C++ Insights

How stuff works, Lambdas and more!

Andreas Fertig
https://AndreasFertig.Info
post@AndreasFertig.Info
@Andreas__Fertig

fertig

adjective /ˈfɛrtɪç/

finished
ready
complete
completed

Motivation

```
MyType i{};
i++;
```


Motivation

```
MyType i{};
i.operator++(0);
```

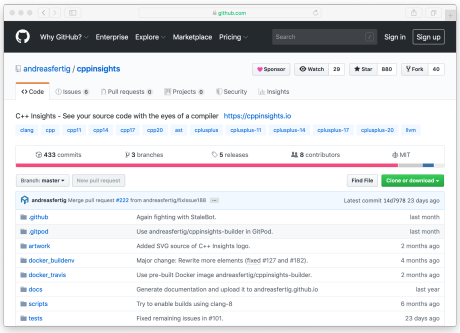
Implicit Conversions

```
1
2  short int max(short int a, short int b)
3  {
4    return (a > b) ? a : b;
5  }
6
7  void Main()
8  {
9    short int          a = 1;
10   unsigned short int b = 65'530;
11
12   printf("max: %d\n", max(a, b));
13 }
```

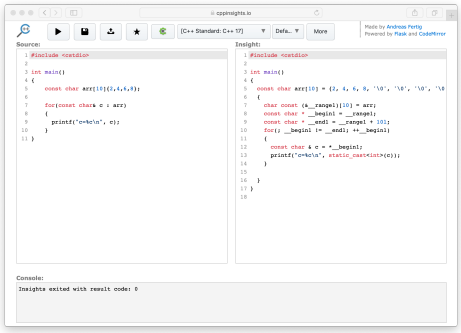

C++ Insights

- 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.*

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



`https://cppinsights.io`



A word about limitations

- C++ Insights is a Clang based tool.
- 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.
- Not *all* statements are currently matched.

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 }
```

https://AndreasFertig.Info
post@AndreasFertig.Info

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"
```

Andreas Fertig
v1.0

C++ Insights

9

What is an AST

```
1 #include <iostream>
2
3 int main()
4 {
5   std::cout << "Hello, C++!\n";
6 }
```

Andreas Fertig
v1.0

C++ Insights

10

Default Argument

- How does a default argument take effect?

```
1 void Func(int x = 23) {}
2
3 int main()
4 {
5   Func();
6 }
```

Initialization

```
1 int main()
2 {
3   char a[5];
4   char b[5]{};
5   char c[5]{0};
6   char d[5]{77};
7 }
```

https://AndreasFertig.Info
post@AndreasFertig.Info

In-Class Member Initialization

```
1  class Init
2  {
3  public:
4    Init()
5    : i{9}
6    {
7    }
8
9    int              i{0};
10   std::vector<int> v{1, 2, 3};
11   std::string      s{"Hello"};
12 };
```

Lambda Internals

```
1 int main()
2 {
3   const char hello[]{"Hello, MUC++!"};
4
5   [&] { printf("%s\n", hello); }();
6 }
```

https://AndreasFertig.Info
post@AndreasFertig.Info

Generic Lambda

C++14

- Have a call operator which is a operator template with return type `auto`.
- The `auto` parameters are template parameters.

```
1 auto l = []( auto  v) { return v * 2; };
2
3 auto d = l(2.0);
4 auto i = l(2);
```


Templated Lambdas

C++20

```
1 int main()
2 {
3   auto max = [](auto x, auto y) {
4     return (x > y) ? x : y;
5   };
6
7   max(2, 3);    // ok
8   max(2, 3.0);  // not wanted
9 }
```

Templated Lambdas

C++20

```
1  int main()
2  {
3    auto max = []<typename T>(T x, T y)
4    {
5      return (x > y) ? x : y;
6    };
7
8    max(2, 3);  // ok
9    // max(2, 3.0);  // does not compile anymore
10 }
```

constexpr Lambdas

C++17

```
1  int main()
2  {
3    auto x = [] {};
4  }
```

https://AndreasFertig.Info
post@AndreasFertig.Info

Getting the type right

```
1  int main()
2  {
3    map<int, char> m{};
4
5    for(const pair<int, char>& t : m) {
6    }
7
8    for(const pair<const int, char>& t : m) {
9    }
10
11   for(const auto& t : m) {
12   }
13
14   for(const auto& [k, v] : m) {
15   }
16 }
```

Source: [1]

Range-based for statements with temporary

```
1  struct Keeper
2  {
3    std::vector<int> data{1, 2, 3};
4
5    auto& items() { return data; }
6  };
7
8  Keeper get()
9  {
10   return {};
11 }
12
13 int main()
14 {
15   for(auto& item : get().items()) {
16     std::cout << item << '\n';
17   }
18 }
```

Range-based for statements with initializer

C++20

```
1  struct Keeper
2  {
3    std::vector<int> data{1, 2, 3};
4
5    auto& items() { return data; }
6  };
7
8  Keeper get()
9  {
10   return {};
11 }
12
13 int main()
14 {
15   for(auto&& items = get();
16       auto&  item : items.items()) {
17     std::cout << item << '\n';
18   }
19 }
```

Range-based for statements with initializer

C++20

```
1  #include <cstdio>
2  #include <vector>
3
4  int main()
5  {
6    std::vector<int> v{2, 3, 4, 5, 6};
7
8    for(size_t idx{0}; const auto& e : v) {
9      printf("[%ld] %d\n", idx++, e);
10   }
11 }
```

C++17

auto as non-type template parameter

- We can have auto as a non-type template parameter in C++17

```
1  #include <iostream>
2
3  template<auto sep = ' ', typename T, typename... Ts>
4  void Print(const T& targ, const Ts&... args)
5  {
6    std::cout << targ;
7
8    if constexpr(sizeof...(args) > 0) {
9      std::cout << sep;
10     Print<sep>(args...);
11   }
12 }
13
14 int main()
15 {
16   Print("Hello", "C++", 20);
17 }
```

Source: [2]

C++17

auto as non-type template parameter

- We can have auto as a non-type template parameter in C++17

```
1  #include <iostream>
2
3  template<auto sep = ' ', typename T, typename... Ts>
4  void Print(const T& targ, const Ts&... args)
5  {
6    std::cout << targ;
7    auto coutSpaceAndArg = [](const auto& arg) {
8      std::cout << sep << arg;
9    };
10
11   (..., coutSpaceAndArg(args));
12 }
13
14 int main()
15 {
16   Print("Hello", "C++", 20);
17 }
```

Source: [2]

<=>

C++20

- With C++20 there will be spaceships in C++.
- The promise: You have to write only one comparison function (`operator<=>` and the compiler generates all the others (`<`, `>`, `<=`, `>=`, `==`, `!=`).
- The question: How does this work?

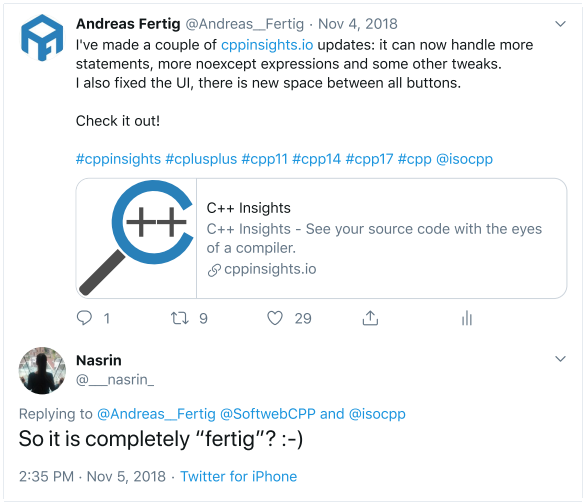
Support the project


https://github.com/andreasfertig/cppinsights


https://www.patreon.com/cppinsights


https://shop.spreadshirt.de/cppinsights

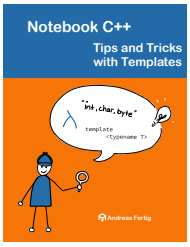
So it is completely ”fertig”? :-)



Source: [3]

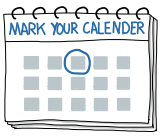
}

I am Fertig.



http://bit.ly/notebookcpp-oop

Discounted version for you!



−15% für ”Programmieren mit C++11 bis C++17” mit **OOP20**

18. - 20.3.2020 Ludwigsburg

http://bit.ly/programmieren-mit-cpp11-17

Valid until 12.02.2020

https://AndreasFertig.Info

Used Compilers

- Compilers used to compile (most of) the examples.
  - g++ (Homebrew GCC 9.2.0_1) 9.2.0
  - clang version 9.0.0 (https://github.com/llvm/llvm-project.git 0399d5a9682b3cef71c653373e38890c63c4c365)

References

[1] Josuttis N., “When C++ auto is better”. https://twitter.com/nicojosuttis/status/1132620873054261256?s=21

[2] Josuttis N., *C++17 - The Complete Guide*. leanpub, 2019.

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

Images:

28: Franziska Panter

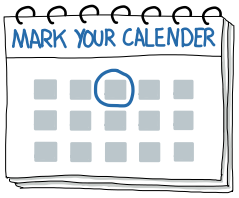
31: Franziska Panter

Upcoming Events

Training Classes

- *Programmieren mit C++11 bis C++17*, Andreas Fertig, March 18, 2020
- *C++1x für eingebettete Systeme*, QA Systems, October 15, 2020

For my upcoming talks check my *Talks* (`https://andreasfertig.info/talks/`) page.
For my training services you can check `https://andreasfertig.info/training/`.



About Andreas Fertig

Andreas is an independent trainer and consultant for C++ specializing in embedded systems. Since his computer science studies in Karlsruhe, he has dealt with embedded systems and the associated requirements and peculiarities. He worked for about 10 years for Philips Medizin Systeme GmbH as a C++ software developer and architect with focus on embedded systems.
Andreas is involved in the C++ standardization committee, especially in SG14 which deals with embedded systems.
He also develops macOS applications and is the creator of cppinsights.io.