rax
8 mov ax, word ptr [rip + .L__const.main.arr+8
9 mov word ptr [rbp - 6], ax
10 lea rax, [rbp - 14]
11 mov qword ptr [rbp - 24], rax
12 mov rax, qword ptr [rbp - 24]
13 mov qword ptr [rbp - 32], rax
14 mov rax, qword ptr [rbp - 24]
15 add rax, 10
16 mov qword ptr [rbp - 40], rax
17 .LBB0_1: # =>This Inner L
18 mov rax, qword ptr [rbp - 32]
19 cmp rax, qword ptr [rbp - 40]
20 je .LBB0_4
21 mov rax, qword ptr [rbp - 32]
22 mov qword ptr [rbp - 48], rax
Output (0/0) x86-64 clang (trunk)

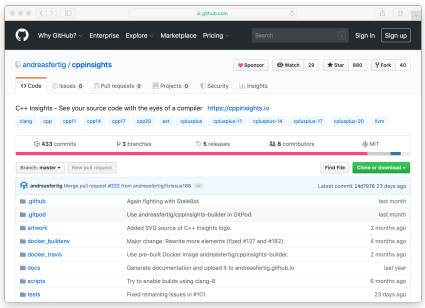

compiler-explorer.com
COMPILER EXPLORER
Add... More Templates
Sponsors intel JETBRAINS
Share Policies Other
C++ source #1
Save/Load + Add new... Vim CppInsights Quick-bench C++
x86-64 clang (trunk) (Editor #1)
x86-64 clang (trunk)
-std=c++23
Output... Filter... Libraries Overrides + Add new... Add tool...
#include <cstdio>
int main()
{
const char arr[10]{2,4,6,8};
for(const char& c : arr)
{
printf("c=%c\n", c);
}
}
main: # @main
push rbp
mov rbp, rsp
sub rsp, 48
mov dword ptr [rbp - 4], 0
mov rax, qword ptr [rip + .L__const.main.arr
mov qword ptr [rbp - 14], rax
mov ax, word ptr [rip + .L__const.main.arr+8
mov word ptr [rbp - 6], ax
lea rax, [rbp - 14]
mov qword ptr [rbp - 24], rax
mov rax, qword ptr [rbp - 24]
mov qword ptr [rbp - 32], rax
mov rax, qword ptr [rbp - 24]
add rax, 10
mov qword ptr [rbp - 40], rax
.LBB0_1: # =>This Inner L
mov rax, qword ptr [rbp - 32]
cmp rax, qword ptr [rbp - 40]
je .LBB0_4
mov rax, qword ptr [rbp - 32]
mov qword ptr [rbp - 48], rax
Output (0/0) x86-64 clang (trunk)


cppinsights.io
[C++ Standard: C++ 20]
20
More
Made by Andreas Fertig
Powered by Flask and CodeMirror
Source:
Insight:
#include <cstdio>
int main()
{
const char arr[10]{2,4,6,8};
for(const char& c : arr)
{
printf("c=%c\n", c);
}
}
#include <cstdio>
int main()
{
const char arr[10] = {2, 4, 6, 8, '\0', '\0', '\0', '\0', '\0', '\0'};
{
const char (&__range1)[10] = arr;
const char * __begin1 = __range1;
const char * __end1 = __range1 + 10L;
for(; __begin1 != __end1; ++__begin1) {
const char & c = *__begin1;
printf("c=%c\n", static_cast<int>(c));
}
}
return 0;
}
Console:
Insights exited with result code: 0

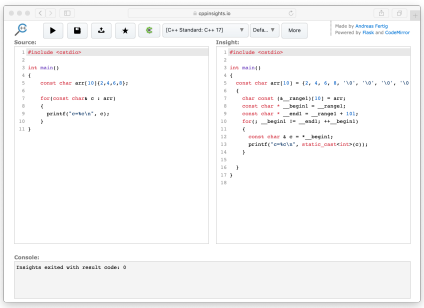
C++ Insights

- Show what is going on.
- Make invisible things visible to assist in teaching.
- Create valid code.
- Create code that compiles.
- *Of course, it is open-source.*
- You can run a local instance of C++ Insights [1].

https://github.com/andreasfertig/cppinsights/



https://cppinsights.io



About C++ Insights

- C++ Insights is a Clang-based tool.
  - Basically, it is a source-to-source transformation tool.
  - It uses Clang's AST. It is way more than a preprocessor!
- The official builds use the latest release version of Clang.
  - Hence, not all the newest interesting features are available.
- It uses the Clang AST, which shows no optimizations.
  - Hence, tuning with `-O n` does not change anything in C++ Insights.
- Thanks to Clang libstdc++ and libc++ are supported.
- Not *all* statements are currently matched.

https://AndreasFertig.com
post@AndreasFertig.com

A word about limitations: Templates

- Creating code that compiles from templates is hard.
- To make it a bit easier for me, there is a `#ifdef INSIGHTS_USE_TEMPLATE` to have the code, but inactive.

```
1 template<typename T>
2 void Func()
3 {}
4
5 class Demo
6 {
7 };
8
9 int main()
10 {
11   Func<Demo>();
12 }
```


What is an AST

```
`-FunctionDecl 0x106ee15a8 <astExample0/astExample0.cpp:3:1, line:6:1> line:3:5 main 'int ()'
  `-CompoundStmt 0x106ee3ed8 <line:4:1, line:6:1>
    `-CXXOperatorCallExpr 0x106ee3ea0 <line:5:3, col:16> 'basic_ostream<char, std::__1::char_traits<char> >':'std::__1::/
      basic_ostream<char>' lvalue adl
      |-ImplicitCastExpr 0x106ee3e88 <col:13> 'basic_ostream<char, std::__1::char_traits<char> > &(*)(basic_ostream<char,/
       std::__1::char_traits<char> > &, const char *)' <FunctionToPointerDecay>
      | `-DeclRefExpr 0x106ee3df0 <col:13> 'basic_ostream<char, std::__1::char_traits<char> > &(basic_ostream<char, std::/
      __1::char_traits<char> > &, const char *)' lvalue Function 0x106ee2800 'operator<<' 'basic_ostream<char, std::__1::/
      char_traits<char> > &(basic_ostream<char, std::__1::char_traits<char> > &, const char *)'
      |-DeclRefExpr 0x106ee1698 <col:3, col:8> 'std::__1::ostream':'std::__1::basic_ostream<char>' lvalue Var 0x106ee0fb8/
       'cout' 'std::__1::ostream':'std::__1::basic_ostream<char>'
      `-ImplicitCastExpr 0x106ee3dd8 <col:16> 'const char *' <ArrayToPointerDecay>
        `-StringLiteral 0x106ee16c8 <col:16> 'const char [13]' lvalue "Hello, C++!\n"
```

What is an AST

```
`-FunctionDecl 0x106ee15a8 <astExample0/astExample0.cpp:3:1, line:6:1> line:3:5 main 'int ()'
  `-CompoundStmt 0x106ee3ed8 <line:4:1, line:6:1>
    `-CXXOperatorCallExpr 0x106ee3ea0 <line:5:3, col:16> 'basic_ostream<char, std::__1::char_traits<char> >':'std::__1::/
      basic_ostream<char>' lvalue adl
      |-ImplicitCastExpr 0x106ee3e88 <col:13> 'basic_ostream<char, std::__1::char_traits<char> > &(*)(basic_ostream<char,/
       std::__1::char_traits<char> > &, const char *)' <FunctionToPointerDecay>
      | `-DeclRefExpr 0x106ee3df0 <col:13> 'basic_ostream<char, std::__1::char_traits<char> > &(basic_ostream<char, std::/
      __1::char_traits<char> > &, const char *)' lvalue Function 0x106ee2800 'operator<<' 'basic_ostream<char, std::__1::/
      char_traits<char> > &(basic_ostream<char, std::__1::char_traits<char> > &, const char *)'
      |-DeclRefExpr 0x106ee1698 <col:3, col:8> 'std::__1::ostream':'std::__1::basic_ostream<char>' lvalue Var 0x106ee0fb8/
       'cout' 'std::__1::ostream':'std::__1::basic_ostream<char>'
      `-ImplicitCastExpr 0x106ee3dd8 <col:16> 'const char *' <ArrayToPointerDecay>
        `-StringLiteral 0x106ee16c8 <col:16> 'const char [13]' lvalue "Hello, C++!\n"
```

```cpp
#include <iostream>

int main()
{
  std::cout << "Hello, C++!\n";
}
```


The different transformation types

- Pure AST based transformations
  - implicit conversion
  - **auto** type-deduction
  - default parameters
  - **constexpr if**
- C++ Standard based AST transformations
  - range-based for loop
  - lambda
  - **noexcept**
  - main **return** zero
- Heavy hand-rolled transformations
  - Coroutines
  - Object-lifetime
  - Cfront mode

```
USAGE: insights [options] <source0> [... <sourceN>]

OPTIONS:

Generic Options:
  ...

Insights:

  --alt-syntax-for                        - Transform all for-loops into their equivalent while-loop.
  --alt-syntax-subscription               - Transform array subscriptions E1[E2] into (*(E1 + E2)).
  --show-all-callexpr-template-parameters - Show all template parameters of a CallExpr.
  --show-all-implicit-casts               - Show all implicit casts which can be noisy.
  --stdin                                 - Read the input from <stdin>.
  --use-libc++                            - Use libc++.

Insights-Educational:
This transformations are only for education purposes. The resulting code most likely does not compile.

  --edu-show-cfront                   - Show transformation to C
  --edu-show-coroutine-transformation - Show transformations of coroutines.
  --edu-show-initlist                 - Transform a std::initializer list
  --edu-show-lifetime                 - Show lifetime of objects
  --edu-show-noexcept                 - Transform a noexcept function
  --edu-show-padding                  - Show the padding bytes in a struct/class
```

Full list and examples: https://docs.cppinsights.io/command_line_options.html

Default parameters

- How does a default parameter work?

```
void Fun(int x = 23);

void Use() { Fun(); }
```


Default parameters

- How does a default parameter work?

```
void Fun(
  std::string x =
    "Hello, C++ community! I'm a default parameter.");

void Use()
{
  Fun();
  Fun();
  Fun();
}
```


https://AndreasFertig.com
post@AndreasFertig.com

Padding

```
1 struct Data {
2   int   i;
3   char  c;
4   float f;
5 };
```

Assume a 32 bit architecture.

Use the *-edu-show-padding* option to see what's happening.

Padding

```
1 struct Apple {
2   bool readyForWrite;
3   int  data[15];
4   bool haveData;
5
6   void Update(int value);
7 };
```

Assume a 32 bit architecture.

Use the *-edu-show-padding* option to see what's happening.

Object lifetime visualization

```
struct Apple {
  std::vector<std::vector<int>> v;
};

Apple Fun();

void Use()
{
  for (auto e : Fun().v) {  A
    // ...
  }

  for (auto e : Fun().v[0]) {  B
    // ...
  }
}
```

Use the *-edu-show-lifetime* option to see what's happening.

Object lifetime visualization

```
// From: https://eel.is/c++draft/class.temporary#6.11
struct S {
  int                          mi;
  const std::pair<int, int>& mp;
};

void Use()
{
  S  a{1, {2, 3}};            A
  S* p = new S{1, {2, 3}};    B
}
```

Use the *-edu-show-lifetime* option to see what's happening.

Object lifetime visualization

```
// C++20 / Source: https://wg21.link/p0960r3
struct A {
  int   a;
  int&& r;
};

int f();

void Use()
{
  int n = 10;

  A a1{1, f()};
  A a2(1, f());
  // A a3{1.0, 1}; // narrowing isn't allowed
  A a4(1.0, 1);
  A a5(1.0, std::move(n));
}
```

Use the *-edu-show-lifetime* option to see what's happening.

Reference parameters

```
void PrintInt(const int& num)
{
  std::println("The magic number is: {}", num);
  return;
}

void Use() { PrintInt(43); }
```


https://AndreasFertig.com
post@AndreasFertig.com

Reference parameters

```
void PrintInt(const Lifeguard& num)
{
  std::println("The magic number is: {}", num.val);
  return;
}

void Use() { PrintInt(43); }
```


Reference parameters

```
Task PrintInt(const Lifeguard& num)
{
  std::println("The magic number is: {}", num.val);
  co_return;
}

void Use() { PrintInt(43); }
```

Use the *-edu-show-coroutine-transformation* option to see what's happening.

https://AndreasFertig.com
post@AndreasFertig.com

C++23

The explicit object parameter

```
template<typename T, size_t SIZE>
struct Array {
  T mData[SIZE];

  const T* data() const { return std::addressof(mData[0]); }
  T*       data() { return std::addressof(mData[0]); }

  const T* begin() const { return data(); }
  const T* end() const { return data() + size(); }
  const T* begin() { return data(); }
  const T* end() { return data() + size(); }

  const T& operator[](size_t idx) const { return mData[idx]; }
  T&       operator[](size_t idx) { return mData[idx]; }

  constexpr size_t size() const { return SIZE; }
};
```


C++23

The explicit object parameter

```
template<typename T, size_t SIZE>
struct Array {
  T mData[SIZE];

  auto* data(this auto& self) { return std::addressof(self.mData[0]); }

  auto* begin(this auto& self) { return self.data(); }
  auto* end(this auto& self) { return self.data() + self.size(); }

  auto& operator[](this auto& s, size_t idx) { return s.mData[idx]; }

  constexpr size_t size() const { return SIZE; }
};
```


https://AndreasFertig.com
post@AndreasFertig.com

Back to the Basics

Back to the Basics: The Cfront mode

```
class Apple {
  int mX{};

public:
  Apple(int x) : mX{x} {}

  void Set(int x) { mX = x; }
  int  Get() const { return mX; }
};

int main()
{
  Apple a{7};
  a.Set(4);

  return a.Get();
}
```

Use the *-edu-show-cfront* option to see what's happening.

The Cfront mode: virtual functions

```
1 struct Fruit {
2   double md;
3   virtual ~Fruit() { puts("~Fruit"); }
4   virtual void Fun() { puts("Fruit's Fun"); }
5 };
6
7 struct Apple : Fruit {
8   int  mX{5};
9   void Fun() override { printf("Apple's Fun: %d\n", mX); }
10 };
11
12 struct PinkLady : Apple {
13   int  mApple{8};
14   void Fun() override { printf("Pink Ladies Fun: %d\n", mApple); }
15 };
16
17 int main()
18 {
19   PinkLady delicious{};
20   delicious.Fun();  A
21
22   Fruit* f{static_cast<Fruit*>(&delicious)};  B
23   f->Fun();
24 }
```

Use the *-edu-show-cfront* option to see what's happening.

Support the project

https://github.com/andreasfertig/cppinsights

https://github.com/sponsors/andreasfertig/

https://shop.spreadshirt.de/cppinsights

So it is completely ”fertig”? :-)

Andreas Fertig @Andreas_Fertig · Nov 4, 2018

I've made a couple of cppinsights.io updates: it can now handle more statements, more noexcept expressions and some other tweaks.
I also fixed the UI, there is new space between all buttons.

Check it out!

#cppinsights #cplusplus #cpp11 #cpp14 #cpp17 #cpp @isocpp

C++ Insights
C++ Insights - See your source code with the eyes of a compiler.
cppinsights.io

1 9 29

Nasrin
@__nasrin_

Replying to @Andreas_Fertig @SoftwebCPP and @isocpp

So it is completely “fertig”? :-)

2:35 PM · Nov 5, 2018 · Twitter for iPhone

Source: [2]

}

I am Fertig.

https://AndreasFertig.com

Available online:

https://AndreasFertig.com

Images by Franziska Panter:

https://panther-concepts.de

https://AndreasFertig.com
post@AndreasFertig.com

Used Compilers & Typography

Used Compilers

- Compilers used to compile (most of) the examples.
  - GCC 14.1.0
  - Clang 18.1.0

Typography

- Main font:
  - Camingo Dos Pro by Jan Fromm (https://janfromm.de/)
- Code font:
  - CamingoCode by Jan Fromm licensed under Creative Commons CC BY-ND, Version 3.0 http://creativecommons.org/licenses/by-nd/3.0/

References

[1] Fertig A., "How to run a local instance of c++ insights". https://andreasfertig.blog/2020/01/how-to-run-a-local-instance-of-cpp-insights/

[2] ___nasrin_ , "So it is completely "fertig"? :-)". https://twitter.com/sheIsLearningg/status/1059439178499452929

Images:

3: Franziska Panter
4: Franziska Panter
34: Franziska Panter

https://AndreasFertig.com
post@AndreasFertig.com

Upcoming Events

Talks

- *C++20's Coroutines for Beginners*, NYC++ Meetup, July 25

Training Classes

- *Modern C++: When Efficiency Matters*, CppCon, September 21 - 22

For my upcoming talks you can check https://andreasfertig.com/talks/.
For my courses you can check https://andreasfertig.com/courses/.
Like to always be informed? Subscribe to my newsletter: https://andreasfertig.com/newsletter/.

About Andreas Fertig


Photo: Kristijan Matic www.kristijanmatic.de

Andreas Fertig, CEO of Unique Code GmbH, is an experienced trainer and consultant for C++ for standards 11 to 23.
Andreas is involved in the C++ standardization committee, developing the new standards. At international conferences, he presents how code can be written better. He publishes specialist articles, e.g., for iX magazine, and has published several textbooks on C++.
With C++ Insights (https://cppinsights.io), Andreas has created an internationally recognized tool that enables users to look behind the scenes of C++ and thus understand constructs even better.
Before training and consulting, he worked for Philips Medizin Systeme GmbH for ten years as a C++ software developer and architect focusing on embedded systems.
You can find Andreas online at andreasfertig.com.

https://AndreasFertig.com
post@AndreasFertig.com