# New standards to the rescue: the view through an IDE’s glasses

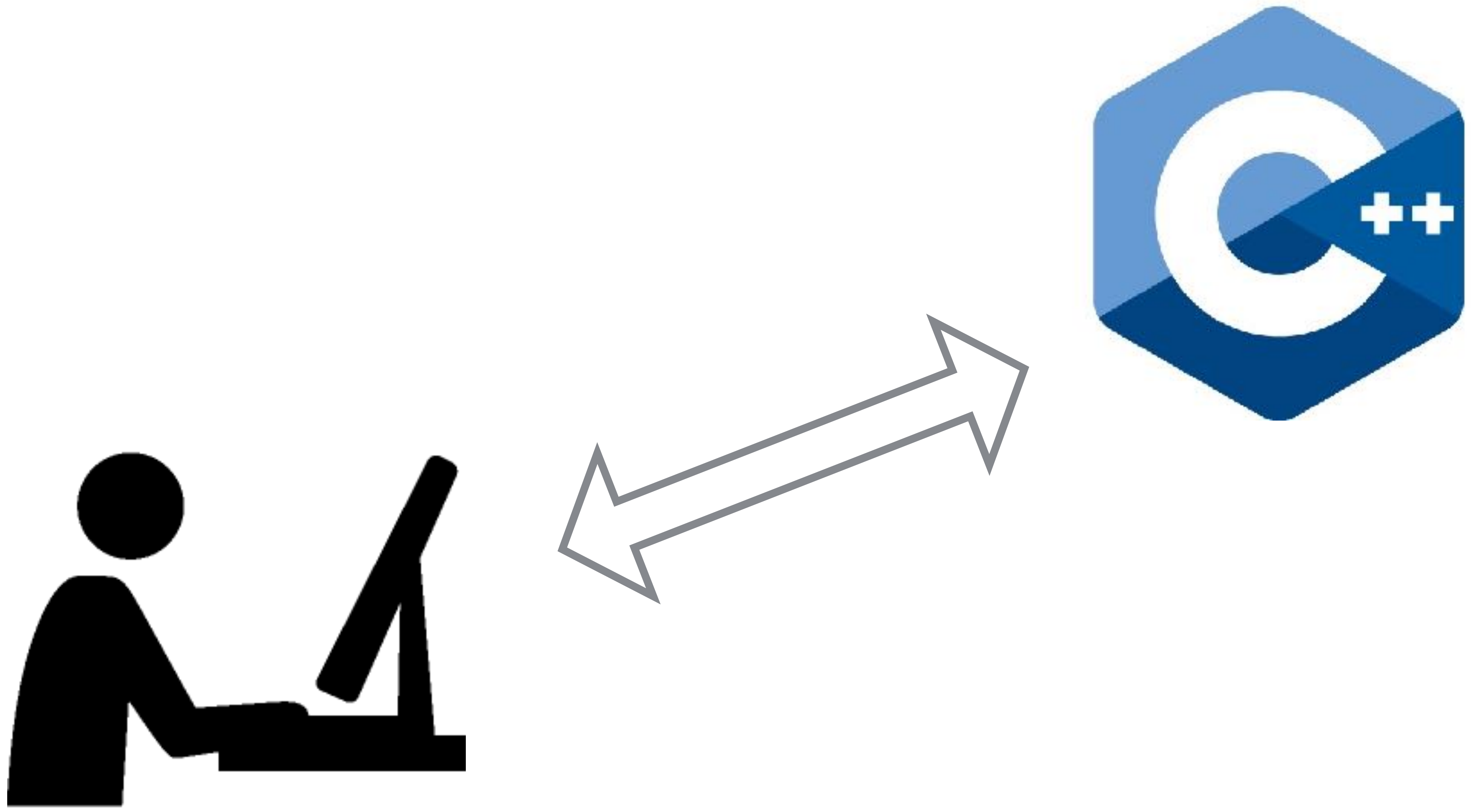
Anastasia Kazakova

JetBrains

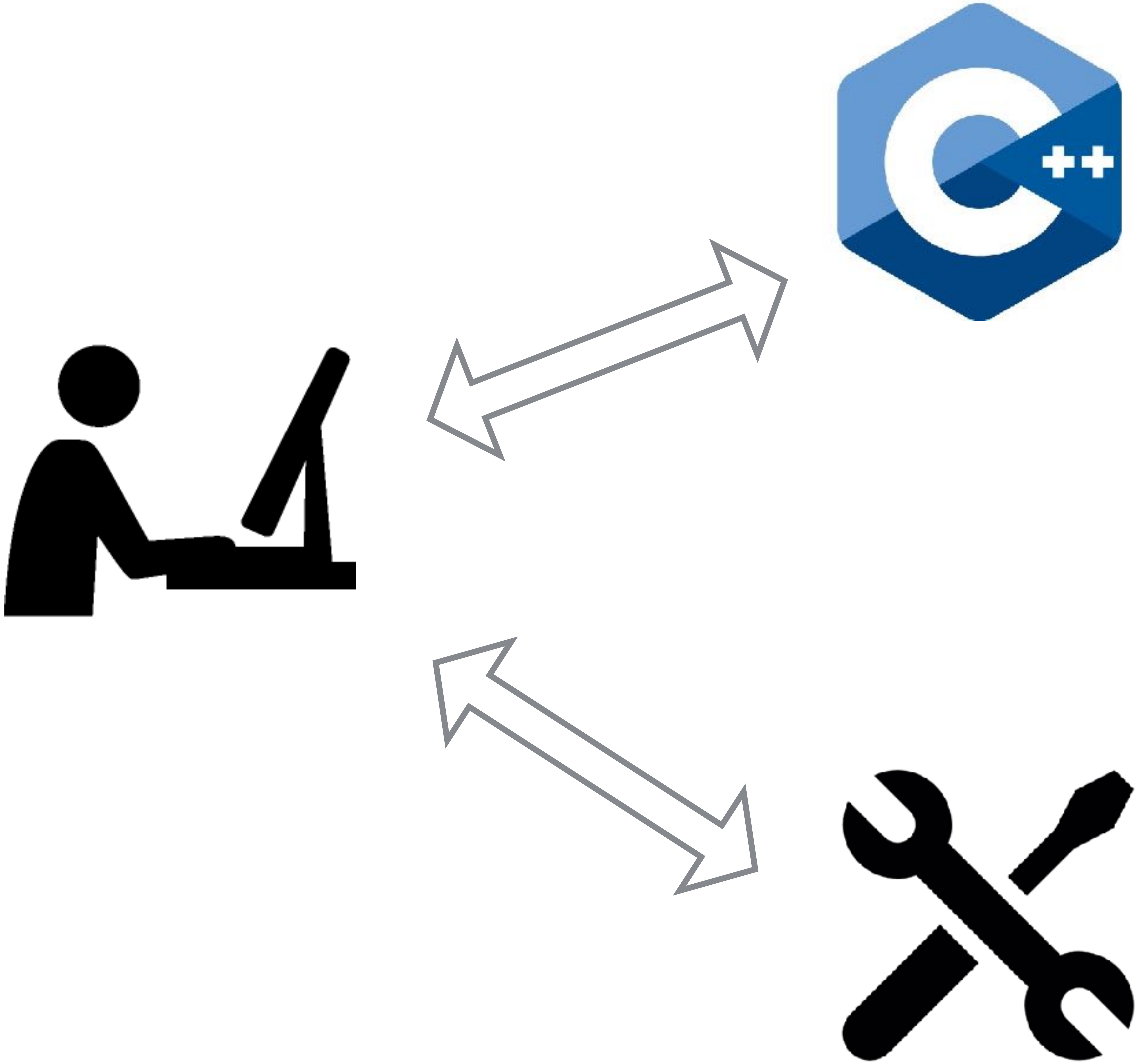
@anastasiak2512

# Perspective

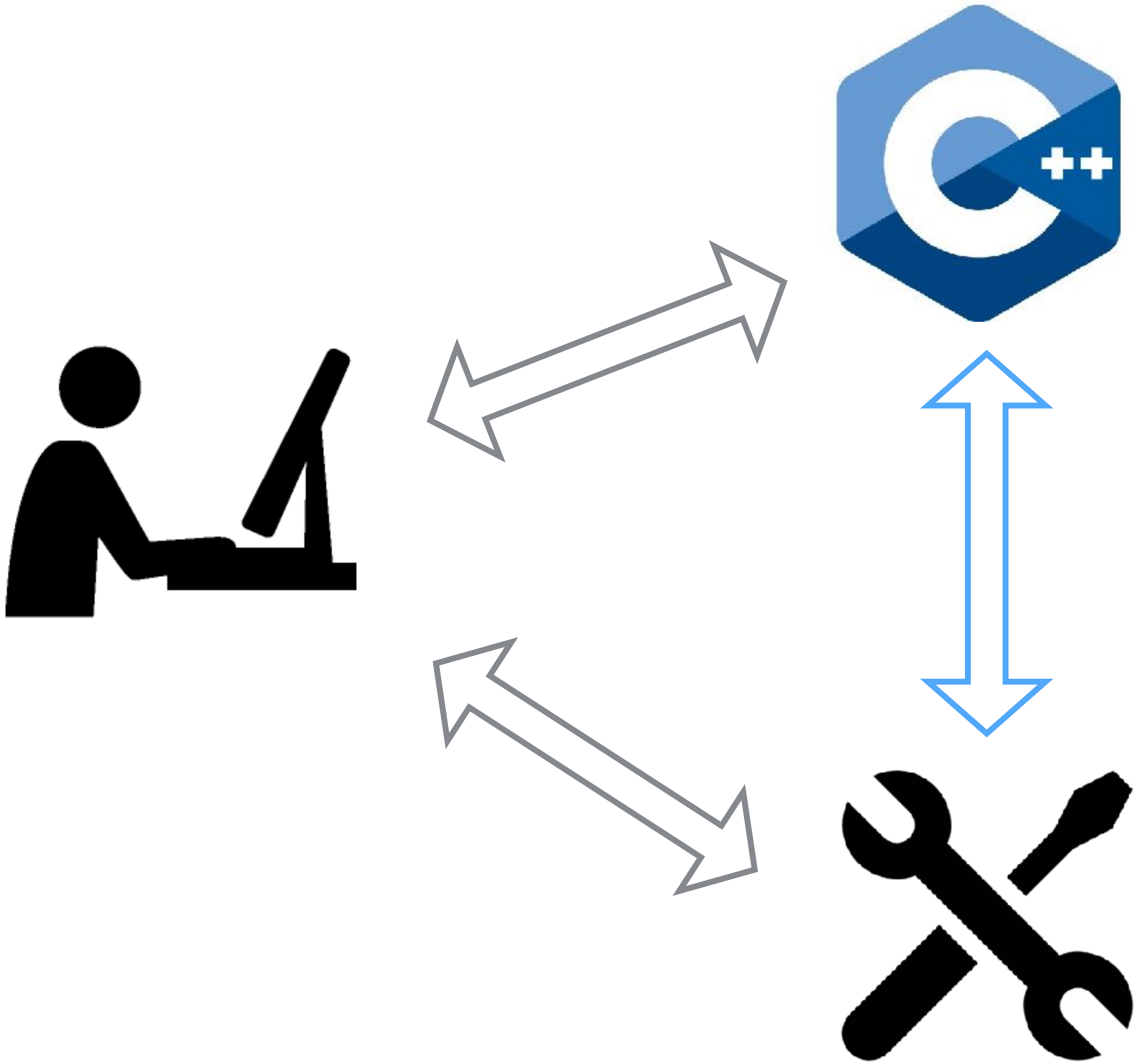
# Perspective



# Perspective



# Perspective



# What this talk is about?

- Where are we now with C++? How we (tools) see it?
- How do we (tools) cope?
- View on the language, hopes for the future

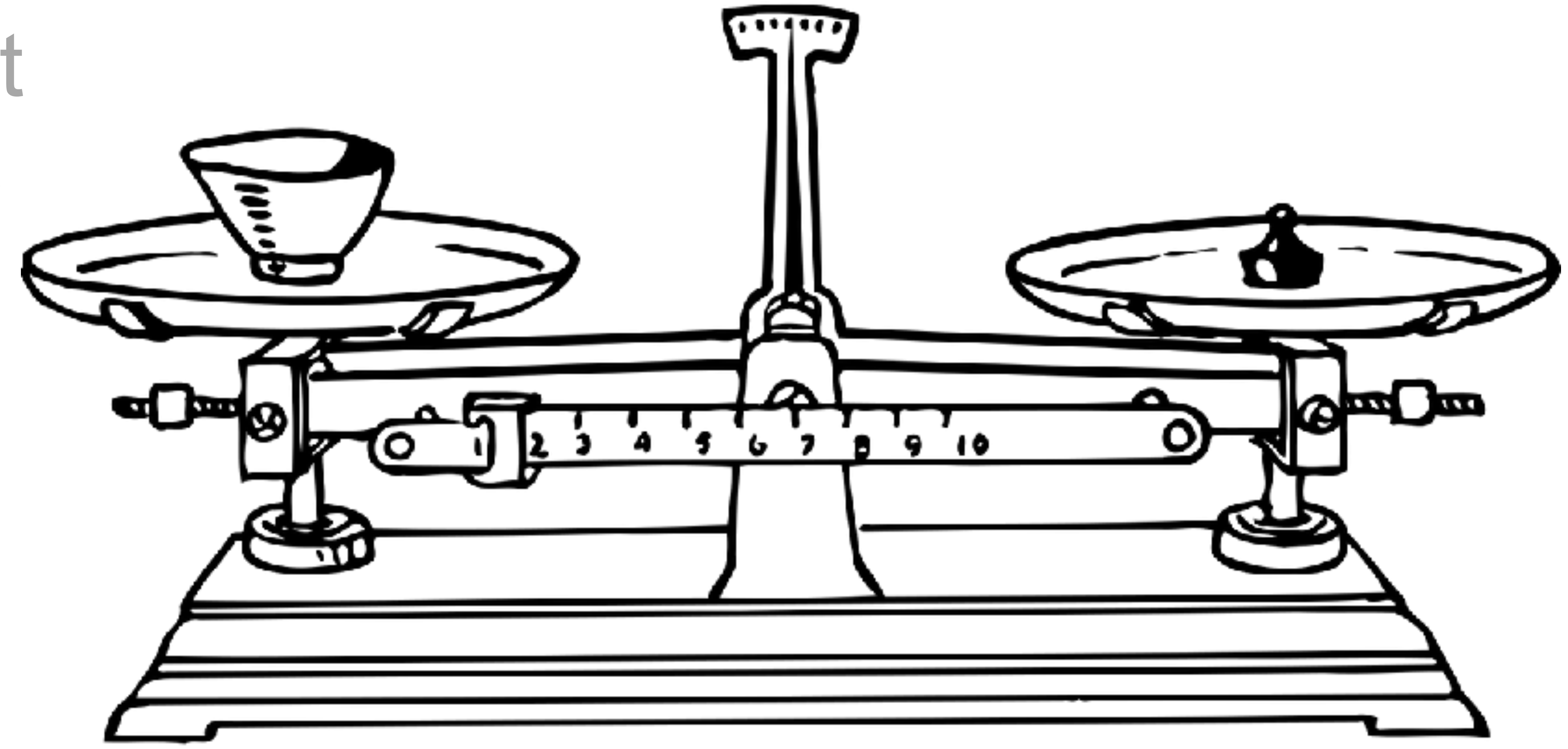
*tools = IDEs

# IDE: Expectation

- *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

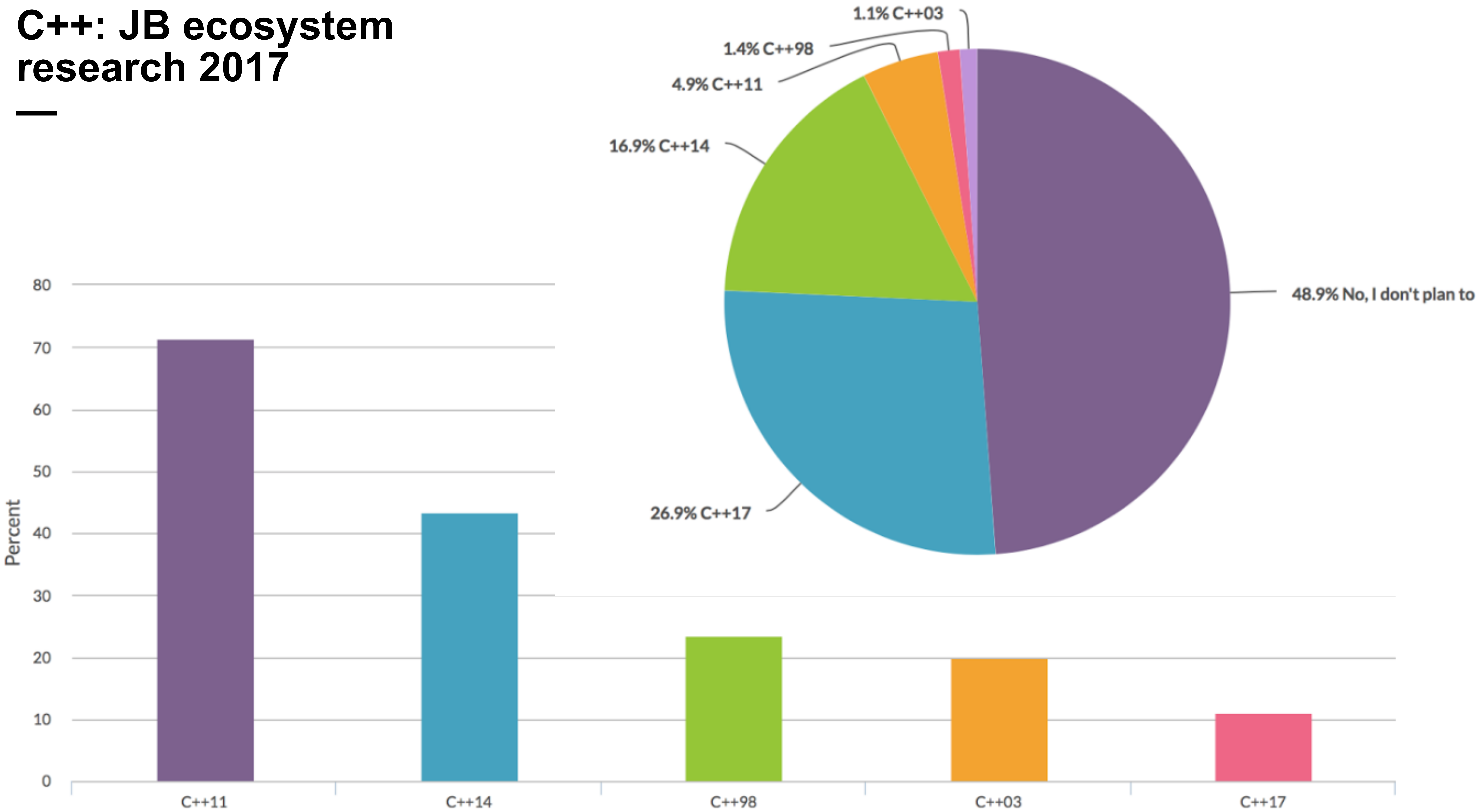
- Correctness: 100% correct in terms of the language
- Performance: provides completion before I’m tired of waiting for it
- Smartness: more on-the-fly intellisense
- Universal: knows about the whole project
- Helpful: can work with the incorrect code
- Swiss army knife: other tools on board



# C++: Our reality

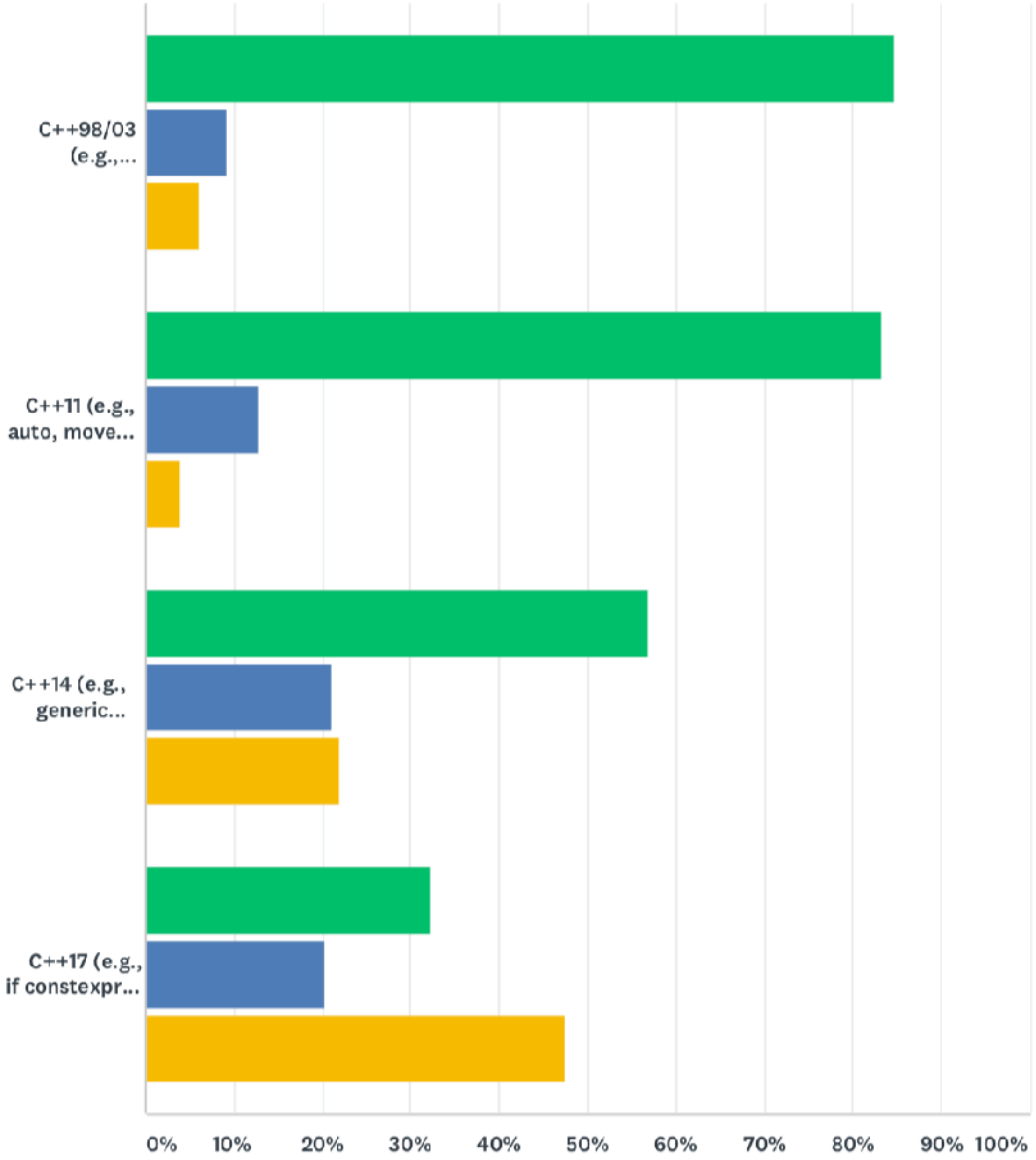
- IDE works with any code
  - Legacy code, decades of language baggage
  - Modern standards, drafts, TS, etc.
  - Legacy code and modern code co-exist
  - Incorrect code

# C++: JB ecosystem research 2017



# C++: ISOCPP research 2018

Constexpr
Edge
References
Resources
Toolchain
Impossible
Practices
MSVC
Learn
Nope
Code
Modules
Compiler
Hard to Understand
New Features
Colleagues
Standard
Amount
Language
Tools
Difficulty
Evolves
New Stuff
Dependencies
Older
Past
Books
Difficult to Understand

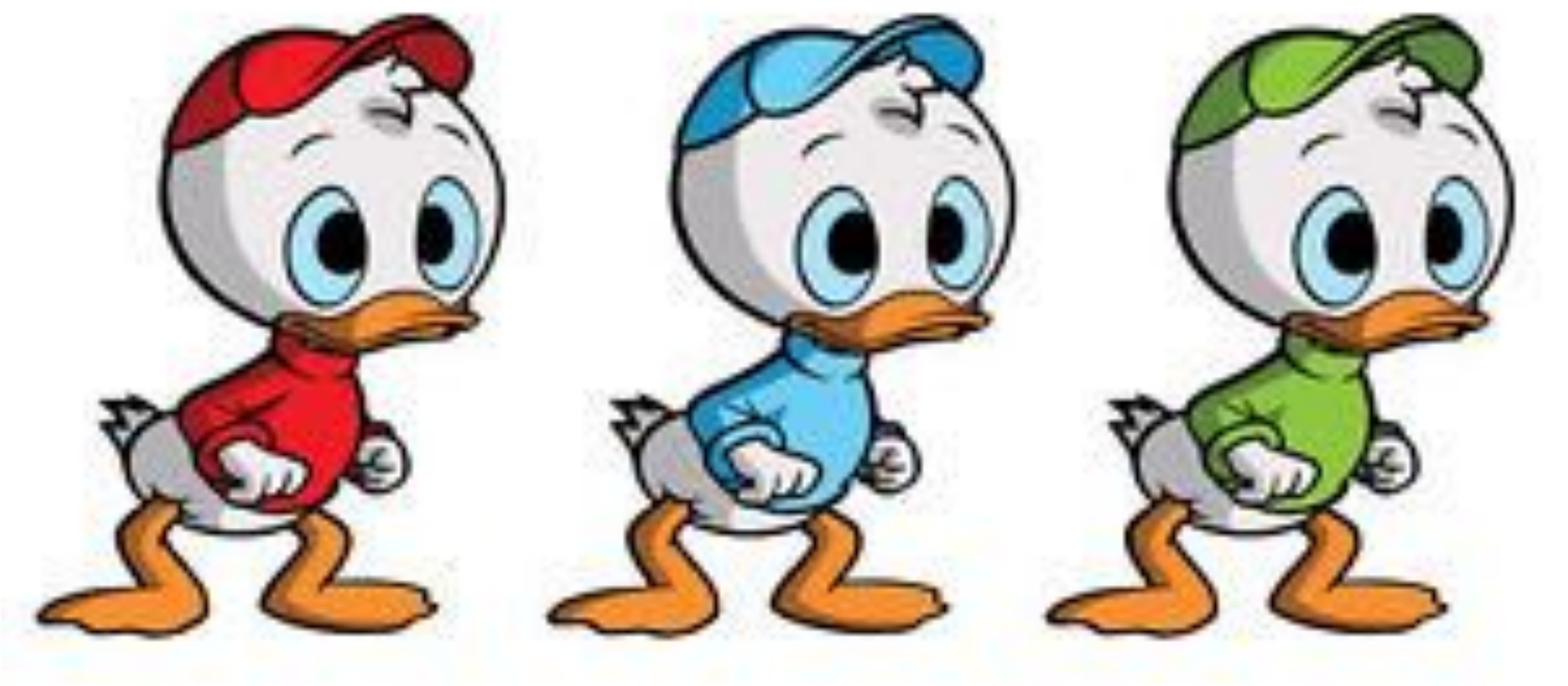


# C++ difficulties

```
template<class T, int ... X>
T pi(T(X...));

int main() {
    return pi<int, 42>;
}
```

# C++ game

Are they different?

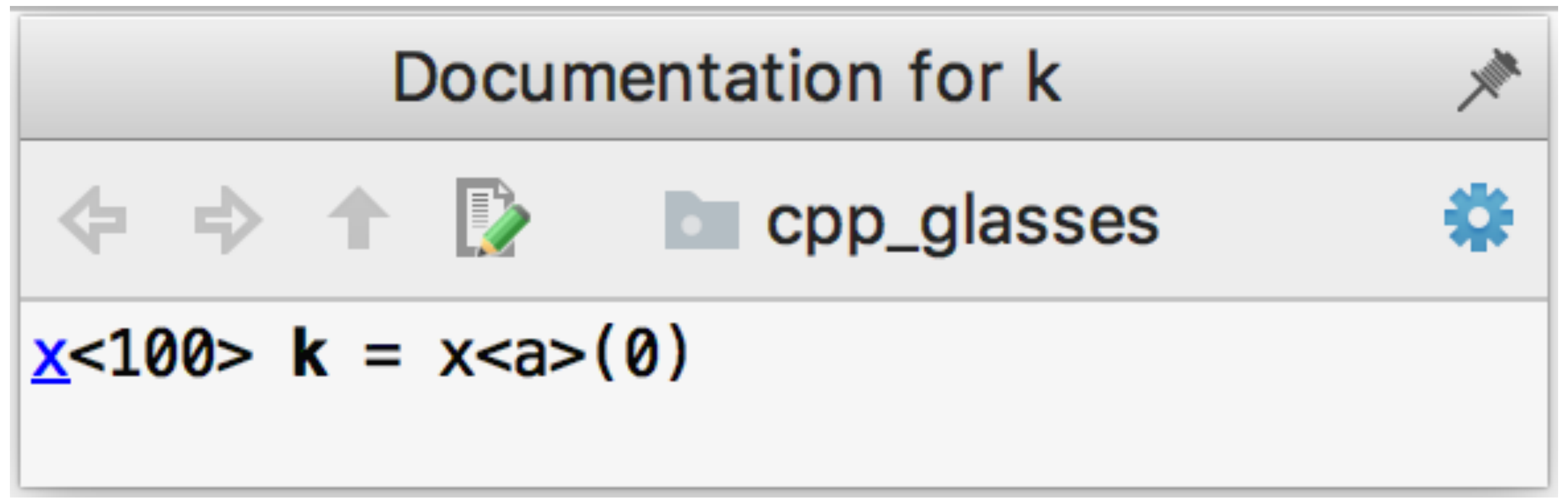


# C++ game

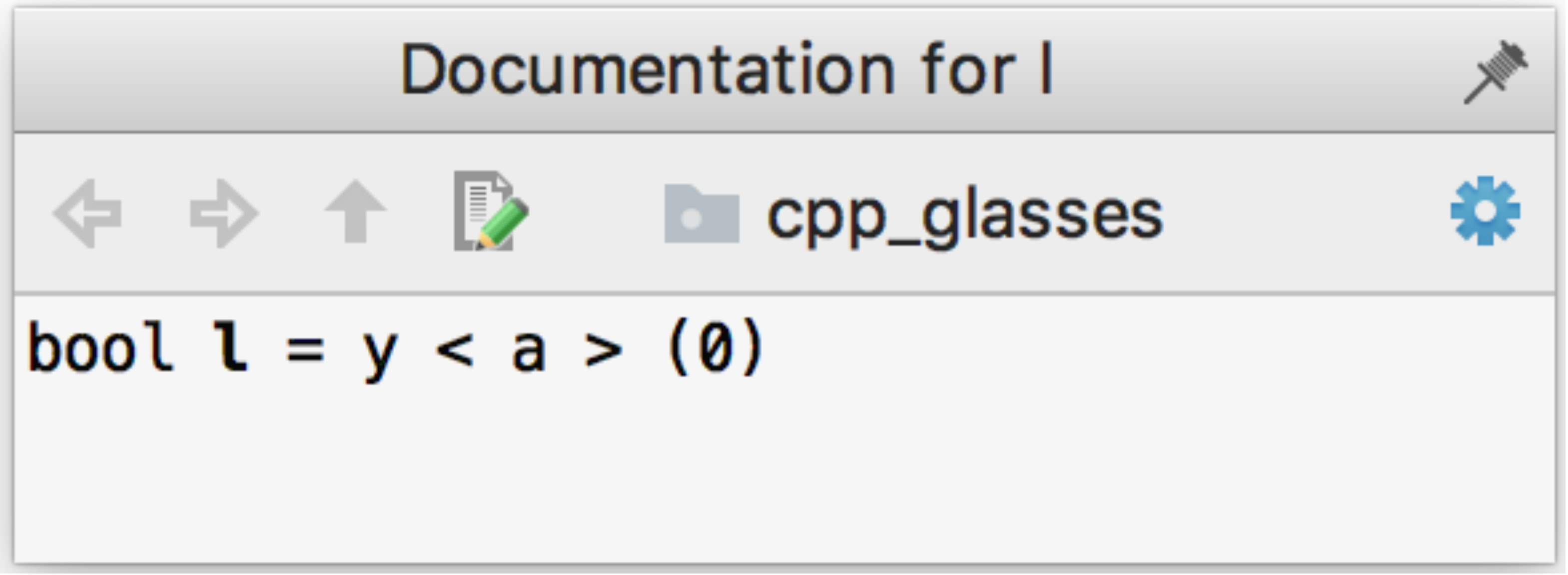
```cpp
template<int>
struct x {
    x(int i) { }
};

void test(int y) {

    const int a = 100;

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

# C++ game

```
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);
}
```



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

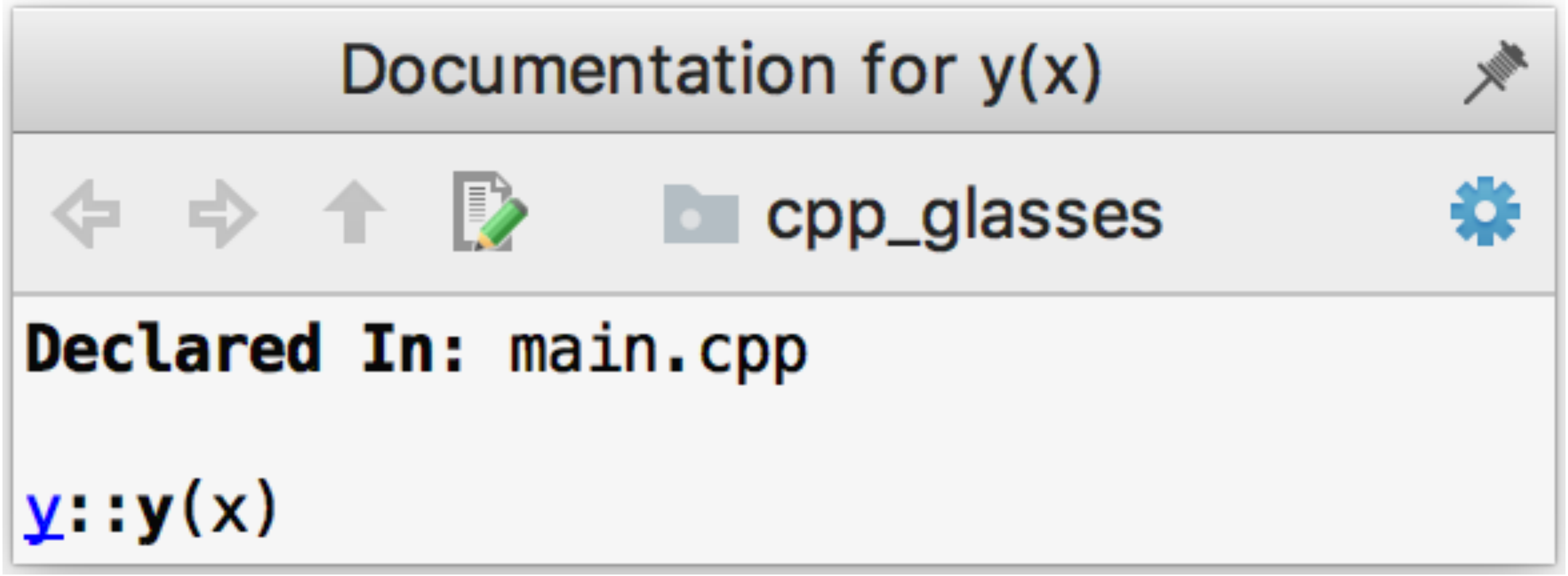


```
bool l = y < a > (0)
```

# C++ game

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

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

# C++ game



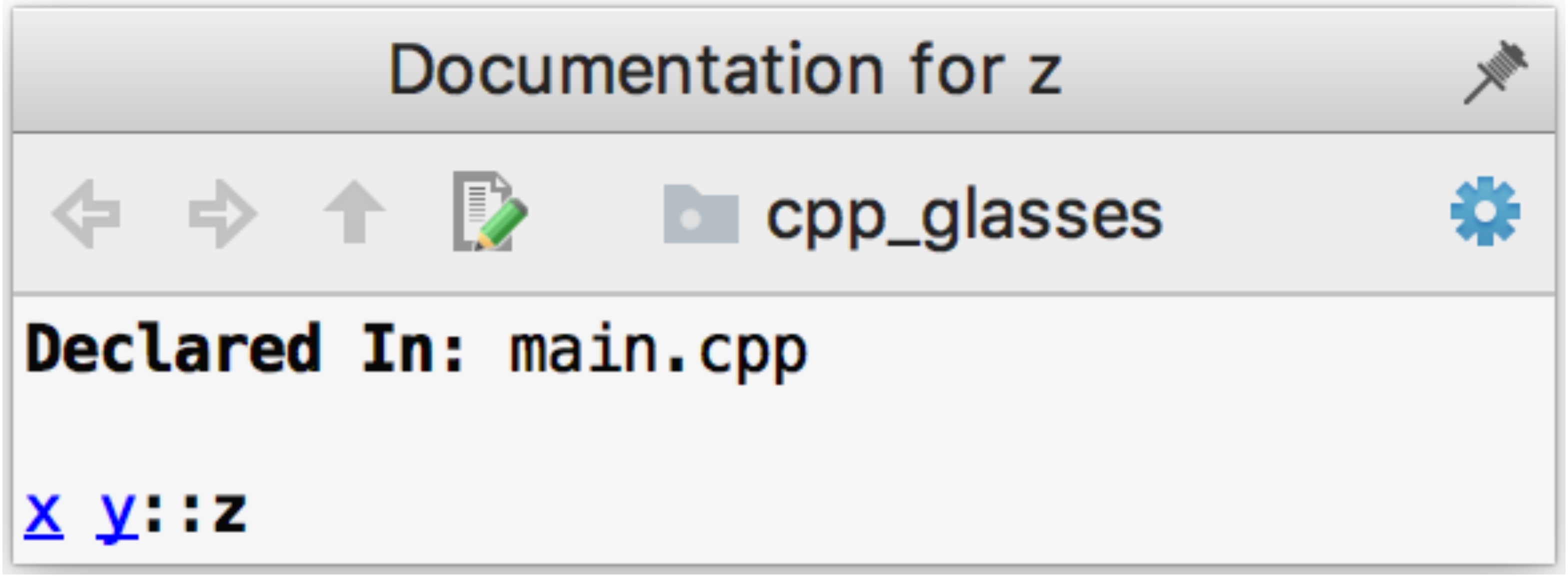
Documentation for y(x)

cpp_glasses

**Declared In:** main.cpp

y::**y**(x)

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

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



Documentation for z

cpp_glasses

**Declared In:** main.cpp

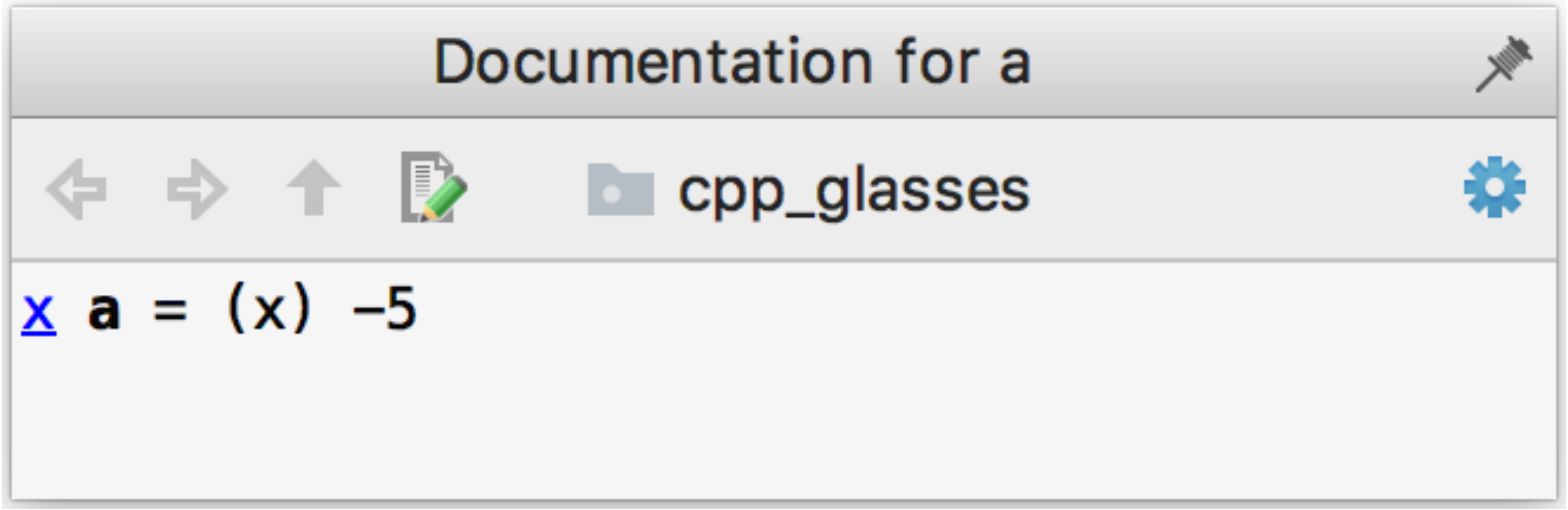
x y::**z**

# C++ game

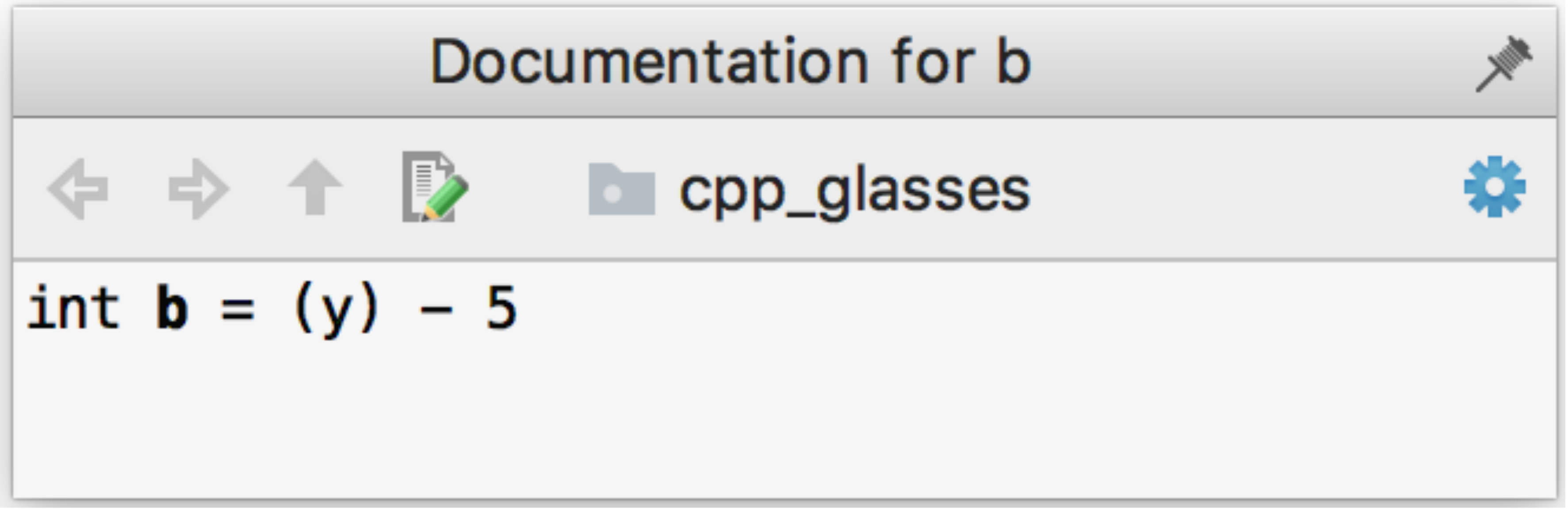
```cpp
void test() {
    struct x {
        x(int) { };
    };

    int y = 100;

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

# C++ game

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

    int y = 100;

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



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



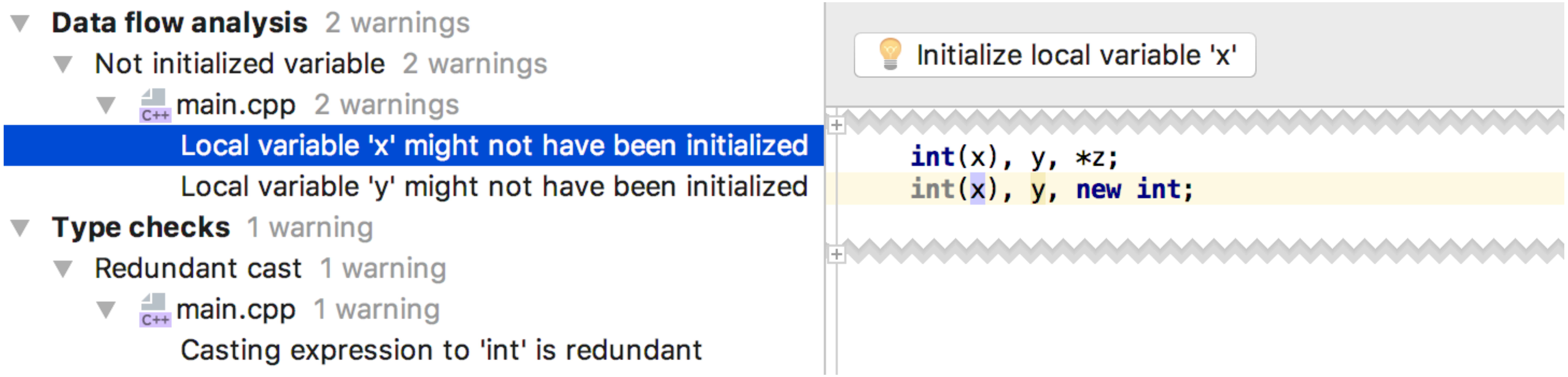
```
int b = (y) - 5
```

# C++ game

```
int(x), y, *z;
int(x), y, new int;
```

# C++ game

```
int(x), y, *z;
int(x), y, new int;
```



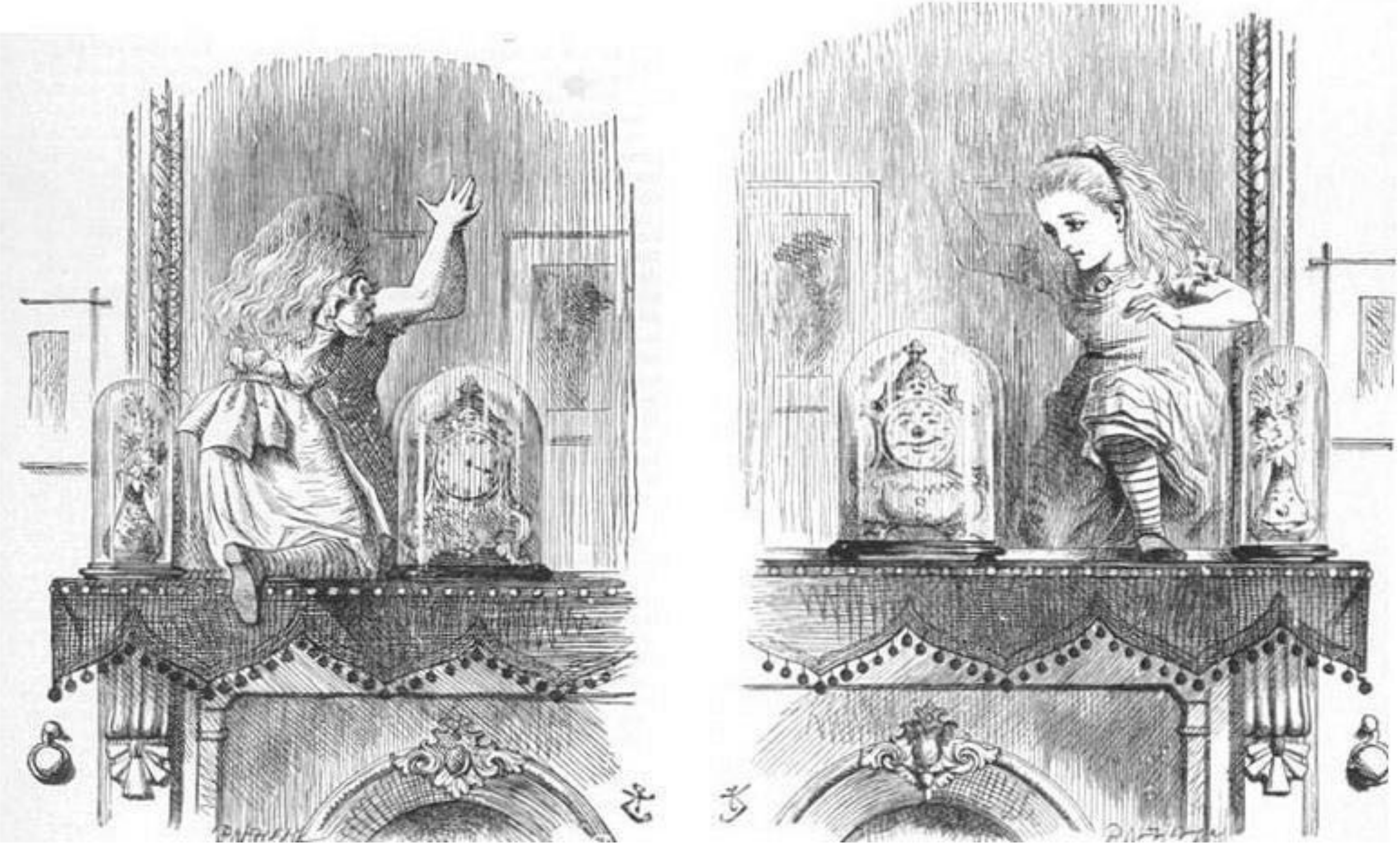
- Data flow analysis 2 warnings
  - Not initialized variable 2 warnings
    - main.cpp 2 warnings
      - Local variable 'x' might not have been initialized
      - Local variable 'y' might not have been initialized
- Type checks 1 warning
  - Redundant cast 1 warning
    - main.cpp 1 warning
      - Casting expression to 'int' is redundant

Initialize local variable 'x'

```
int(x), y, *z;
int(x), y, new int;
```

# Parsing and resolving C++

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



# Why do we need it?

- Highlight code on the fly
- Format (put spaces, new lines, etc. on the fly)
- Show completion on the fly

# Why do we need it?

```
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);
}
```

# Why do we need it?

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

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

# C++ game

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

    int y = 100;

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

# 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);
}
```

## Consequences in IDE

IDE needs parsing/resolve for:

- highlighting
- code completion
- code generation
- navigation
- find usages
- refactoring
- code analysis

# Example: code highlighting

Could we highlight with the lexer?

```
//-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 '> >')
}
```

# Example: code highlighting

Could we highlight with the lexer?

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

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

# Example: code highlighting

Could we highlight with the lexer?

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

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

# How do we cope

- Game of parsers
  - heuristics
  - fuzzy parsers
  - several parsers at a time
  - clang

## How do we cope

- Game of parsers
  - heuristics
  - fuzzy parsers
  - several parsers at a time

- Optimizations in parser:
  - Deferred resolve
  - Local reparse
  - Global reparse
  - Resolve contexts tables

# View on C++

1. if constexpr
2. Structured bindings
3. Concepts
4. Modules
5. Contracts
6. Reflection
7. Metaclasses
8. Modernize tools

# If constexpr

```cpp
#define EPSILON 0.000001

template <class T>
constexpr T absolute(T arg) {
    return arg < 0 ? -arg : arg;
}

template <class T>
constexpr std::enable_if_t<std::is_floating_point<T>::value, bool> close_enough(T a, T b) {
    return absolute(a - b) < static_cast<T>(EPSILON);
}

template <class T>
constexpr std::enable_if_t<!std::is_floating_point<T>::value, bool> close_enough(T a, T b) {
    return a == b;
}
```

# If constexpr

```
template <class T>
constexpr auto precision_threshold = T(0.000001);

template <class T>
constexpr bool close_enough(T a, T b) {

    if constexpr (std::is_floating_point_v<T>)
        return absolute(a - b) < precision_threshold<T>;
    else
        return a == b;

}
```

# Structured bindings

```
class Foo {
    int a, b;
    void bar(const Foo& other) {
        auto x = other.a; // Does this compile?
        auto y = other.b;
        // ...
    }
};
```

# Structured bindings

```
class Foo {
    int a, b;
    void bar(const Foo& other) {
        auto [x, y] = other; // Does this compile?
        // ...
    }
};
```

# Structured bindings

- Compilation error:
  - Cannot decompose non-public member 'a' of ‘Foo’ !

```
class Foo {
    int a, b;
    void bar(const Foo& other) {
        auto [x, y] = other;
        // ...
    }
};
```

# Structured bindings

- Compilation error:
  - Cannot decompose non-public member 'a' of ‘Foo’ !

- Why? In C++17, if you bind to the members of a class, there was a requirement that those members **must all be public**, and if they are in a base class, that needs to be a public base as well.

```
class Foo {
    int a, b;
    void bar(const Foo& other) {
        auto [x, y] = other;
        // ...
    }
};
```

# Structured bindings

- Compilation error:
  - Cannot decompose non-public member 'a' of ‘Foo’ !

- Why? In C++17, if you bind to the members of a class, there was a requirement that those members **must all be public**, and if they are in a base class, that needs to be a public base as well.

- What should be? You can bind to a member whenever it is accessible in the context of that declaration.

```
class Foo {
    int a, b;
    void bar(const Foo& other) {
        auto [x, y] = other;
        // ...
    }
};
```

# Concepts

Templates with proper interface – Concepts!

- clear semantic
- IDE analysis
- IDE performance improvement
- intellisense in templates

```
template<typename T> concept bool C =
    requires (T a, T b) {
      { a == b } -> bool;
    };
```

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

# 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 TU

```
//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);
}
```

# Modules!

- interface is clear
- postponed parsing
- less context dependent

```cpp
//my_module.ixx
module My;

export
int my_shiny_fun(int x) {
…
}

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

# Modules in std

Standard library (std2?)

- IDEs often build some kind of an interface
- Not easy for cross-platform tools working with multiple toolchains
- std2 = “legal hacks”

# Contracts

- on-the-fly code analysis

```
void foo(int* p)
[[expects: p != nullptr]]
{
    int x = *p;
    //…
}

int area(int height, int width)
{
    auto res = height * width;
    [[ensures: res > 0]]
    return res;
}
```

# Reflection

- Introspection

```
PersonData p = {31, 1234567, "Anastasia Kazakova"};

std::cout << "Person identifications comes with " <<
boost::pfr::tuple_size<PersonData>::value << " parameters. " << std::endl;
std::cout << boost::pfr::get<2>(p) << " is " << boost::pfr::get<0>(p) << std::endl;
```

# Reflection

- Introspection
- Code generation

# Metaclasses

```
interface Shape {
    int area() const;
    void scale_by(double factor);
};
```

```
$class interface {
    constexpr {
        compiler.require($interface.variables().empty(),
                         "interfaces may not contain data");
        for... (auto f : $interface.functions()) {
            compiler.require(!f.is_copy() && !f.is_move(),
                "interfaces may not copy or move; consider a"
                " virtual clone() instead");
            if (!f.has_access()) f.make_public();
            compiler.require(f.is_public(),
                "interface functions must be public");
            f.make_pure_virtual();
        }
    }
    virtual ~interface() noexcept { }
};
```

```
struct Shape {
    virtual int area() const = 0;
    virtual void scale_by(double factor) = 0;
    virtual ~Shape() noexcept {
    }
}
```

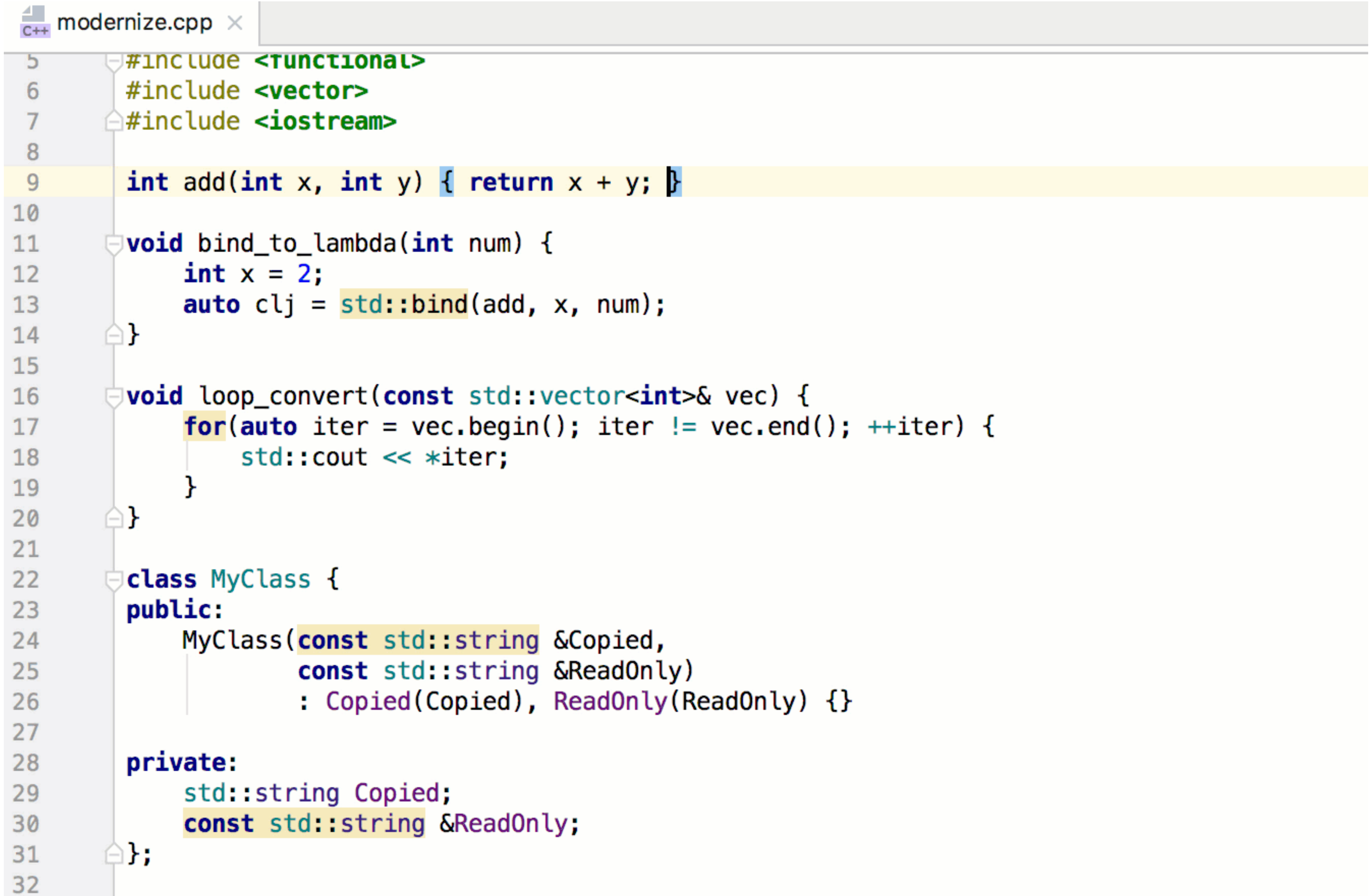
# Thoughts on Metaclasses in IDEs

- One way
  - allow all methods
  - treat metaclasses definitions as text
- Better way
  - check conditions
  - parse metaclass definition
  - complete metaclasses in hierarchical definitions
  - show previews!

# Modernize tools

- C++ Core Guidelines support
- Clang-Tidy
- Any other

# Modernize tools



modernize.cpp

```cpp
#include <functional>
#include <vector>
#include <iostream>

int add(int x, int y) { return x + y; }

void bind_to_lambda(int num) {
    int x = 2;
    auto clj = std::bind(add, x, num);
}

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

class MyClass {
public:
    MyClass(const std::string &Copied,
            const std::string &ReadOnly)
            : Copied(Copied), ReadOnly(ReadOnly) {}

private:
    std::string Copied;
    const std::string &ReadOnly;
};
```

# References

- Bjarne Stroustrup, Writing Good C++14
  - [CppCon 2015] https://www.youtube.com/watch?v=1OEu9C51K2A
- JetBrains, Developer Ecosystem Survey 2017
  - https://www.jetbrains.com/research/devecosystem-2017/
- Bartlomiej Filipek's blog on *if constexpr*
  - http://www.bfilipek.com/2018/03/ifconstexpr.html
- Antony Polukhin, reflection library
  - https://github.com/apolukhin/magic_get
- Jackie Kay, Practical (?) Applications of Reflection
  - [C++Now 2017] https://www.youtube.com/watch?v=JrOJ012XxNg
- Herb Sutter, Metaclasses: Thoughts on generative C++
  - https://herbsutter.com/2017/07/26/metaclasses-thoughts-on-generative-c/
- Ilya Biryukov, How compiler frontend is different from what IDE needs?
  - [LLVM Developers' Meeting US 2016] https://www.youtube.com/watch?v=CZg2d3LoL84
  - https://www.dropbox.com/s/tqed22izc4wd5es/spbusergroup.pdf?dl=0

# Thank you
for your attention

—

Questions?