C++ Insights

Sehen Sie Ihren Quellcode mit den Augen eines Compilers

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

Motivation

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

Motivation

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

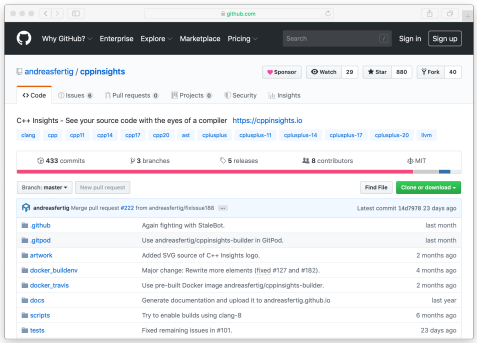

Implizite Konvertierungen

```
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 }
```

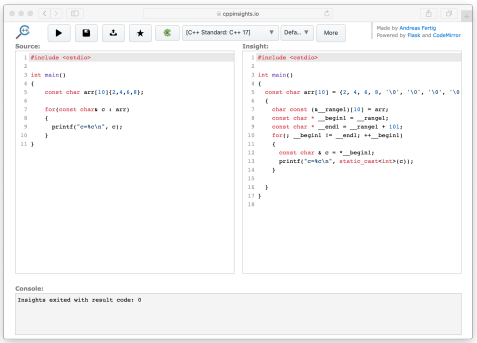
C++ Insights

- Zeigen was der Compiler verändert.
- Unsichtbare Dinge sichtbar machen hilft bei Schulungen.
- Generieren von validem Code.
- Generieren von Code der compiliert.
- *Selbstverständlich ist C++ Insights Open-Source.*

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



https://cppinsights.io



Ein Wort zu den Limitierungen

- C++ Insights ist ein Clang basiertes Tool.
- Die offiziellen Builds verwenden die neueste releaste Clang Version.
  - Dardurch sind nicht immer die neuesten Features verfügbar.
- C++ Insights nutzen den AST von Clang, welcher keine Optimierungen zeigt.
  - Die Verwendung der Option –O n hat deshalb keine Auswirkung.
- Nicht alle Ausdrücke sind aktuell unterstützt.

Ein Wort zu den Limitierungen: Templates

- Das Generieren von Code der von Templates kommt ist schwer.
- Um das ein wenig einfacher zu gestalten gibt es `#ifdef INSIGHTS_USE_TEMPLATE`.

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

Was ist ein 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"
```

Was ist ein AST

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


Default Parameter

- Wie funktioniert ein Default-Parameter?

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


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

Initialisierung

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

Default Member Initialisierung

```
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 Interna

```
1 int main()
2 {
3   const char hello[]{”Hallo, ESE Kongress”};
4
5   [&] { printf(”%s\n”, hello); }();
6 }
```


Generische Lambdas

C++14

- Verfügen über einen Call-Operator der ein Template ist, mit einem `auto` Rückgabetyp.
- Alle `auto`-Parameter sind Template-Parameter.

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

Template 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 }
```

Template 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 }
```

Range-based for-Schleife mit temporärem Objekt

```
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 }
```


C++20

Range-based for-Schleife mit Initialisierung

```
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 }
```

Das Projekt unterstützen

https://github.com/andreasfertig/cppinsights

https://www.patreon.com/cppinsights

https://shop.spreadshirt.de/cppinsights

So it is completely fertig?

Quelle: [1]

}

Ich bin Fertig.

https://AndreasFertig.Info

Available online:

https://AndreasFertig.Info

Images by Franziska Panter:

https://panther-concepts.de

Verwendete Compiler & Typografie

Verwendete Compiler

- Compiler welche zum Übersetzen des (meisten) Codes verwendet wurden.
  - g++ 10.2.0
  - clang version 10.0.0 (https://github.com/llvm/llvm-project.git d32170dbd5b0d54436537b6b75beaf44324e0c28)

Typografie

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

Quellen

[1] ___NASRIN_ , “So it is completely “fertig”? :-)”. https://twitter.com/shelsLearningg/status/1059439178499452929

Bilder:

24: Franziska Panter

Nächste Events

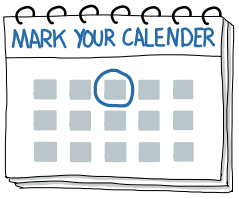
Talks

- *C++20 Templates – die nächste Generation: Concepts*, betterCode(C++20), January 21, 2021

Training Classes

- *Programmieren mit C++11 bis C++17*, Andreas Fertig, February 22, 2021
- *C++ Clean Code – Best Practices für Programmierer*, golem Akademie, March 08, 2021

Zukünftige Vorträge: https://andreasfertig.info/talks/.
Mehr zu meinen Trainingsangeboten gibt es hier: https://andreasfertig.info/training/.
Immer aktuell Informiert? Hier geht es zum Newsletter: https://andreasfertig.info/newsletter/.



Über Andreas Fertig

Foto: Kristijan Matic www.kristijanmatic.de

Andreas Fertig ist Geschäftsführer der Unique Code GmbH, die Schulungen und Beratung für C++ anbietet mit dem Spezialgebiet eingebettete Systeme. Er arbeitete zehn Jahre für die Philips Medizin Systeme GmbH als C++ Softwareentwickler und Architekt mit Schwerpunkt auf eingebetteten Systemen.
Andreas engagiert sich im C++ Standardisierungskomitee. Als Referent ist er regelmäßig international auf Konferenzen anzutreffen. Fachbücher sowie Fachartikel von Andreas gibt es in Deutsch und Englisch.
Andreas hat eine Leidenschaft dafür, Menschen beizubringen, wie C++ funktioniert, weshalb er C++ Insights (cppinsights.io) geschaffen hat.