A look at C++ through the glasses of a language tool

Anastasia Kazakova
@anastasiak2512

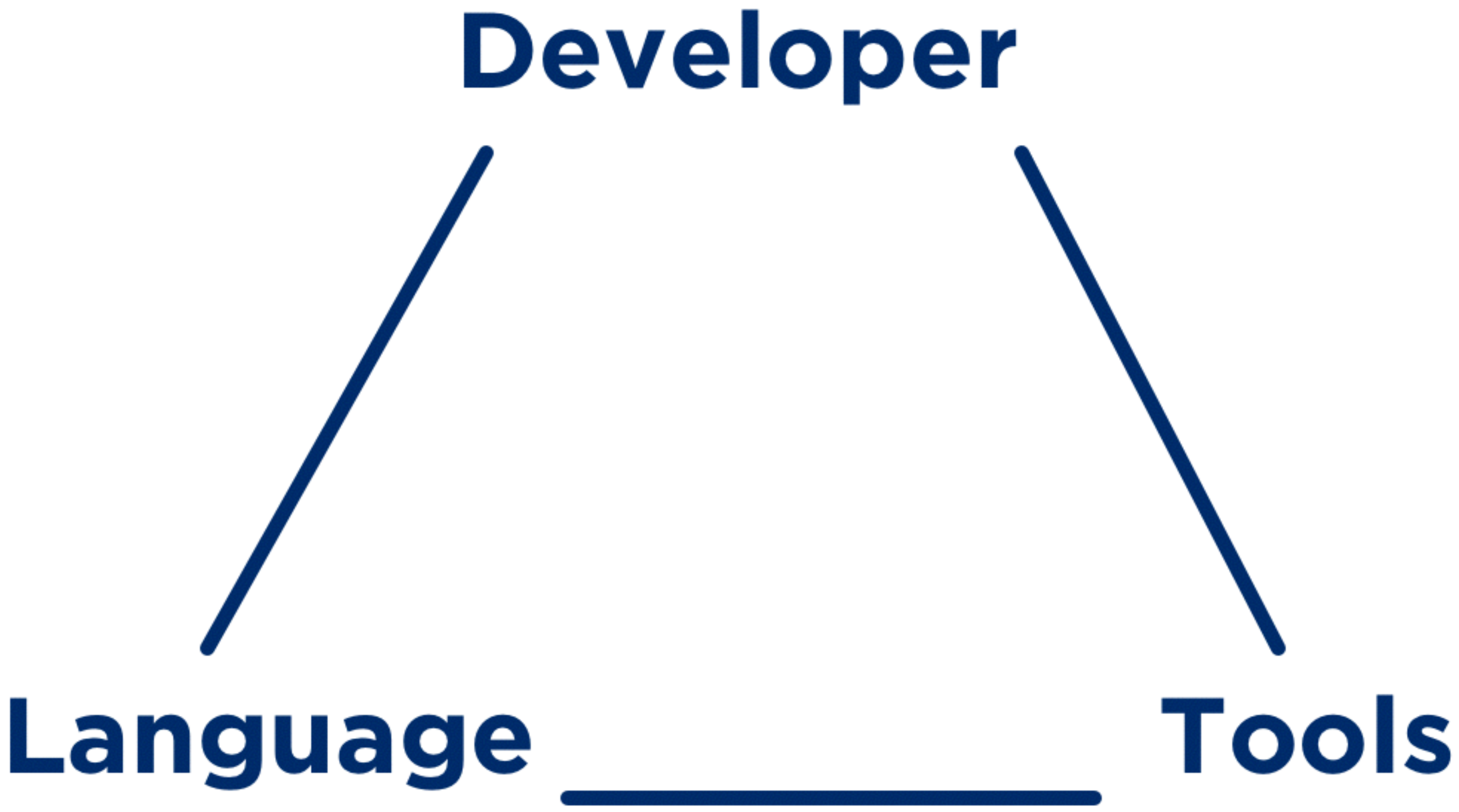
C++Now 2017

Background

- C/C++, embedded Linux on VoIP gateways and routers, VIM-addicted
- C++, congestion & users policies in 3G/4G/LTE networks, NetBeans user
- Product Marketing Manager for CLion

All connected

- All three have a common goal
- All three need each other
- All three rely on each other



Developer

Language

Tools

IDE.
What do you expect?

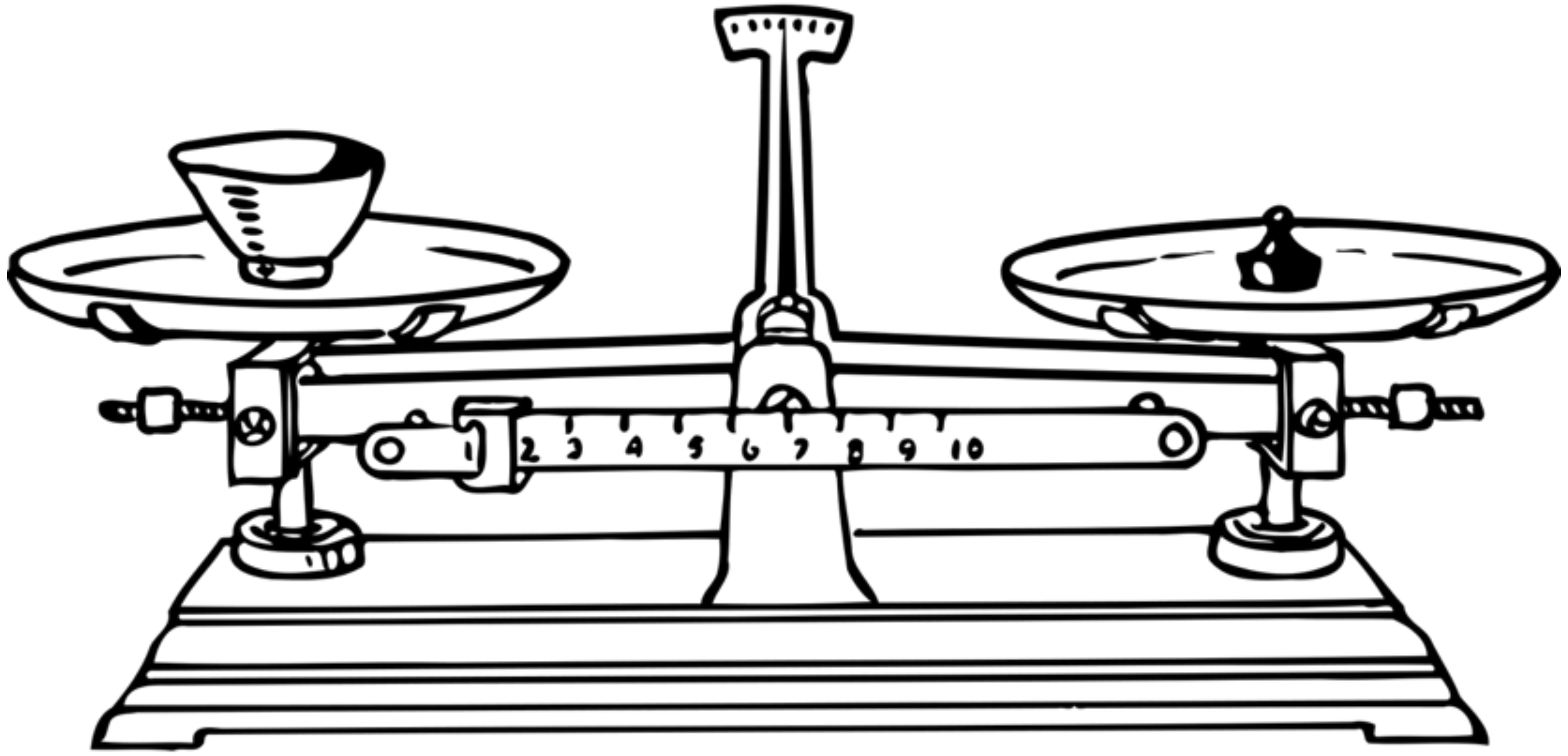
—

- Correctness: 100% correct in terms of the language
- Performance: provides completion before I’m tired of waiting for it
- Smartness: more on-the-fly intellisense
- Universal: knows about the whole project
- Helpful: can work with the incorrect code
- Swiss army knife: other tools on board

IDE.
Balance

—

- Correctness: 100% correct in terms of the language
- Performance: provides completion before I’m tired of waiting for it
- Smartness: more on-the-fly intellisense
- Universal: knows about the whole project
- Helpful: can work with the incorrect code
- Swiss army knife: other tools on board



IDE.
Our reality

—

- IDE has to deal with any code
  - Legacy code, decades of language baggage
  - Modern standards, drafts, TS, etc.
  - Legacy code and modern code co-exist
  - Incorrect code

- If to compare with another “language tools” – compilers:
  - different goals
  - knowledge about the whole project, not just one translation unit
  - error-recovery

Why this talk?

- Share the view – knowledge is power
- Share excitement, pain, lessons learned
- Share it with program committee / those who influence the language
- Tips to avoid foot-shooting

Let’s play

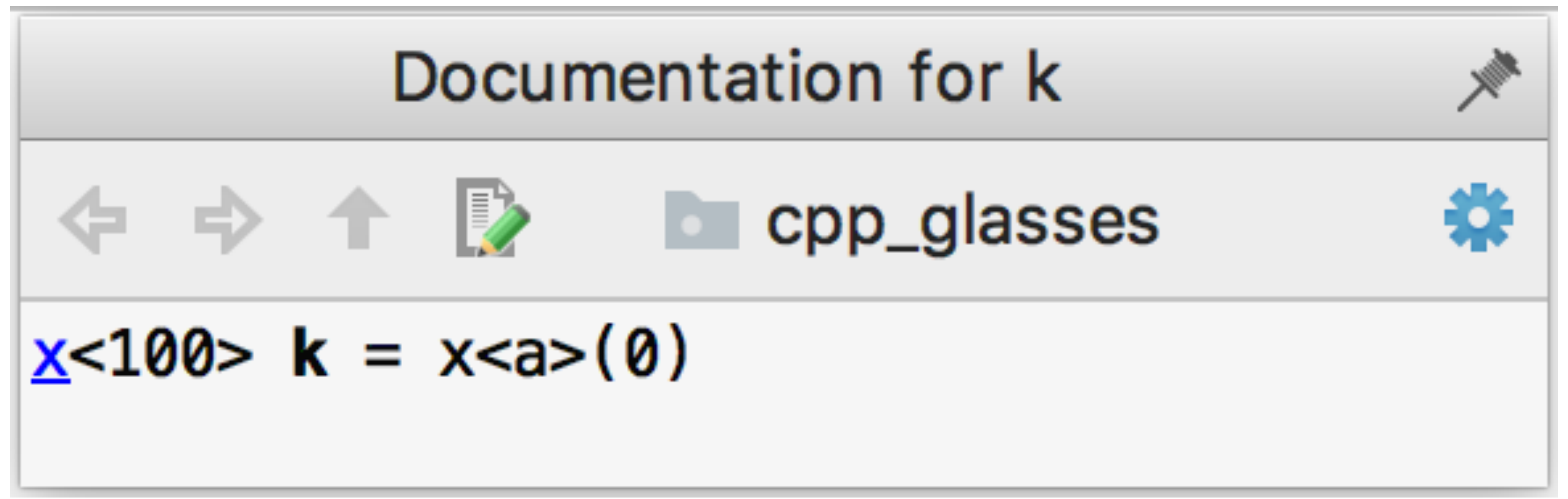
How about some quick C++ game?

Let’s play

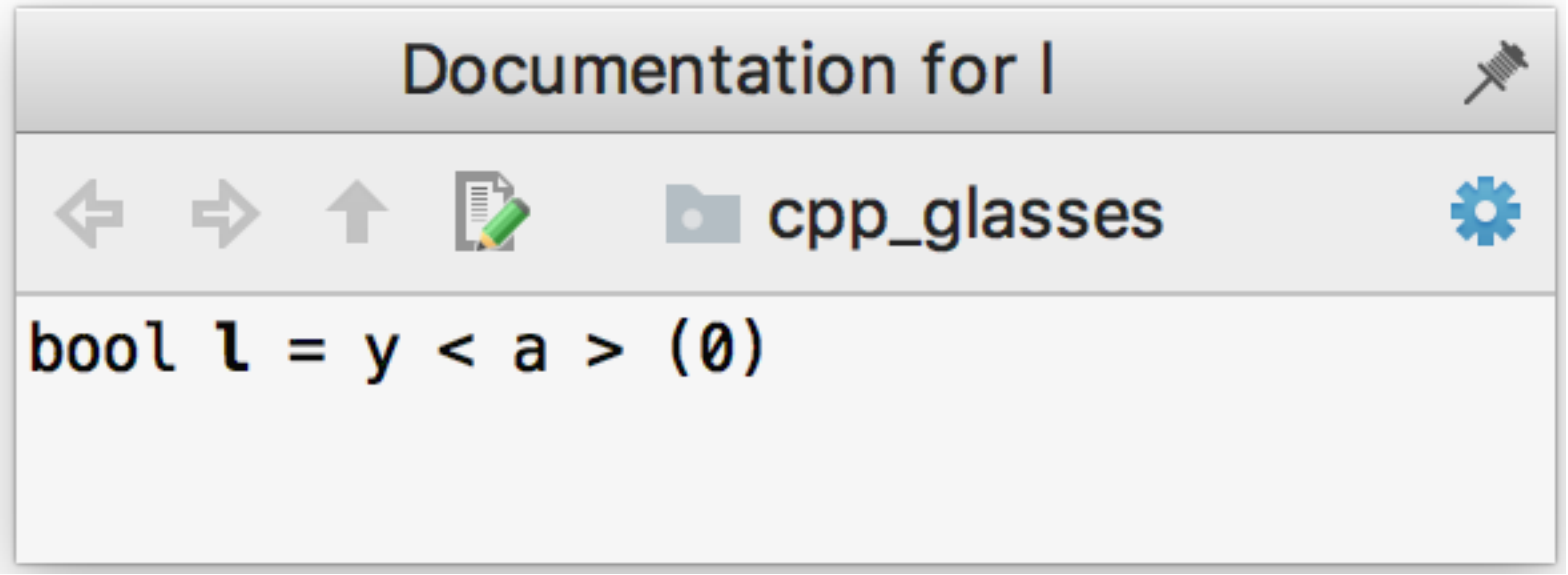
Guess about k and l?

```
template<int>
struct x {
    x(int i) { }
};

void test(int y) {

    const int a = 100;

    auto k = x<a>(0);
    auto l = y<a>(0);
}
```

Let’s play



```
x<100> k = x<a>(0)
```



```
bool l = y < a > (0)
```

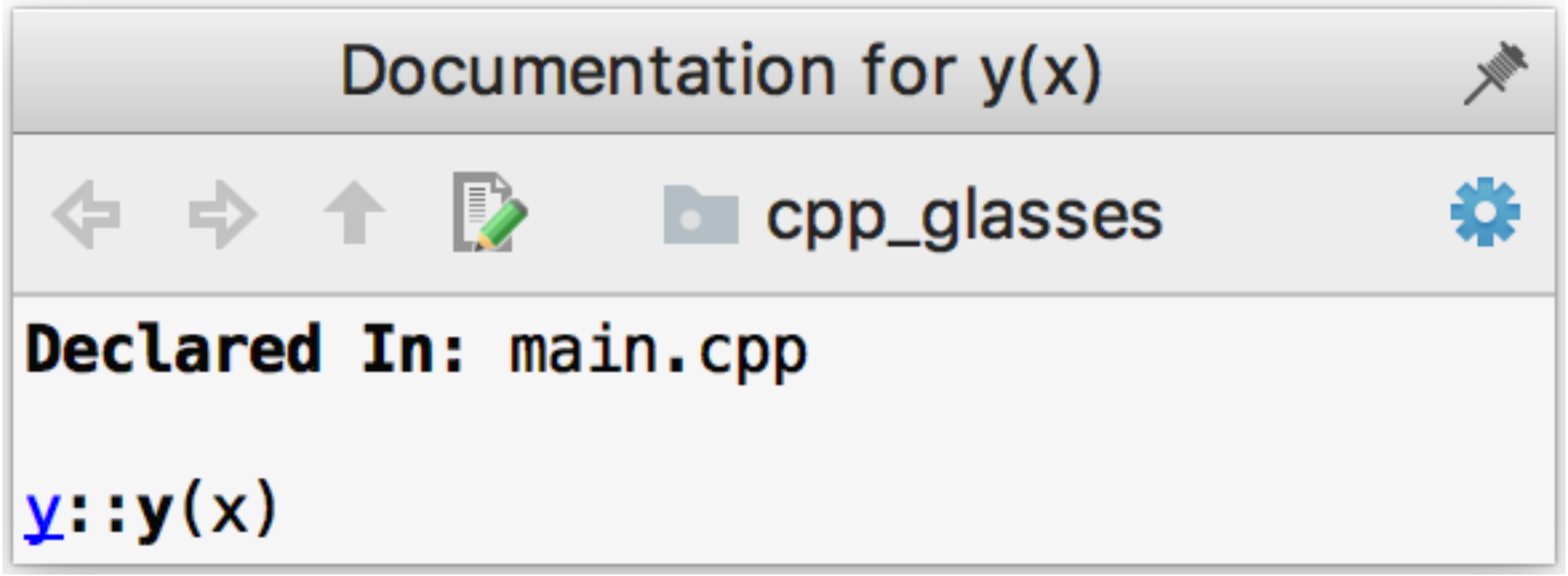
```
template<int>
struct x {
    x(int i) { }
};

void test(int y) {

    const int a = 100;

    auto k = x<a>(0);
    auto l = y<a>(0);
}
```

Let’s play

Guess about y and z?

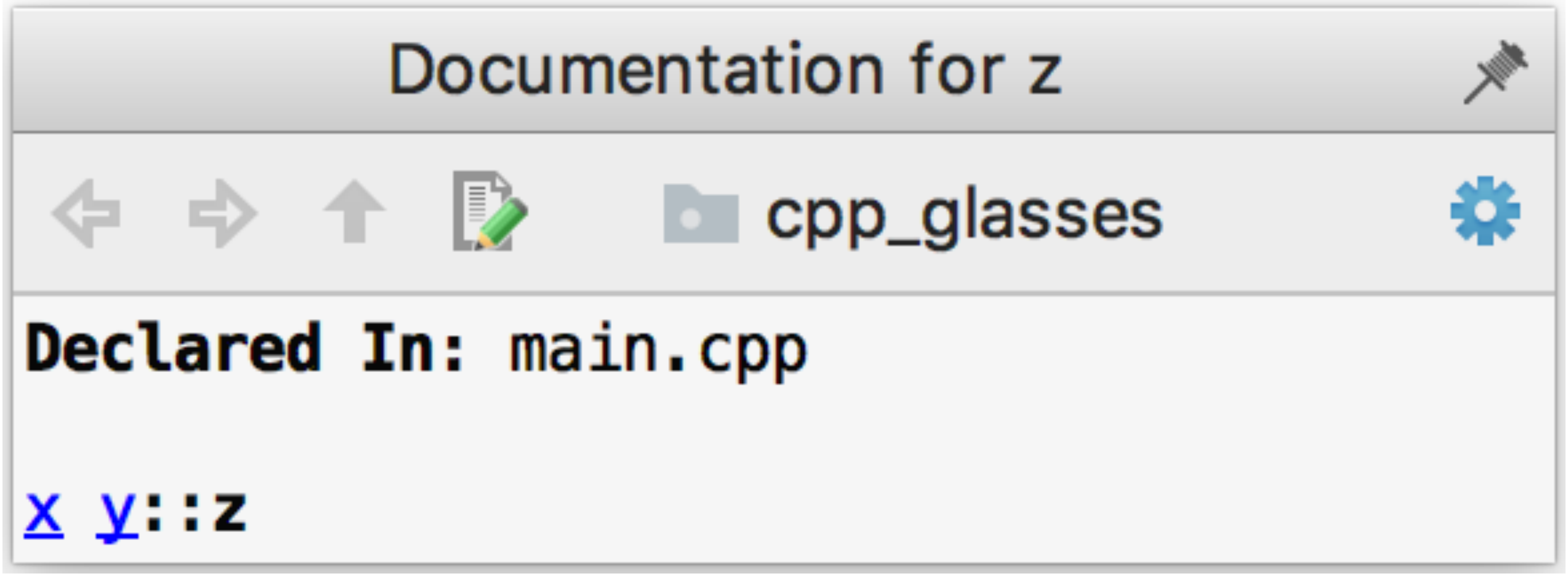
```
void test() {
    struct x {
    };

    struct y {
        y(x) {};
        x(z);
    };
}
```

Let’s play



Declared In: main.cpp

y::y(x)



Declared In: main.cpp

x y::z

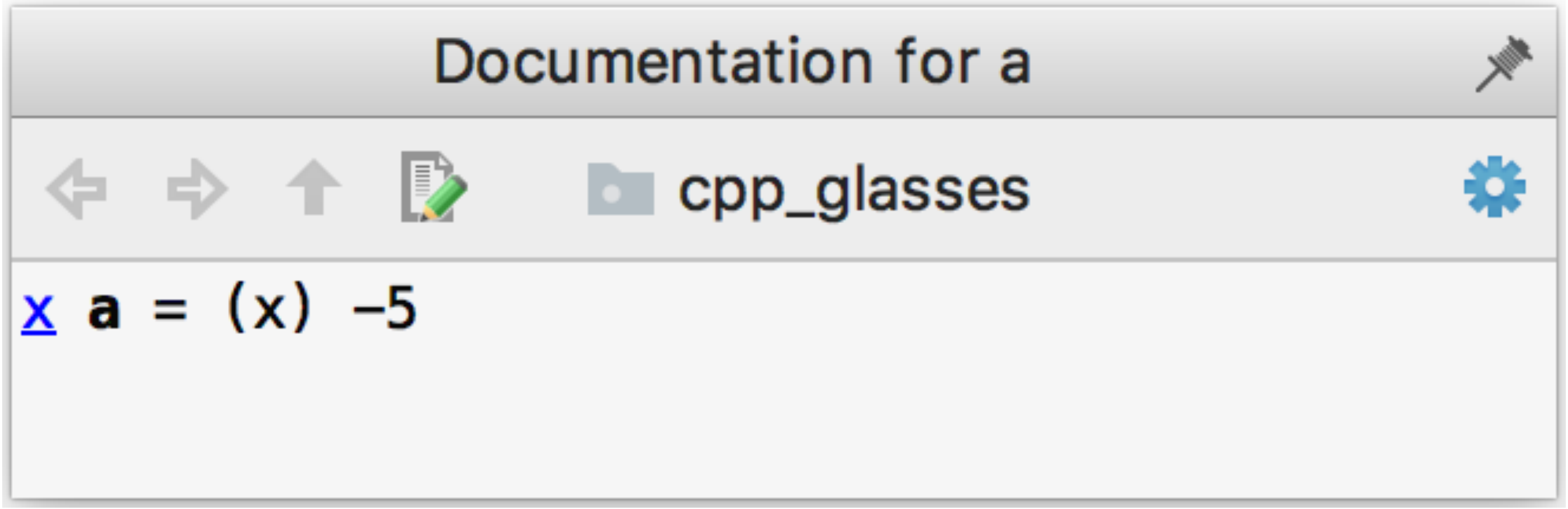
```
void test() {
    struct x {
    };

    struct y {
        y(x) {};
        x(z);
    };
}
```

Let’s play

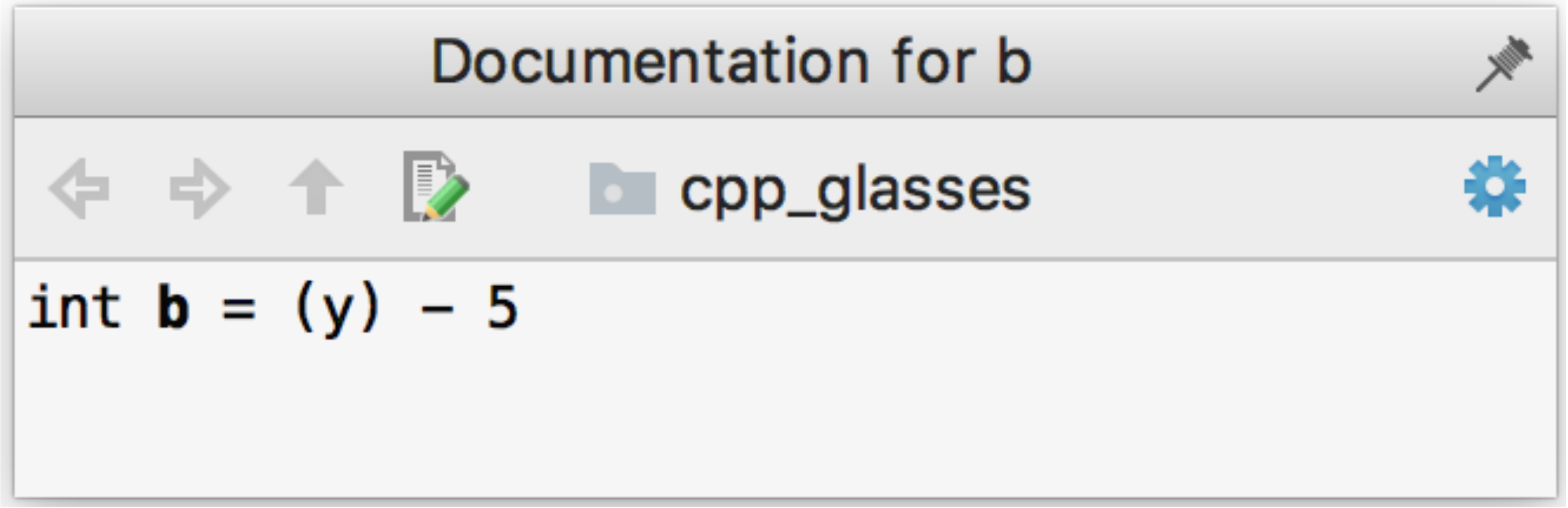
Guess about a and b?

```
void test() {
    struct x {
        x(int) { };
    };

    int y = 100;

    auto a = (x)-5;
    auto b = (y)-5;
}
```

Let’s play



```
x a = (x) -5
```



```
int b = (y) - 5
```

```
void test() {
    struct x {
        x(int) { };
    };

    int y = 100;

    auto a = (x)-5;
    auto b = (y)-5;
}
```

Why C++ is different?
Parser & Resolve

Summarizing all the samples:

*To parse C++ we need to distinguish **types** from **non-types***

```
//List of declarations
int(x), y, *const z;
//int x; int y; int *const z;

//List of expressions
int(x), y, new int;
//( (int(x)), (y), (new int) );
```

Why C++ is different?
Parser & Resolve

1. With C++ we need to resolve while parsing to understand if something is a type or not.

Why C++ is different?
Parser & Resolve

1. With C++ we need to resolve while parsing to understand if something is a type or not.

We need it for:

- highlighting
- formatting

As well as:

- completion
- showing instant navigation
- code analysis
- etc.

What affects the resolve?

—

Resolve depends on: ?

What affects the resolve?

Resolve depends on:

- order of the definitions

```
void test1() {
    fun();
}

int fun();

void test2() {
    fun();
}
```

What affects the resolve?

Resolve depends on:

- order of the definitions
- default arguments

```
int fun(int);

void test1() {
    fun(); //Too few arguments
}

int fun(int = 0);

void test2() {
    fun();
}
```

What affects the resolve?

Resolve depends on:

- order of the definitions
- default arguments
- overload resolution

```
int fun(int (&arr)[3]);

struct c {
    static int arr[];
};

void test1() {
    fun(c::arr);
//no matching function for call to 'fun'
}

int c::arr[] = {0, 1, 2};

void test2() {
    fun(c::arr);
}
```

C++ Code Highlighting

—

Could we highlight with the lexer?

C++ Code Highlighting

Could we highlight with the lexer?

```cpp
//-std=c++03, clang 4.0
template<typename T> struct S{};

void foo() {
    S<S<int>> t; //error: a space is required between consecutive right angle brackets (use '> >')
}
```

C++ Code Highlighting

Could we highlight with the lexer?

For highlighting matching < >, the tool needs parser/resolve

```cpp
template<typename T> struct S{};

void foo() {
    S<S<int>> t;
}
```

C++ Code Highlighting

Could we highlight with the lexer?

```
#define X(T) T ## T

void foo() {
    int X(public);
}
```

C++ Code Highlighting

Could we highlight with the lexer?

Public keyword can’t be highlighted properly with lexer!

```
#define X(T) T ## T

void foo() {
    int X(public);
}
```

Overload resolution and templates

Code inspections & highlighting

```
struct S1{};
struct S2{};

int foo(S1);
double foo(S2);

template<typename T> struct IT {
    typedef int X;
};

template<> struct IT<int>  {
    static int X;
};

int main() {
    IT<decltype(foo(S1()))>::X a;
    IT<decltype(foo(S2()))>::X b;
}
```

Overload resolution and templates

Templates with proper interface – Concepts!

```
template <class T>
concept bool Magic =
    requires (T a, T b) {
      {a + b} -> Boolean;
      {a * b} -> Boolean;
    };
```

Concepts

C++ Core Guidelines:

- T.10: Specify concepts for all template arguments
- T.12: Prefer concept names over auto for local variables
- and more

```
template <class T>
concept bool Magic =
    requires (T a, T b) {
      {a + b} -> Boolean;
      {a * b} -> Boolean;
    };
```

Concepts

IDE experience:

- Additional information
- Can cache the concept
- Can provide intellisense inside the template

```
template <class T>
concept bool has_foo =
    requires (T t) {
        {t.foo()} noexcept -> int;
    };
```

Why C++ is different?

1. With C++ we need to resolve while parsing to understand if something is a type or not.
2. Functions

Function bodies

- Forms most of the user code
- Nothing escapes to the outer code
- Independant

Function bodies

- Forms most of the user code
- Nothing escapes to the outer code ?
- Independant ?

```
auto foo() {
    struct X {};
    return X();
}
```

Function bodies

```cpp
template<class T, class U>
auto multiply(T const& lhs, U const& rhs) -> decltype(lhs * rhs) {
    return lhs * rhs;
}
```

Function bodies

Simplify your template code with … *if constexpr*!

Function bodies

```cpp
// SFINAE
template <typename T, std::enable_if_t<std::is_pointer<T>{}>* = nullptr>
auto get_value(T t) {
    return *t;
}

template <typename T, std::enable_if_t<!std::is_pointer<T>{}>* = nullptr>
auto get_value(T t) {
    return t;
}
```

Function bodies

```
template <typename T>
auto get_value(T t) {
    if constexpr (std::is_pointer_v<T>) return *t;
    else return t;
}
```

Why C++ is different?

1. With C++ we need to resolve while parsing to understand if something is a type or not.
2. Functions
3. Includes

Why C++ is different?
Includes

Includes

- header files provide information to parser

```
//foo.h
template<int>
struct x {
    x(int i) { }
};

//foo.cpp
#include "foo.h"
void test(int y) {

    const int a = 100;

    auto k = x<a>(0);
    auto l = y<a>(0);
}
```

Why C++ is different?
Includes

Includes

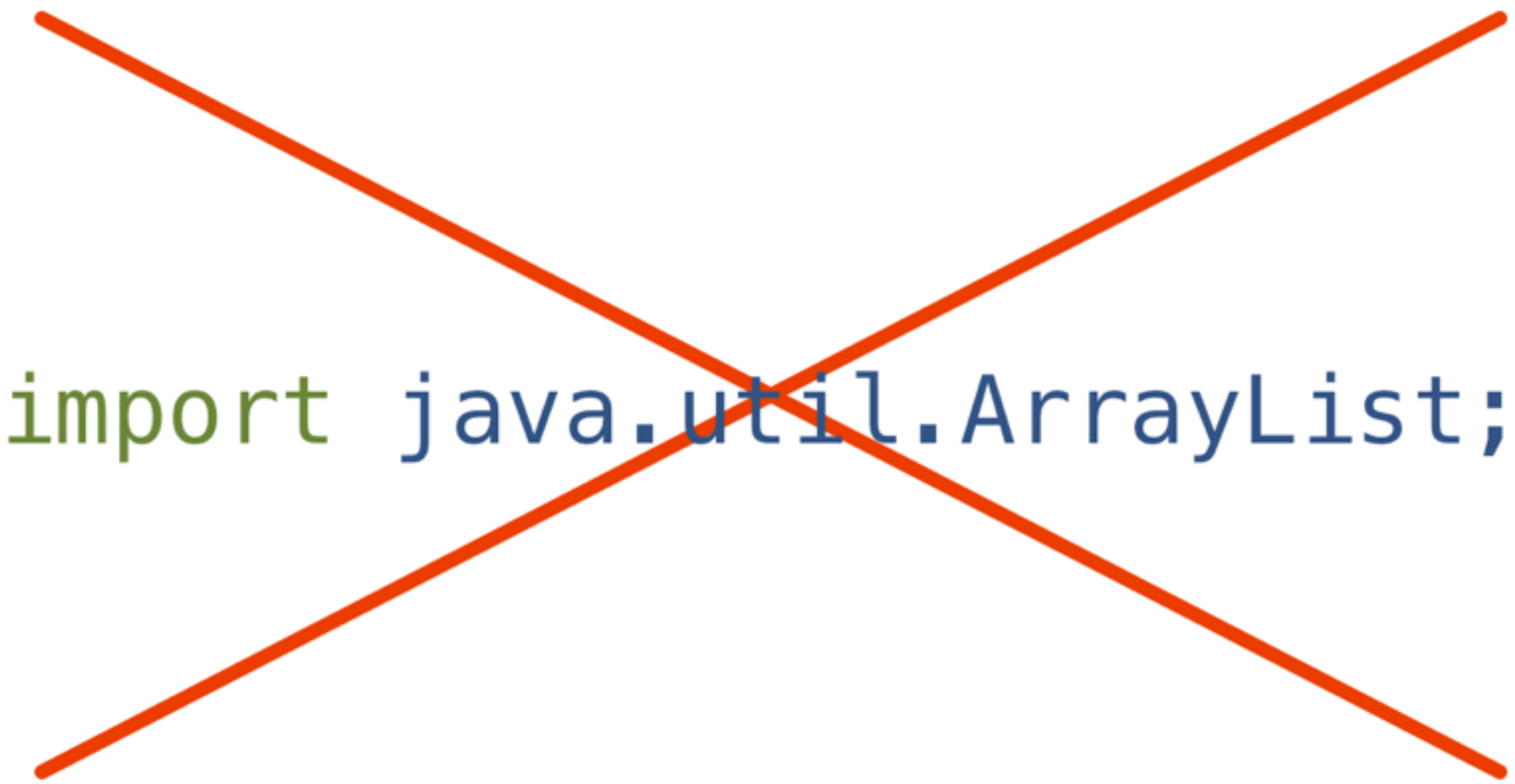
- header files provide information to parser
- they are affected by the context

```
//foo.h
#ifdef MAGIC
template<int>
struct x {
    x(int i) { }
};
#else
int x = 100;
#endif

//foo.cpp
#include "foo.h"
void test(int y) {

    const int a = 100;

    auto k = x<a>(0);
    auto l = y<a>(0);
}
```

Why C++ is different?
Includes

Includes

- header files provide information to parser
- they are affected by the context
- no information about what is included



```
import java.util.ArrayList;
```

Why C++ is different?
Includes

Includes

- header files provide information to parser
- they are affected by the context
- no information about what is included
- takes most of the time
- same headers are included in multiple translation units

```
#include <boost/…>
```

Why C++ is different?
Includes

Good ways to deal with includes:

Why C++ is different? Includes

Good ways to deal with includes:

- Precompiled headers

Why C++ is different?
Includes

Good ways to deal with includes:

- Precompiled headers
- Global includes, less affected by the context

Why C++ is different?
Includes

Good ways to deal with includes:

- Precompiled headers
- Global includes, less affected by the context
- Ill-formed includes are evil

```
//foo.h
return x + 42;

//foo.cpp
auto fun(int x) {
#include "foo.h"
}
```

```
//foo.h
std::vector<int>({1, 2, 3});

//foo.cpp
auto fun() {
   auto x =
       #include "foo.h"
}
```

Why C++ is different?
Includes

Good ways to deal with includes:

- Precompiled headers
- Global includes, less affected by the context
- Ill-formed includes are evil
- Modules are great!

```
//my_module.ixx
module My;

export
int my_shiny_fun(int x) {
…
}

//usage.cpp
int main() {
    my_shiny_fun(10);
}
```

How can the language help?

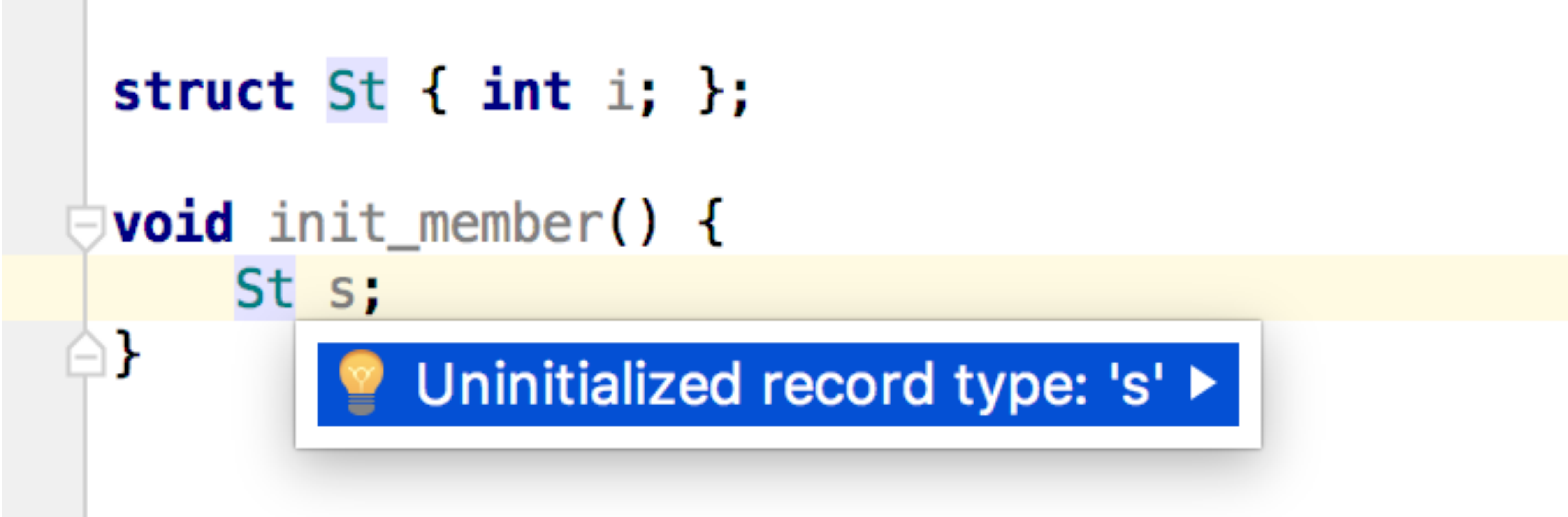
- Modules
- if constexpr
- Concepts
- C++ Core Guidelines

C++ Core Guidelines

- Improve the readability
- Force precisely typed / self-contained code
- Pushing concepts

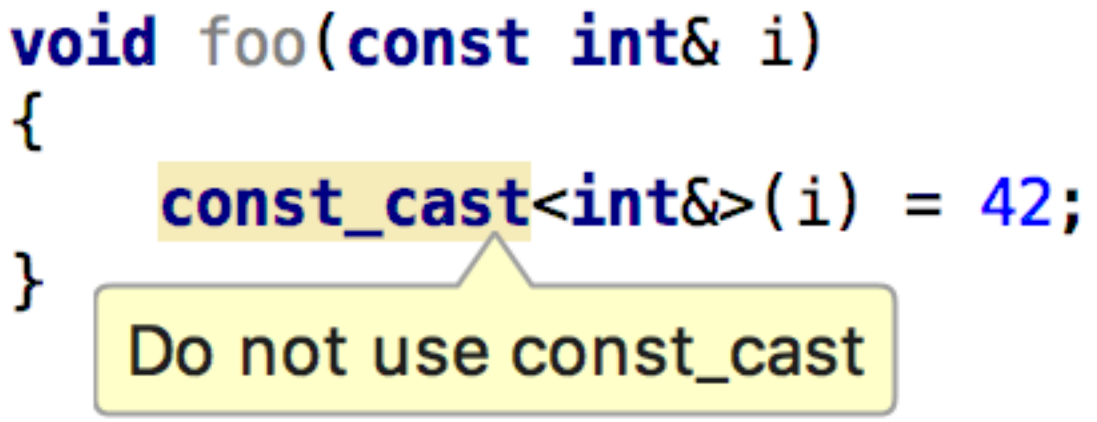
C++ Core Guidelines

- Improve the readability
- Force precisely typed code
- Reduce the side effects
- Pushing concepts



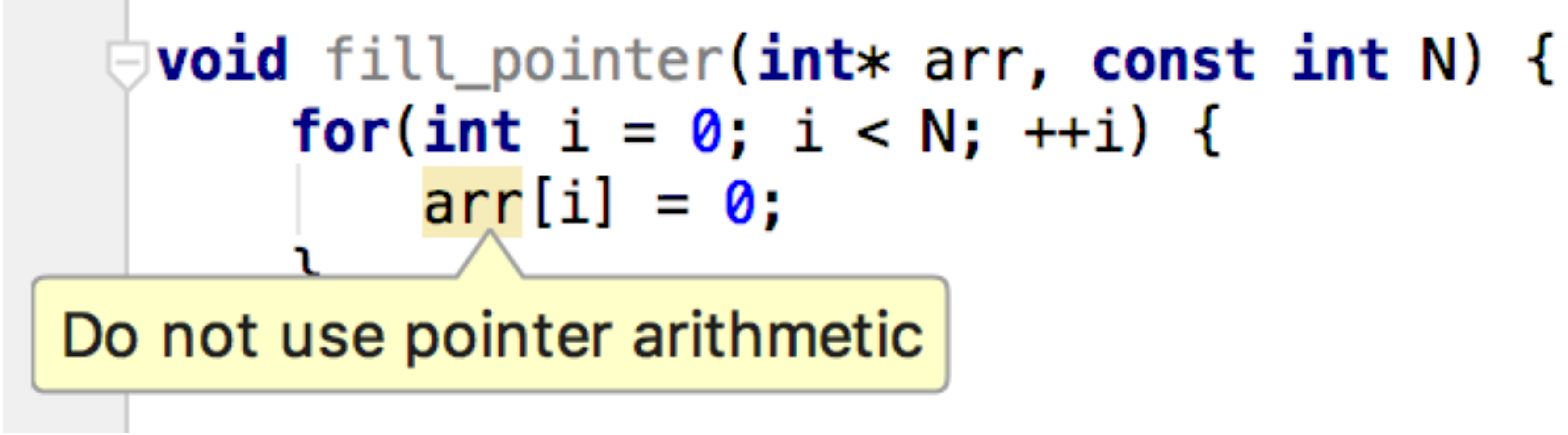
```
struct St { int i; };

void init_member() {
    St s;
}
```

Uninitialized record type: 's'



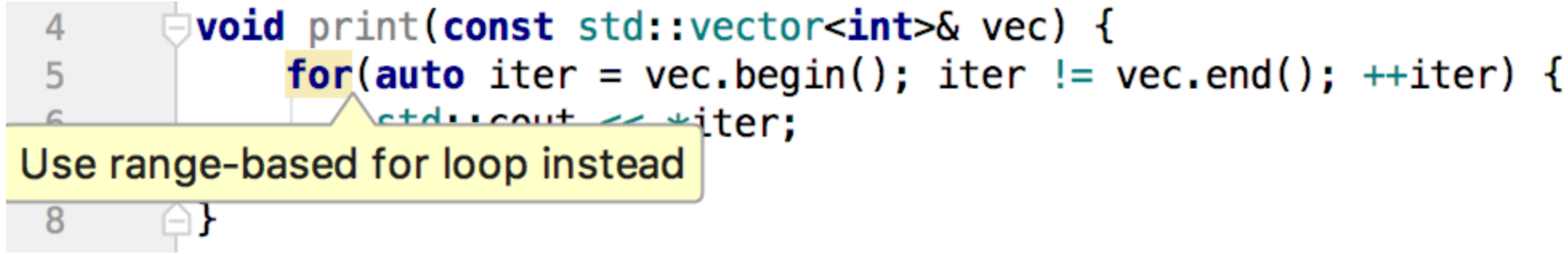
```
void foo(const int& i)
{
    const_cast<int&>(i) = 42;
}
```

Do not use const_cast



```
void fill_pointer(int* arr, const int N) {
    for(int i = 0; i < N; ++i) {
        arr[i] = 0;
    }
```

Do not use pointer arithmetic



```
void print(const std::vector<int>& vec) {
    for(auto iter = vec.begin(); iter != vec.end(); ++iter) {
        std::cout << *iter;
}
```

Use range-based for loop instead

C++ ecosystem

- Build systems
- Compilers
- Unit test frameworks
- Code styles
- Dependency managers

Thank you for your attention

—

Questions?