Implementing the C++ Core Guidelines' Lifetime Safety Profile in Clang

Matthias Gehre

SILEXICA

Gábor Horváth

E·L·T·E

Outline

- ❏ The paper versus the clang-based implementation
- ❏ Intra-function analysis
- ❏ Inter-function analysis
- ❏ Status and outlook

Paper vs. Implementation

Herb Sutter’s Paper

- ❏ Available on C++ Core Guidelines repo [1]
- ❏ Intends to catch common lifetime errors (not all!)
- ❏ Watch his plenary talk

[1] https://github.com/isocpp/CppCoreGuidelines/blob/master/docs/Lifetime.pdf
[2] https://herbsutter.com/2018/09/20/lifetime-profile-v1-0-posted

The clang-based implementation (aka -Wlifetime)

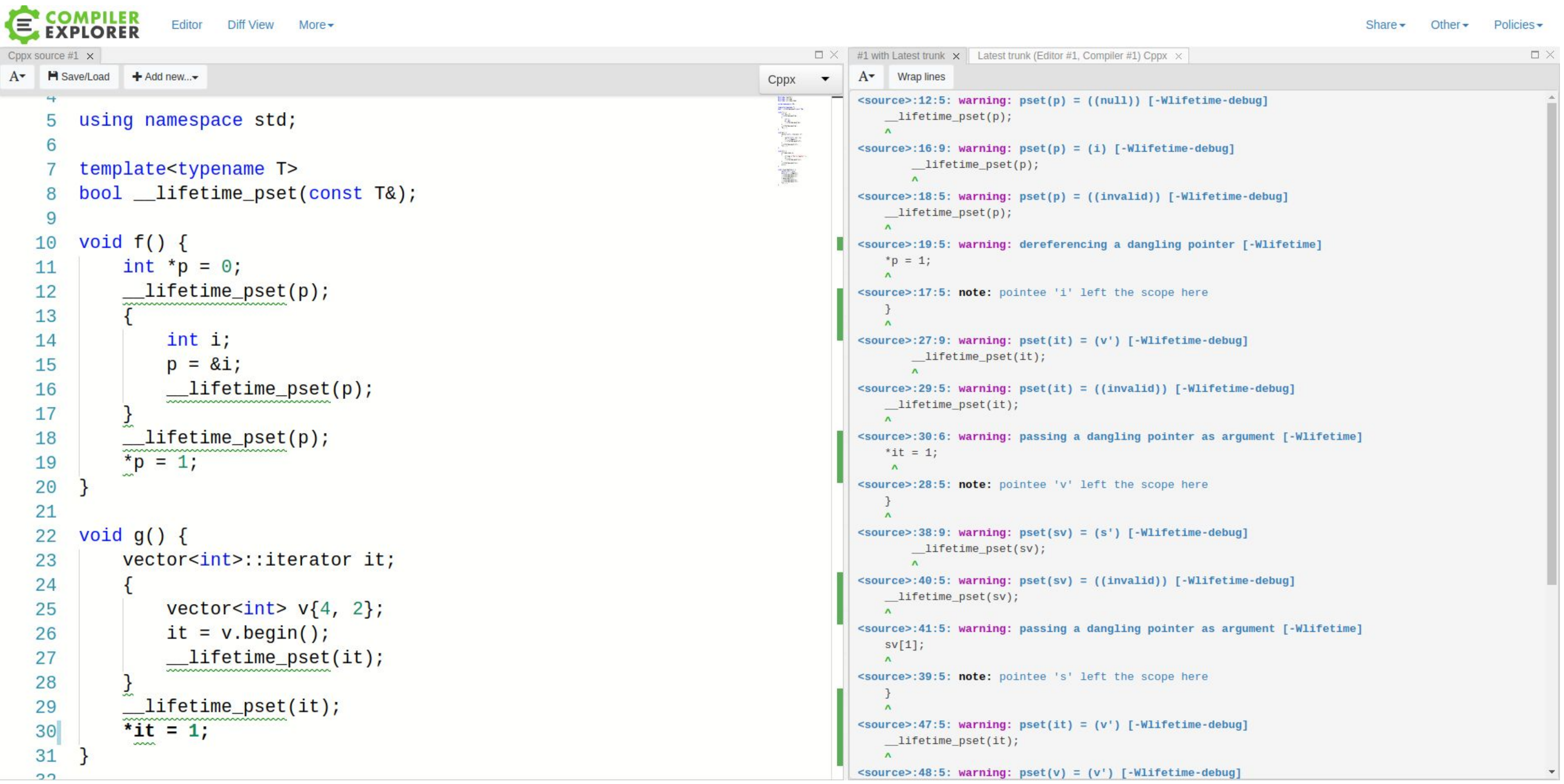
- ❏ Alpha version
- ❏ Some examples don’t give expected results (yet)
- ❏ We’ll point out if this is due to the implementation or due to the approach itself

Note: There is also a MSVC-based implementation (not in the scope of this talk)

Intra-Function Analysis

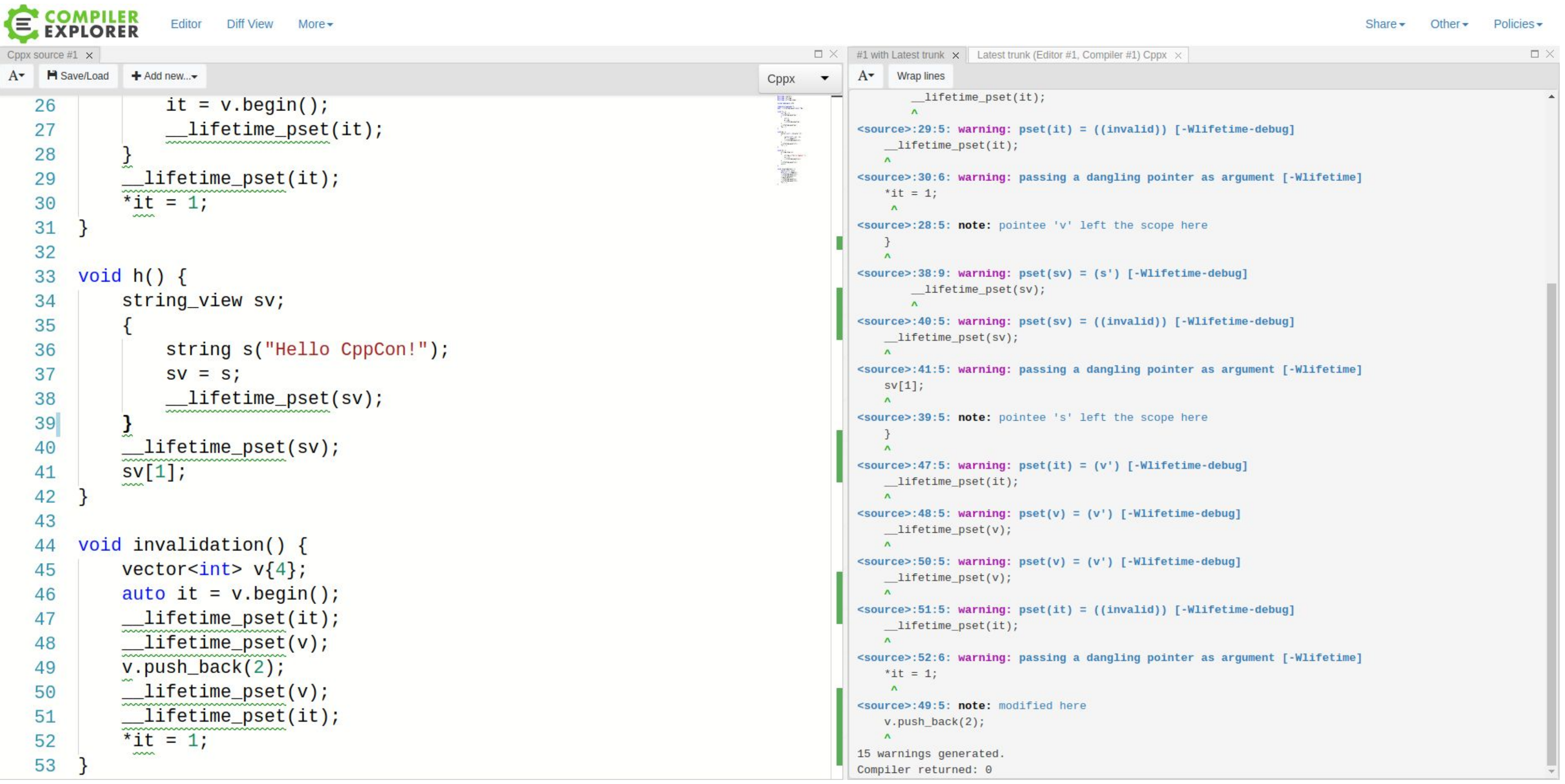
Examples

- https://cppx.godbolt.org/z/IdGRsT



```
 5  using namespace std;
 6
 7  template<typename T>
 8  bool __lifetime_pset(const T&);
 9
10  void f() {
11      int *p = 0;
12      __lifetime_pset(p);
13      {
14          int i;
15          p = &i;
16          __lifetime_pset(p);
17      }
18      __lifetime_pset(p);
19      *p = 1;
20  }
21
22  void g() {
23      vector<int>::iterator it;
24      {
25          vector<int> v{4, 2};
26          it = v.begin();
27          __lifetime_pset(it);
28      }
29      __lifetime_pset(it);
30      *it = 1;
31  }
```

```
<source>:12:5: warning: pset(p) = ((null)) [-Wlifetime-debug]
    __lifetime_pset(p);
    ^
<source>:16:9: warning: pset(p) = (i) [-Wlifetime-debug]
        __lifetime_pset(p);
        ^
<source>:18:5: warning: pset(p) = ((invalid)) [-Wlifetime-debug]
    __lifetime_pset(p);
    ^
<source>:19:5: warning: dereferencing a dangling pointer [-Wlifetime]
    *p = 1;
    ^
<source>:17:5: note: pointee 'i' left the scope here
    }
    ^
<source>:27:9: warning: pset(it) = (v') [-Wlifetime-debug]
        __lifetime_pset(it);
        ^
<source>:29:5: warning: pset(it) = ((invalid)) [-Wlifetime-debug]
    __lifetime_pset(it);
    ^
<source>:30:6: warning: passing a dangling pointer as argument [-Wlifetime]
    *it = 1;
     ^
<source>:28:5: note: pointee 'v' left the scope here
    }
    ^
<source>:38:9: warning: pset(sv) = (s') [-Wlifetime-debug]
        __lifetime_pset(sv);
        ^
<source>:40:5: warning: pset(sv) = ((invalid)) [-Wlifetime-debug]
    __lifetime_pset(sv);
    ^
<source>:41:5: warning: passing a dangling pointer as argument [-Wlifetime]
    sv[1];
    ^
<source>:39:5: note: pointee 's' left the scope here
    }
    ^
<source>:47:5: warning: pset(it) = (v') [-Wlifetime-debug]
    __lifetime_pset(it);
    ^
<source>:48:5: warning: pset(v) = (v') [-Wlifetime-debug]
```



```
26         it = v.begin();
27         __lifetime_pset(it);
28     }
29     __lifetime_pset(it);
30     *it = 1;
31 }
32
33 void h() {
34     string_view sv;
35     {
36         string s("Hello CppCon!");
37         sv = s;
38         __lifetime_pset(sv);
39     }
40     __lifetime_pset(sv);
41     sv[1];
42 }
43
44 void invalidation() {
45     vector<int> v{4};
46     auto it = v.begin();
47     __lifetime_pset(it);
48     __lifetime_pset(v);
49     v.push_back(2);
50     __lifetime_pset(v);
51     __lifetime_pset(it);
52     *it = 1;
53 }
```

```
        __lifetime_pset(it);
        ^
<source>:29:5: warning: pset(it) = ((invalid)) [-Wlifetime-debug]
    __lifetime_pset(it);
    ^
<source>:30:6: warning: passing a dangling pointer as argument [-Wlifetime]
    *it = 1;
     ^
<source>:28:5: note: pointee 'v' left the scope here
    }
    ^
<source>:38:9: warning: pset(sv) = (s') [-Wlifetime-debug]
        __lifetime_pset(sv);
        ^
<source>:40:5: warning: pset(sv) = ((invalid)) [-Wlifetime-debug]
    __lifetime_pset(sv);
    ^
<source>:41:5: warning: passing a dangling pointer as argument [-Wlifetime]
    sv[1];
    ^
<source>:39:5: note: pointee 's' left the scope here
    }
    ^
<source>:47:5: warning: pset(it) = (v') [-Wlifetime-debug]
    __lifetime_pset(it);
    ^
<source>:48:5: warning: pset(v) = (v') [-Wlifetime-debug]
    __lifetime_pset(v);
    ^
<source>:50:5: warning: pset(v) = (v') [-Wlifetime-debug]
    __lifetime_pset(v);
    ^
<source>:51:5: warning: pset(it) = ((invalid)) [-Wlifetime-debug]
    __lifetime_pset(it);
    ^
<source>:52:6: warning: passing a dangling pointer as argument [-Wlifetime]
    *it = 1;
     ^
<source>:49:5: note: modified here
    v.push_back(2);
    ^
15 warnings generated.
Compiler returned: 0
```

Debugging (add -Wlifetime-debug)

```
// Diagnoses the points-to set of the argument
template <typename T>
bool __lifetime_pset(const T &) {}

// Diagnoses the location of the argument
template <typename T>
bool __lifetime_pset_ref(const T &) {}

// Diagnoses the type category of the template
argument
template <typename T>
void __lifetime_type_category() {}
```

Example: https://cppx.godbolt.org/z/ZGMBIw

Approach

- ❏ Classify types into categories:
  - ❏ **Owners** (vector, unique_ptr)
  - ❏ **Pointers** (int*, double&, std::reference_wrapper, any iterator, string_view)
  - ❏ **Aggregates** (similar concept to PODs)
  - ❏ **Values** (anything else)

Type Categories - Owners

- Own their memory, should never dangle
- Always point to their owned memory
- Ownership might be transferred (move)
- May be invalidated
- Some may be null (unique_ptr)
- Assumed to be correct
  - Rust does the same: owners use unsafe
- Need to know the owned type for the call model
  - Limitations with variants

- Containers, smart pointers

Type Categories - Pointers

- Do not own memory, might dangle
- Can be (generalized) null
- Might point into owners (and dangle)
- **Track** points-to sets
- Need to know the pointee type for the call model

- Pointers, references, iterators, string_view

Type Categories - Aggregates & Values

- Aggregates are similar to PODs
- No user-written copy, move, destruct operations
- We handle them memberwise in all operations

- Values are everything that did not fit into the first 3 categories

- Encapsulation is respected

Points-to map and Points-to set

- Each function is analyzed separately, walking CFG
- pmap: Maps variables to their psets
- pset: Set of what a Pointer may currently point to:
  - invalid
  - null
  - static (e.g. globals or unknown)
  - any local variable/parameter/Aggregate member[1]
  - into an Owner

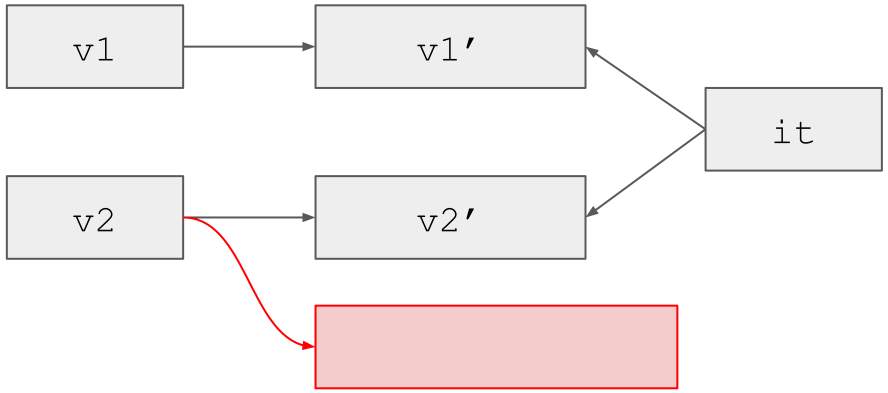
[1] Aggregates support is not yet implemented

Branching

```
int* p; // pset(p) = {(invalid)}
if (cond) {
  p = &i;       // pset(p) = {i}
} else {
  p = nullptr; // pset(p) = {(null)}
}
// pset(p) = {i, (null)}
```

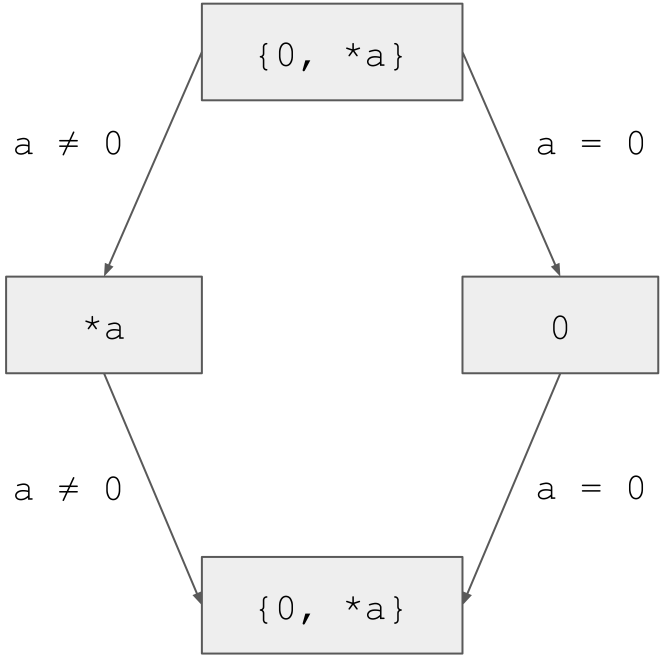
Vector example

```
vector v1{…};
vector v2{…};
auto it = v1.begin();
if (cond)
   it = v2.begin();
v2.push_back(…);
*it = …; // warning
```



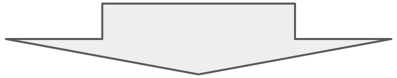
Null Tracking

```
void f(int* a) {
  // pset(a) = {(null), *a)
  if (a) {
    // pset(a) = {*a}
  } else {
    // pset(a) = {(null)}
  }
  // pset(a) = {(null), *a)
}
```

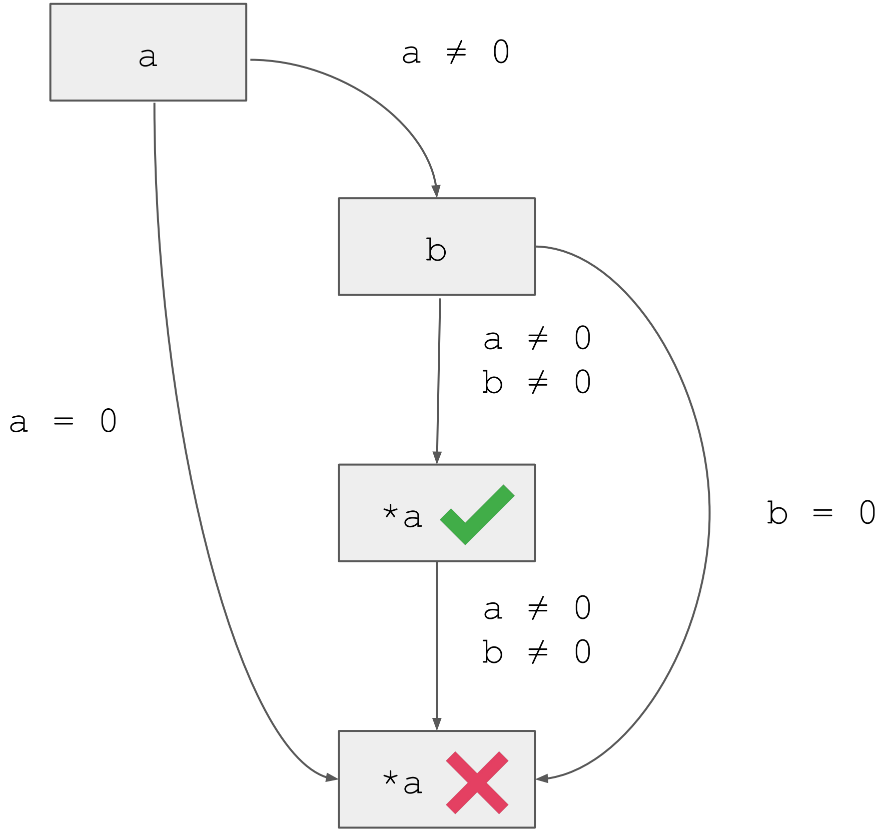


Null Tracking

```
if (a && b) {
    *a;
}
*a;
```



```
if (a) {
    if (b) {
        *a; // OK
    }
}
*a; // warning
```



Inter-Function Analysis

```
void f() {
  int i = 17;
  auto& r = foo(i, i-1);
  [...]
}
```

```
void f() {
  int i = 17;
  auto& r = foo(i, i-1);
  // r is valid
}
```

```
int& foo(int& a, int b) {
   a += b;
   return a;
}
```

```cpp
void f() {
  int i = 17;
  auto& r = foo(i, i-1);
  // r is dangling!
}
```

```cpp
const int& foo(const int& a,
               const int& b) {
  return (a < b) ? a : b;
}
```

Detour: Type safety

main.c

```
double sqrt();

int main() {

   // runtime error
   sqrt(2);

   // runtime error
   sqrt("2.1");

}
```

sqrt.c

```
double sqrt(x)

    double x; {

   [...]

}
```

Detour: Type safety

main.cpp

```
double sqrt(double x);

int main() {

   // implicit cast
   sqrt(2);

   // compile-time error
   sqrt("2.1");

}
```

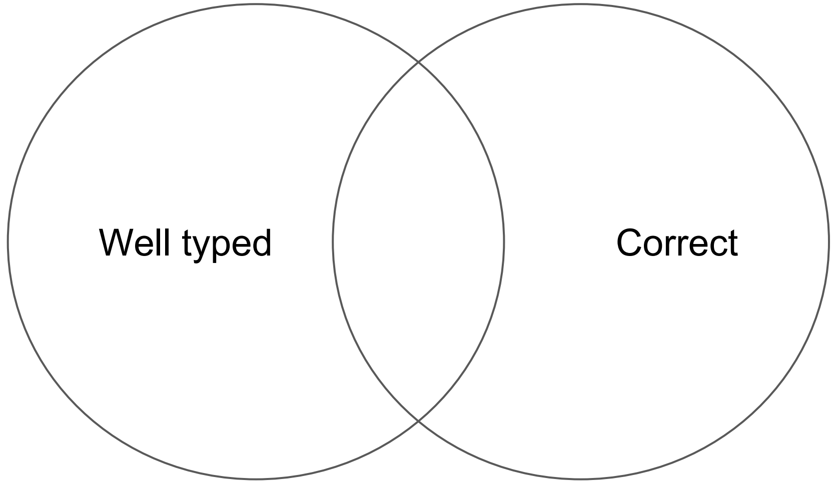
sqrt.cpp

```
double sqrt(double x) {

      [...]

      // compile-time error
      return "2.1";

}
```

How can the compiler detect the error?
Interfaces

Detour: Type safety

- Type system have false pos./true neg.
  - Not all correct programs are well typed
  - Not all well typed programs are correct
- Well typed programs are less likely to contain errors
- Not well typed programs can be rewritten to be well typed



So we need lifetime annotations everywhere?

```
void f() {

  int i = 17;
  auto& r = foo(i, i-1);

}
```

```
[[gsl::lifetime(a)]]
int& foo(int& a, int b) {
   a += b;
   return a;
}

[[gsl::lifetime(a, b)]]
const int& foo(const int& a,
               const int& b) {
   return (a < b) ? a : b;
}
```

So we need lifetime annotations everywhere?

```
void f() {

    in
    au

}
```

```
[[gsl::lifetime(a)]]
                    {



                    t& a,
                    t& b) {
                    : b;
}
```

Hopefully not!

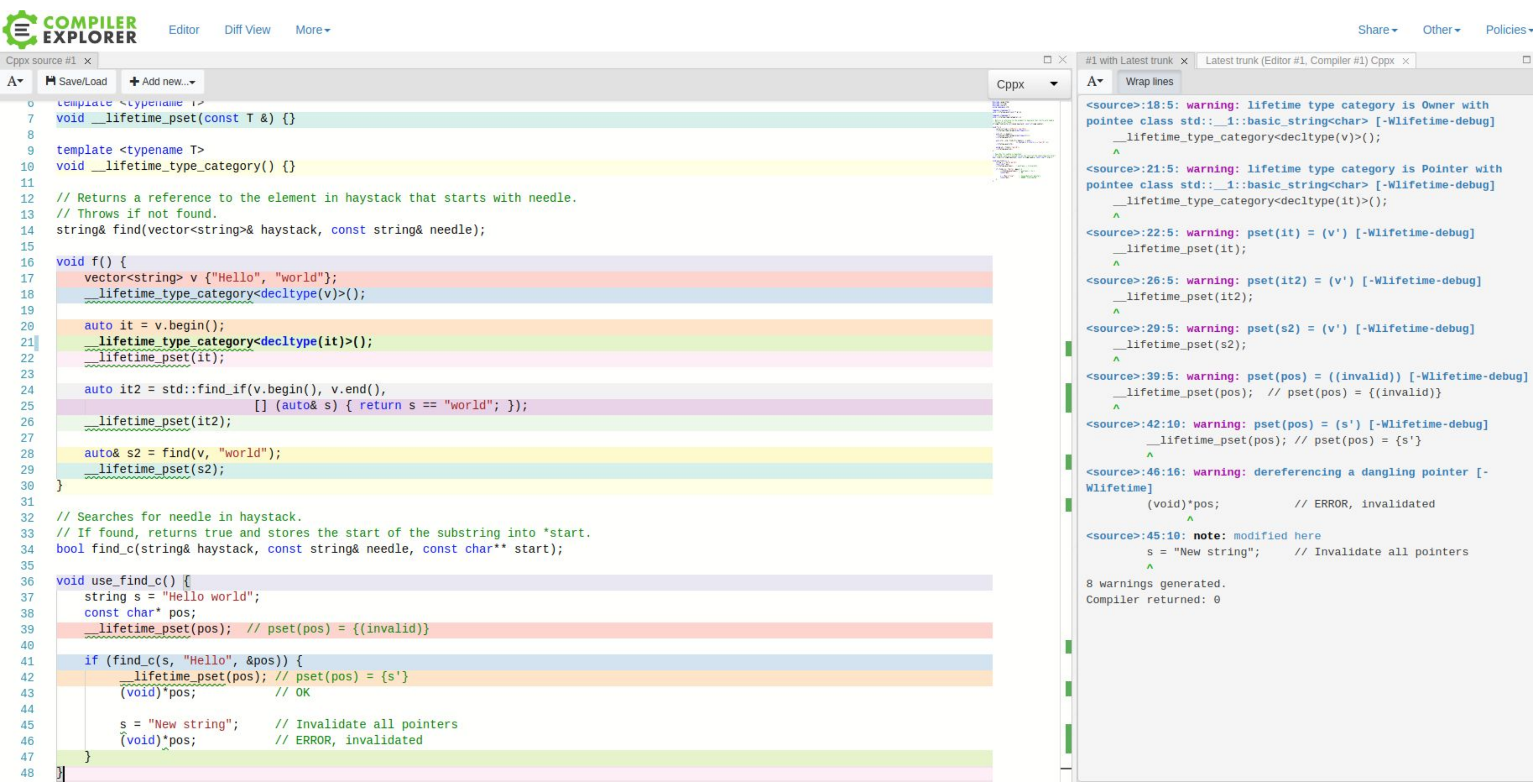
Heuristics - Survey

```
struct S {
  void f1(int* p);
  void f2(S** p);
  void f3(string_view& p);
  static S& f4();
  [...]
};
const char* f5(string& s, const string& t);
```

Heuristics - Survey # 2

```
const char* f5(string& s, const string& t);

int main() {
    string str;
    const char* c = f5(str, "hello");
};
```

Parameter (example)	Before call	After call
Value (int)	valid[1]	-
Pointer (int*, string_view) Input	valid or null	-
Pointer to Pointer (S**)		
❑ Top level Input	valid or null	-
❑ Deref Output	don’t care	valid or null
Reference to Pointer (string_view&)		
❑ Top level Input	valid	-
❑ Deref In/Out	valid or null	valid or null (different)



```cpp
 6  template <typename T>
 7  void __lifetime_pset(const T &) {}
 8
 9  template <typename T>
10  void __lifetime_type_category() {}
11
12  // Returns a reference to the element in haystack that starts with needle.
13  // Throws if not found.
14  string& find(vector<string>& haystack, const string& needle);
15
16  void f() {
17      vector<string> v {"Hello", "world"};
18      __lifetime_type_category<decltype(v)>();
19
20      auto it = v.begin();
21      __lifetime_type_category<decltype(it)>();
22      __lifetime_pset(it);
23
24      auto it2 = std::find_if(v.begin(), v.end(),
25                              [] (auto& s) { return s == "world"; });
26      __lifetime_pset(it2);
27
28      auto& s2 = find(v, "world");
29      __lifetime_pset(s2);
30  }
31
32  // Searches for needle in haystack.
33  // If found, returns true and stores the start of the substring into *start.
34  bool find_c(string& haystack, const string& needle, const char** start);
35
36  void use_find_c() {
37      string s = "Hello world";
38      const char* pos;
39      __lifetime_pset(pos);  // pset(pos) = {(invalid)}
40
41      if (find_c(s, "Hello", &pos)) {
42          __lifetime_pset(pos); // pset(pos) = {s'}
43          (void)*pos;           // OK
44
45          s = "New string";     // Invalidate all pointers
46          (void)*pos;           // ERROR, invalidated
47      }
48  }
```

```
<source>:18:5: warning: lifetime type category is Owner with pointee class std::__1::basic_string<char> [-Wlifetime-debug]
    __lifetime_type_category<decltype(v)>();
    ^
<source>:21:5: warning: lifetime type category is Pointer with pointee class std::__1::basic_string<char> [-Wlifetime-debug]
    __lifetime_type_category<decltype(it)>();
    ^
<source>:22:5: warning: pset(it) = (v') [-Wlifetime-debug]
    __lifetime_pset(it);
    ^
<source>:26:5: warning: pset(it2) = (v') [-Wlifetime-debug]
    __lifetime_pset(it2);
    ^
<source>:29:5: warning: pset(s2) = (v') [-Wlifetime-debug]
    __lifetime_pset(s2);
    ^
<source>:39:5: warning: pset(pos) = ((invalid)) [-Wlifetime-debug]
    __lifetime_pset(pos);  // pset(pos) = {(invalid)}
    ^
<source>:42:10: warning: pset(pos) = (s') [-Wlifetime-debug]
        __lifetime_pset(pos); // pset(pos) = {s'}
        ^
<source>:46:16: warning: dereferencing a dangling pointer [-Wlifetime]
        (void)*pos;           // ERROR, invalidated
              ^
<source>:45:10: note: modified here
        s = "New string";     // Invalidate all pointers
        ^
8 warnings generated.
Compiler returned: 0
```

https://godbolt.org/z/02hrWO

Status and Outlook

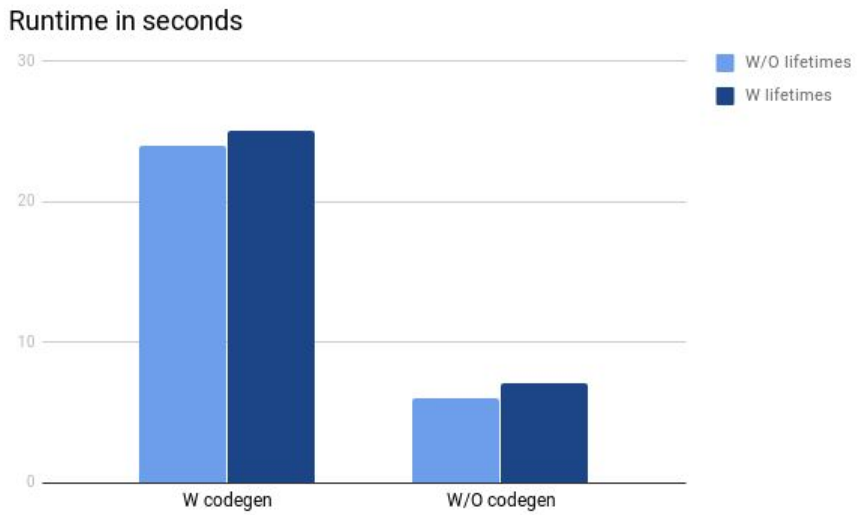
Performance measurements

Measured on: i7-5700HQ CPU, 16GB RAM, HDD

Compiling: SemaDeclCXX.cpp (15k lines, one of the bigger translation units in Clang)

Running: Release build without assertions, lifetime patches included

Caveat: Emitting of actual warnings is turned off



Status and Roadmap

What is in the implementation:

- ❏ Intra-function analysis
- ❏ Inter-function analysis

What is in the paper, but not (yet) implemented:

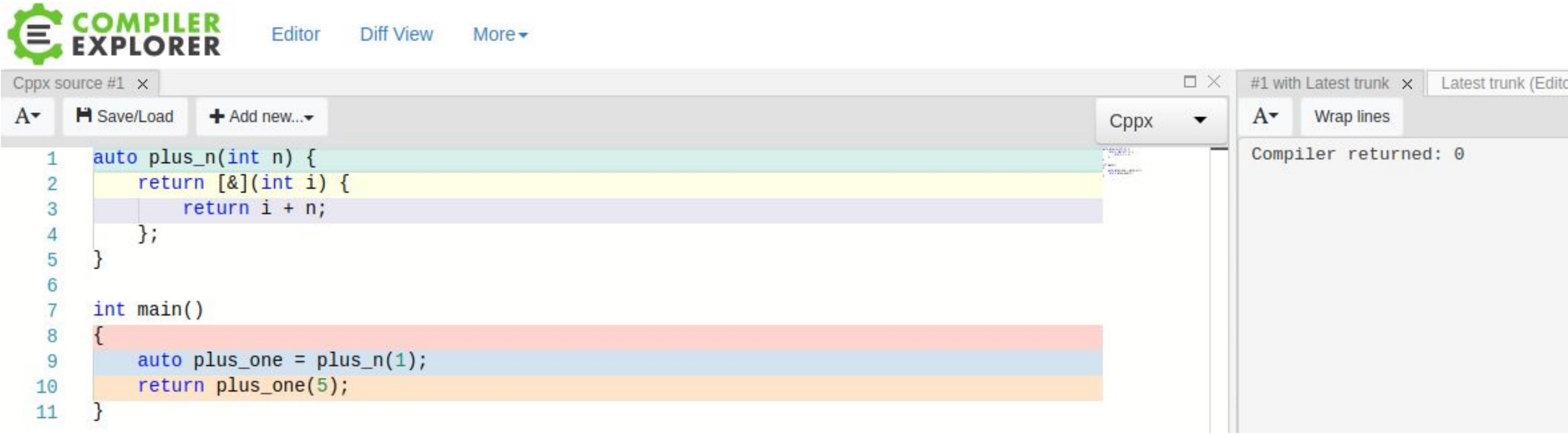
- ❏ Aggregates, exceptions, use-after-move detection
- ❏ [[gsl::lifetime]] and [[gsl::lifetime_out]] annotations
- ❏ Upstreaming

https://github.com/mgehre/clang/issues

Not (yet) working (Reddit and co.)

- Returning a lambda that captures local by reference (https://godbolt.org/z/UFR9AG)
- Data dependence (https://godbolt.org/z/_midIP)
- std::array (https://godbolt.org/z/dkpkGC)
- Jason Turner’s talk (https://cppx.godbolt.org/z/5Uw3co)
- Robert O'Callahan (https://godbolt.org/z/MTqoz9)

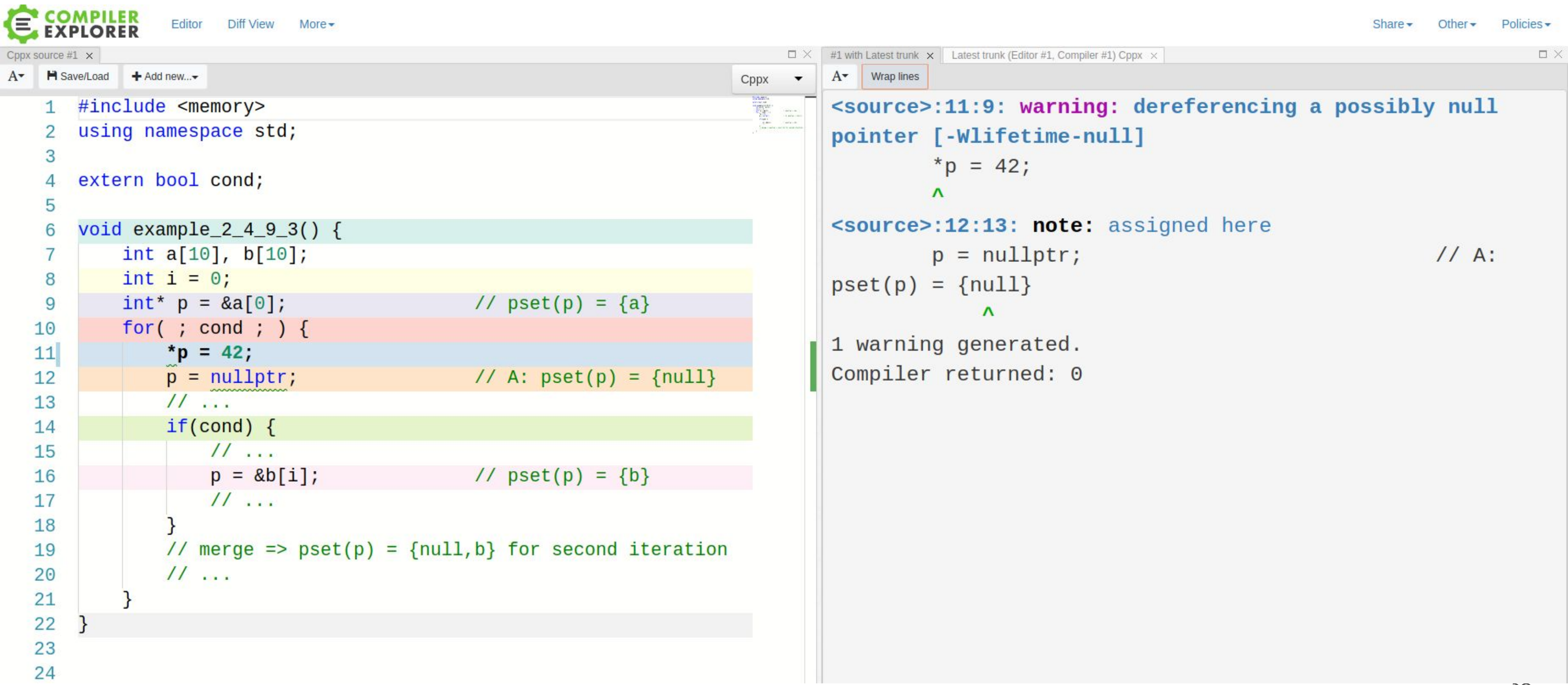
Returning a lambda that captures local by reference (https://godbolt.org/z/UFR9AG)



```cpp
auto plus_n(int n) {
    return [&](int i) {
        return i + n;
    };
}

int main()
{
    auto plus_one = plus_n(1);
    return plus_one(5);
}
```

```
Compiler returned: 0
```

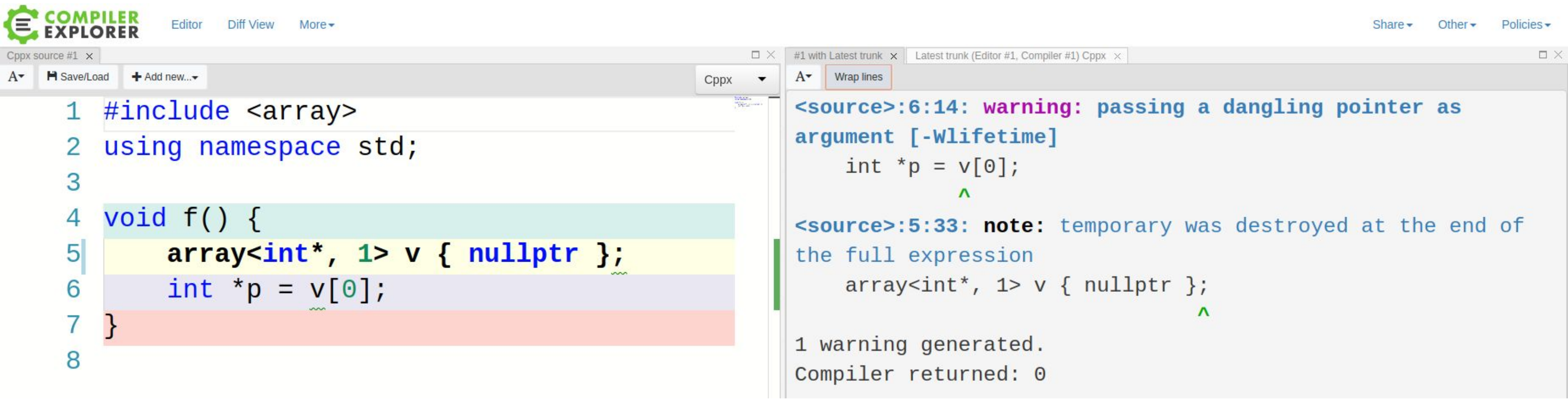
Data dependence (https://godbolt.org/z/_midIP)



```
 1  #include <memory>
 2  using namespace std;
 3
 4  extern bool cond;
 5
 6  void example_2_4_9_3() {
 7      int a[10], b[10];
 8      int i = 0;
 9      int* p = &a[0];                 // pset(p) = {a}
10      for( ; cond ; ) {
11          *p = 42;
12          p = nullptr;                // A: pset(p) = {null}
13          // ...
14          if(cond) {
15              // ...
16              p = &b[i];              // pset(p) = {b}
17              // ...
18          }
19          // merge => pset(p) = {null,b} for second iteration
20          // ...
21      }
22  }
23
24
```

```
<source>:11:9: warning: dereferencing a possibly null
pointer [-Wlifetime-null]
        *p = 42;
        ^
<source>:12:13: note: assigned here
        p = nullptr;                // A:
pset(p) = {null}
            ^
1 warning generated.
Compiler returned: 0
```

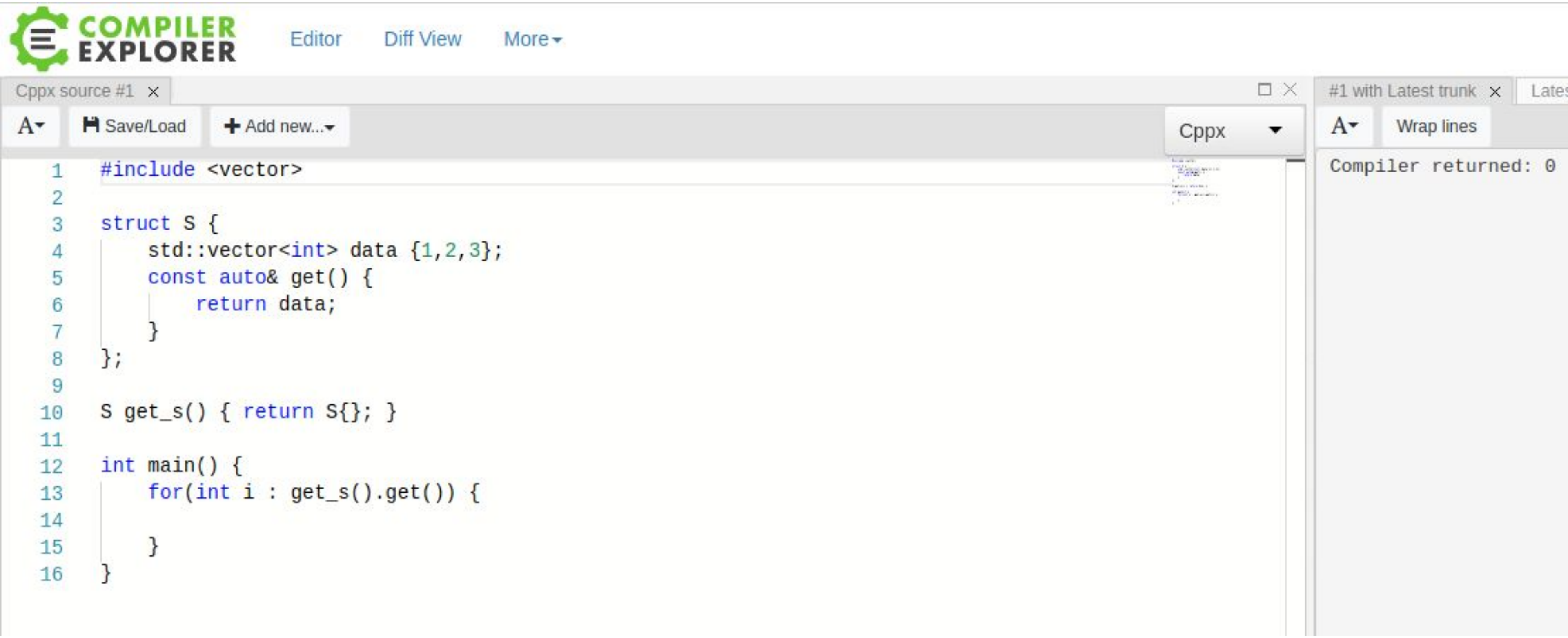
std::array (https://godbolt.org/z/dkpkGC)



```cpp
#include <array>
using namespace std;

void f() {
    array<int*, 1> v { nullptr };
    int *p = v[0];
}

```

```
<source>:6:14: warning: passing a dangling pointer as
argument [-Wlifetime]
    int *p = v[0];
             ^
<source>:5:33: note: temporary was destroyed at the end of
the full expression
    array<int*, 1> v { nullptr };
                                ^
1 warning generated.
Compiler returned: 0
```

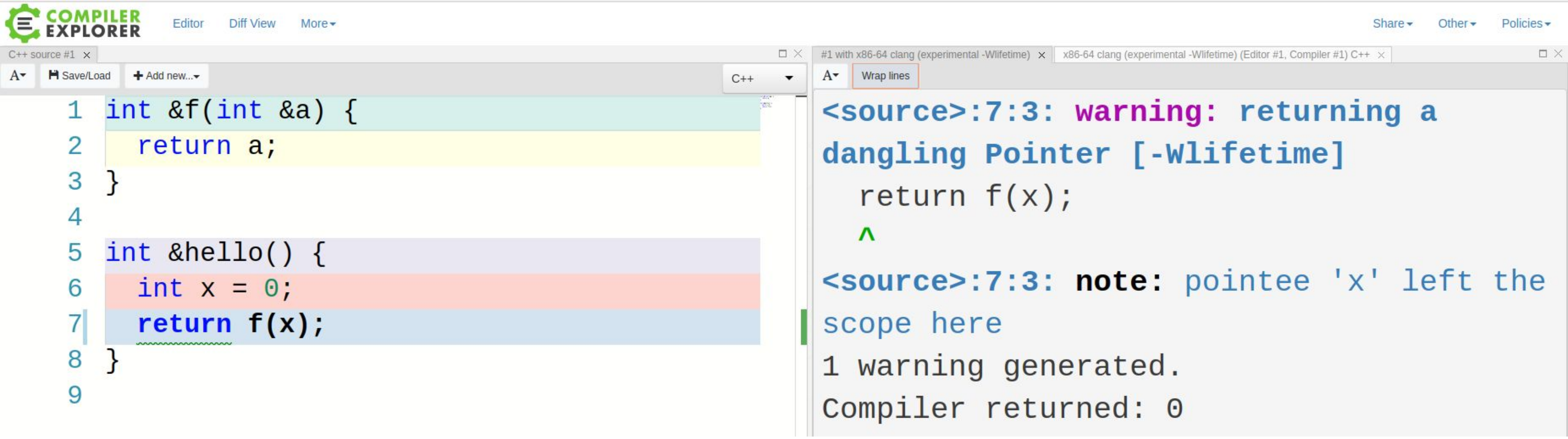
Jason Turner's talk (https://cppx.godbolt.org/z/5Uw3co)



```cpp
#include <vector>

struct S {
    std::vector<int> data {1,2,3};
    const auto& get() {
        return data;
    }
};

S get_s() { return S{}; }

int main() {
    for(int i : get_s().get()) {

    }
}
```

```
Compiler returned: 0
```

Robert O'Callahan (https://godbolt.org/z/MTqoz9)


COMPILER EXPLORER
Editor
Diff View
More
Share
Other
Policies
C++ source #1
Save/Load
Add new...
C++
#1 with x86-64 clang (experimental -Wlifetime)
x86-64 clang (experimental -Wlifetime) (Editor #1, Compiler #1) C++
Wrap lines

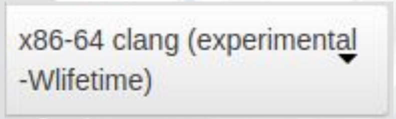
```
int &f(int &a) {
  return a;
}

int &hello() {
  int x = 0;
  return f(x);
}

```

```
<source>:7:3: warning: returning a dangling Pointer [-Wlifetime]
  return f(x);
  ^
<source>:7:3: note: pointee 'x' left the scope here
1 warning generated.
Compiler returned: 0
```

Try it out and give Feedback!

https://github.com/mgehre/clang

https://godbolt.org



x86-64 clang (experimental -Wlifetime)

https://cppx.godbolt.org (With metaclasses; Add -Wlifetime)

Contributions welcome!

Bonus

https://godbolt.org/z/UE-Mb0