



---

# C++ Tutorial

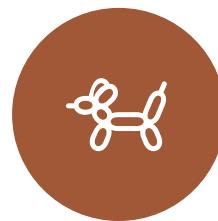


# Talk Outline

---



BRIEF TOUR  
OF C++



TEMPLATES



UNDEFINED  
BEHAVIOUR

# Additional Resources

---

## [isocpp.org/get-started](https://isocpp.org/get-started)

- Recommended book list
- high-level explanations, tutorials, and design guidance

## [cppreference.com/w/](https://cppreference.com/w/)

- Language and standard library documentation

## [coliru.stacked-crooked.com](https://coliru.stacked-crooked.com)

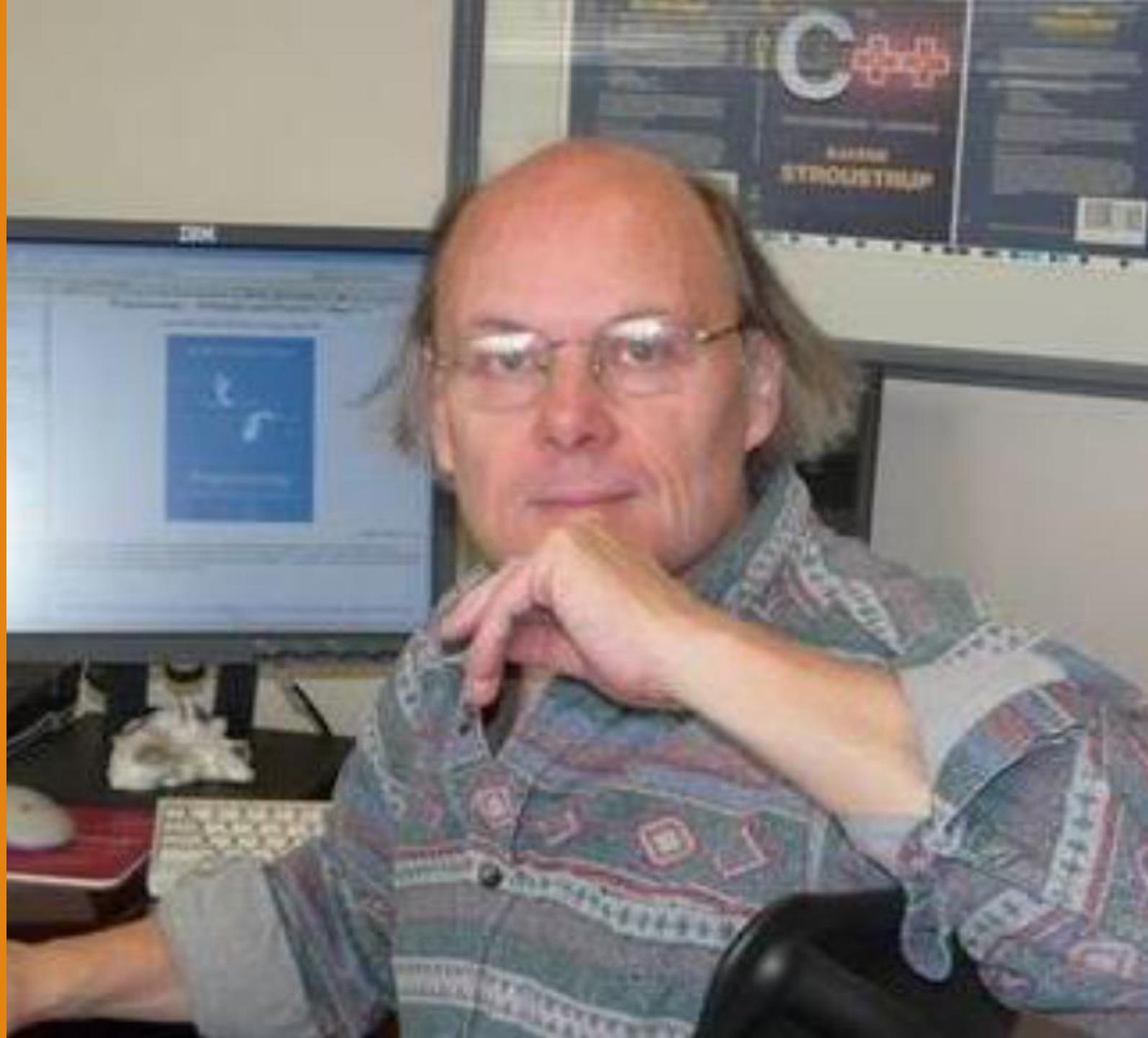
- Free online compiler (great for small exercises)



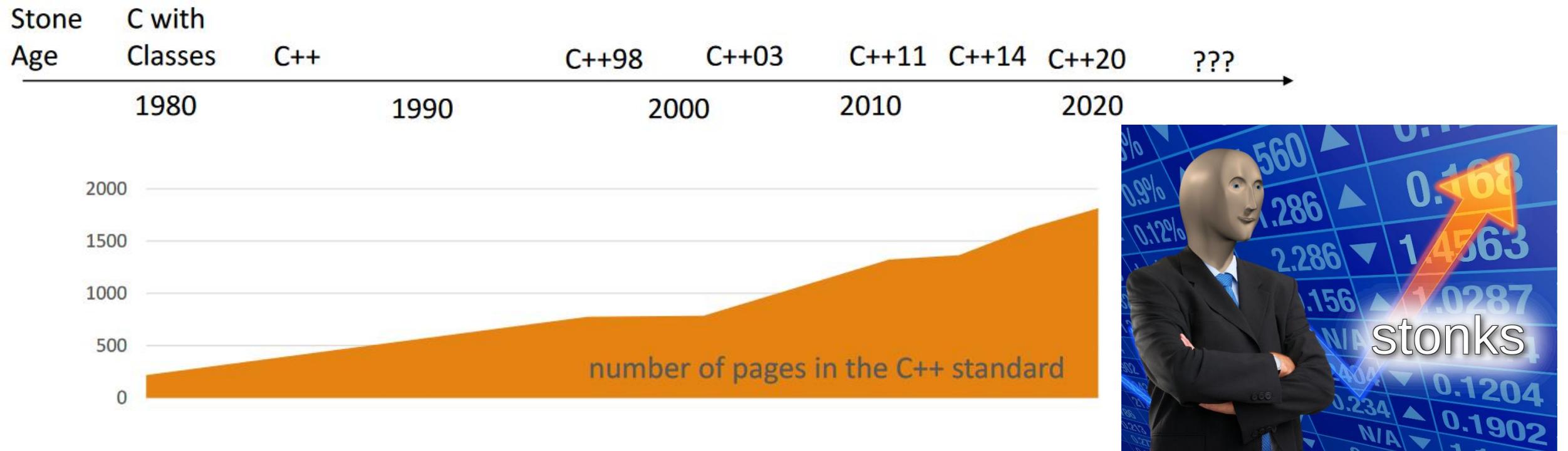
# A Brief Tour of C++

---

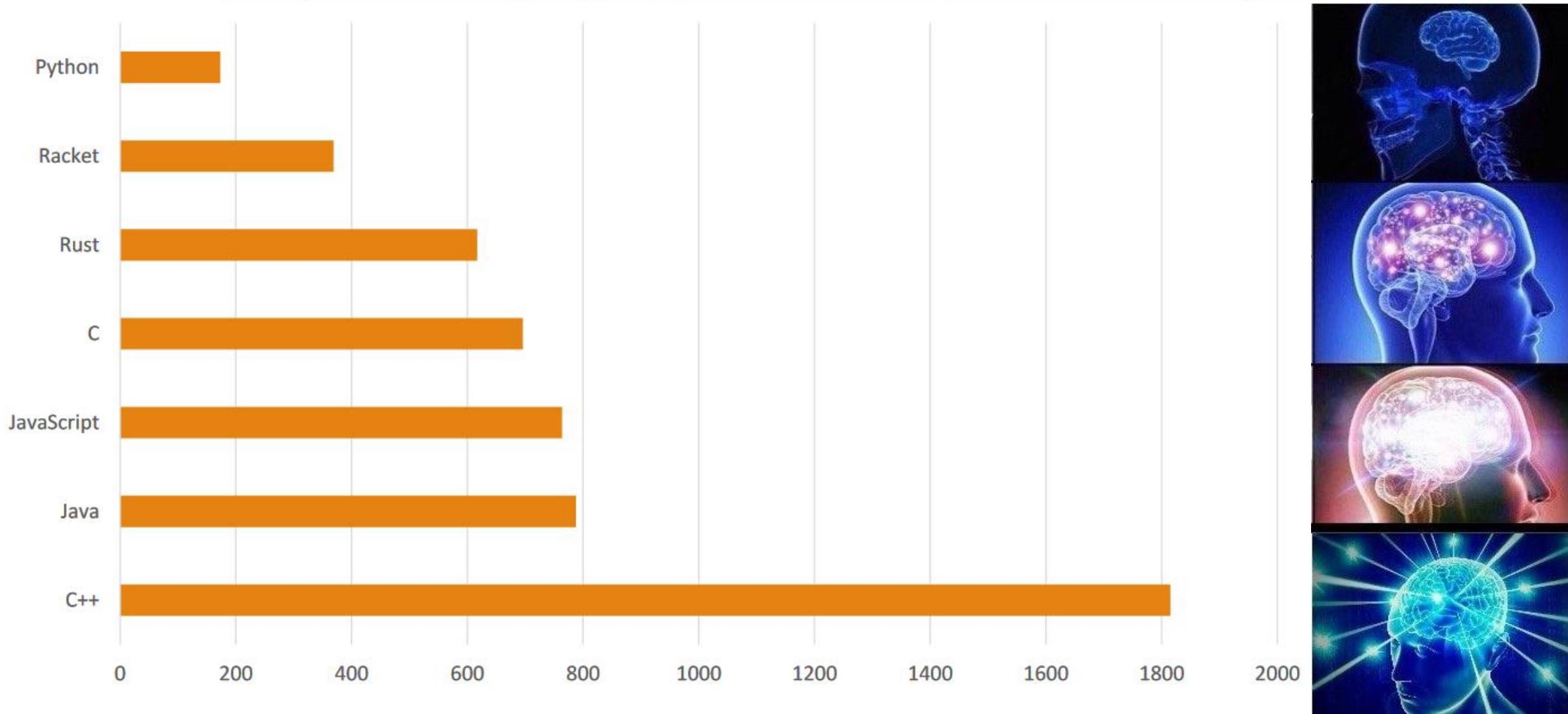
C++ **began** being invented in 1979 by Danish computer scientist **Bjarne Stroustrup** (pictured right)



# C++ is Not Done Being Invented



# Length of Language Specification (Number of Pages)



# Why do C++ programmers like C++?

---

- **Runtime performance**

- Zero-cost abstractions
- Compiler optimizations
- Easy and efficient resource management
- Compile-time programming (for advanced users)



- **Type Safety**

- Many bugs are eliminated at compile time \*



\*

- **Expressiveness**

- Many diverse tools are provided by the C++ language
- Many styles of programming are possible and can be mixed
  - Generic, object-oriented, functional, imperative, procedural, compile-time, template meta-programming, etc



\* but not all

# Why don't C++ programmers like C++?

---

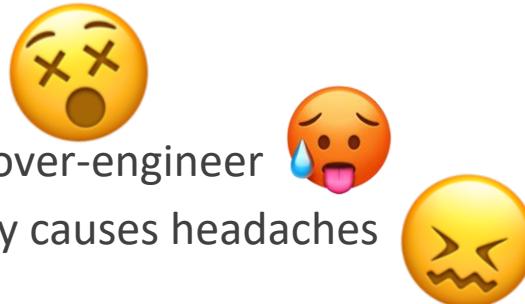
- **Undefined Behaviour**

- C++ gives you the freedom to hurt yourself
- **You are responsible** for preventing bugs
- **The language does not protect you from yourself**



- **Complexity**

- The C++ language is **huge**
- C++ programmers readily over-engineer
- Reasoning about C++ easily causes headaches



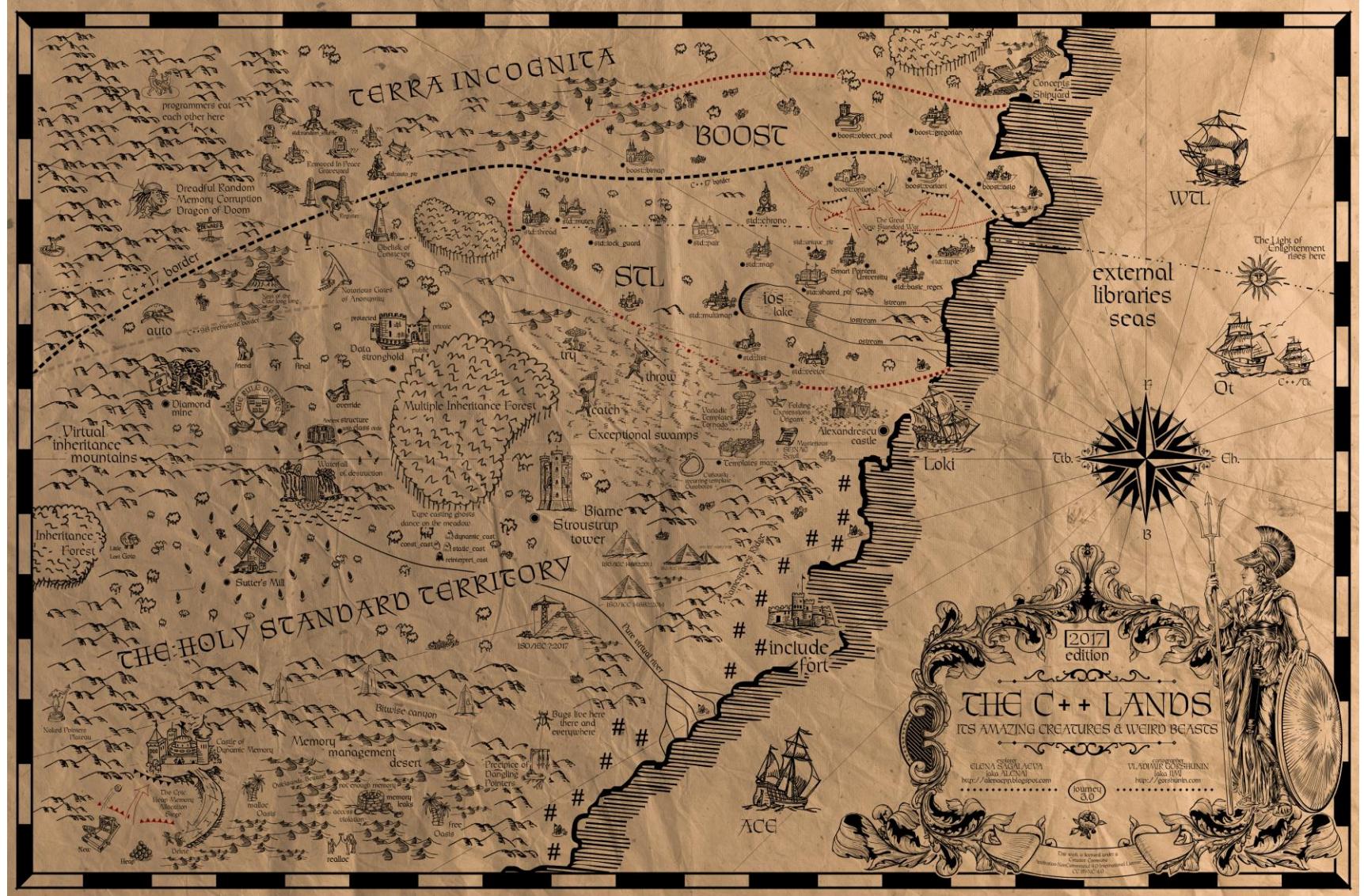
- **Compilation speed**

- Begin a C++ compiler is not easy



# C++ is a very diverse landscape

Image credit:  
<http://fearlesscoder.blogspot.com/2017/02/the-c17-lands.html>



# C++ is a very diverse landscape

Different communities use different subsets of the language in different ways for different goals

Image credit:  
<http://fearlesscoder.blogspot.com/2017/02/the-c17-lands.html>

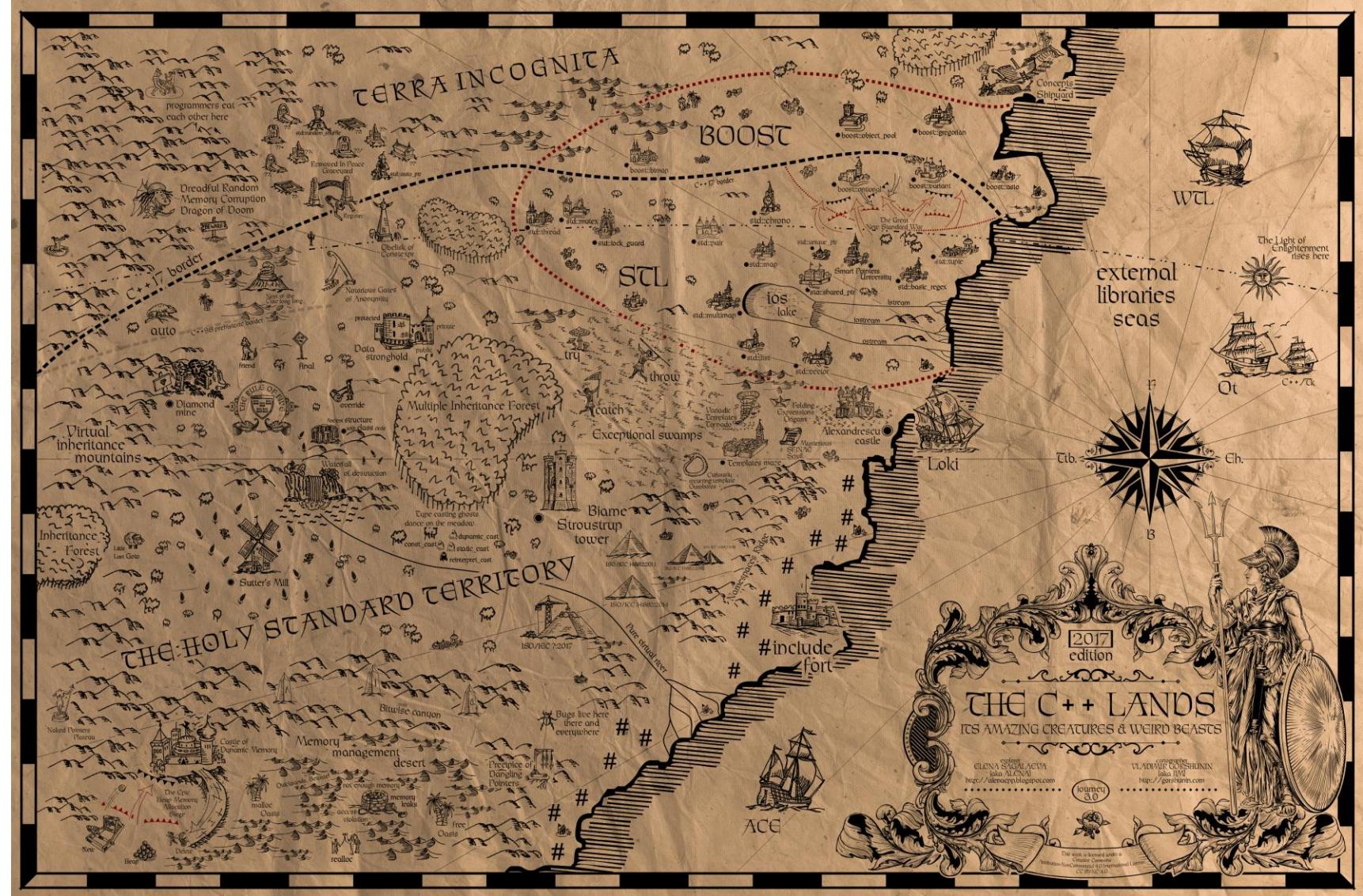


# C++ is a very diverse landscape

Different communities use different subsets of the language in different ways for different goals

Video game programming is **just one** of countless ways of using C++

Image credit:  
<http://fearlesscoder.blogspot.com/2017/02/the-c17-lands.html>

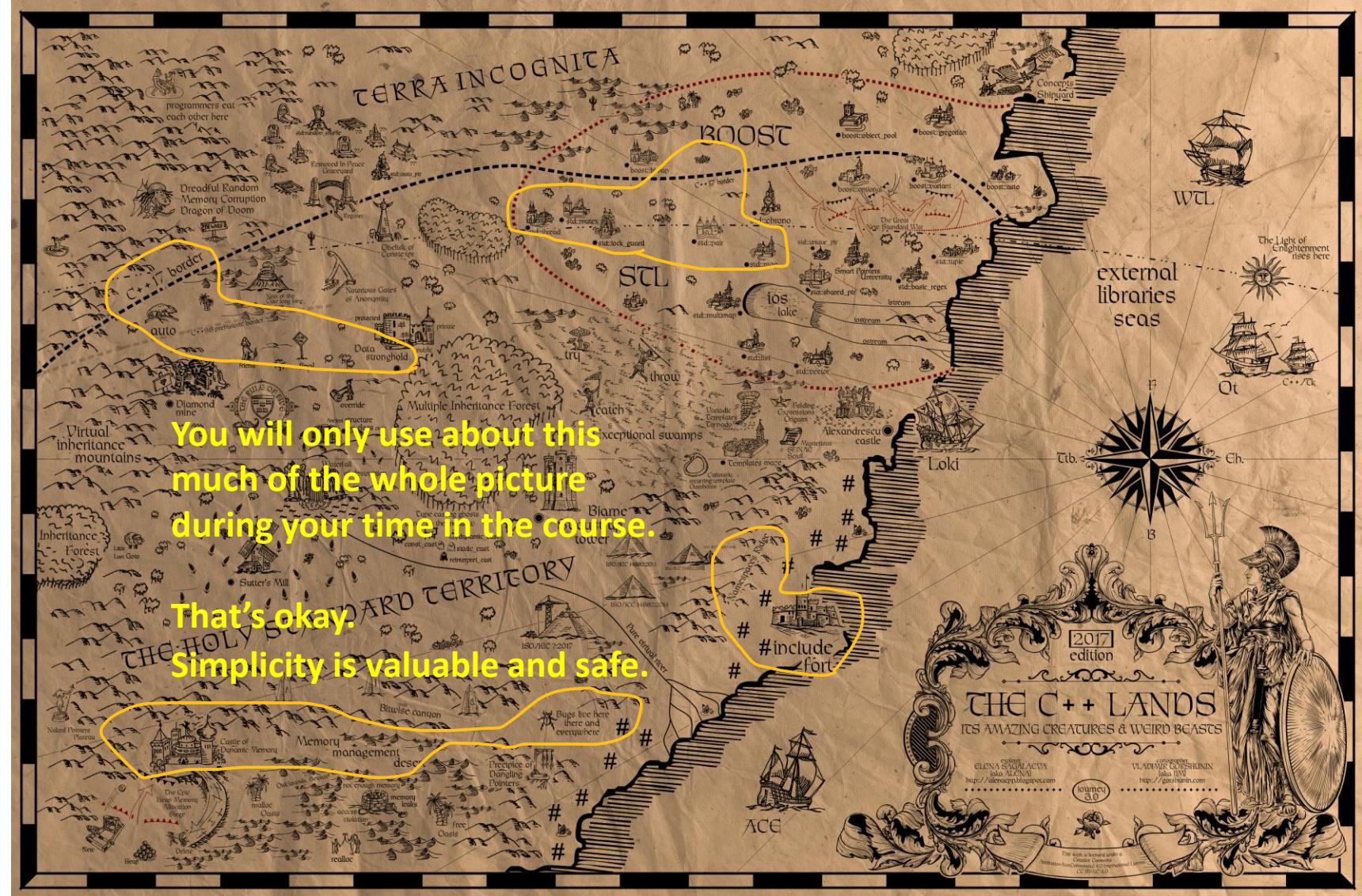


# C++ is a very diverse landscape

Different communities use different subsets of the language in different ways for different goals

Video game programming is **just one** of countless ways of using C++

Image credit:  
<http://fearlesscoder.blogspot.com/2017/02/the-c17-lands.html>





this has been

# A Brief Tour of C++

---

thank you for watching

# C++ Templates

---

# Avoiding Manual Code Duplication

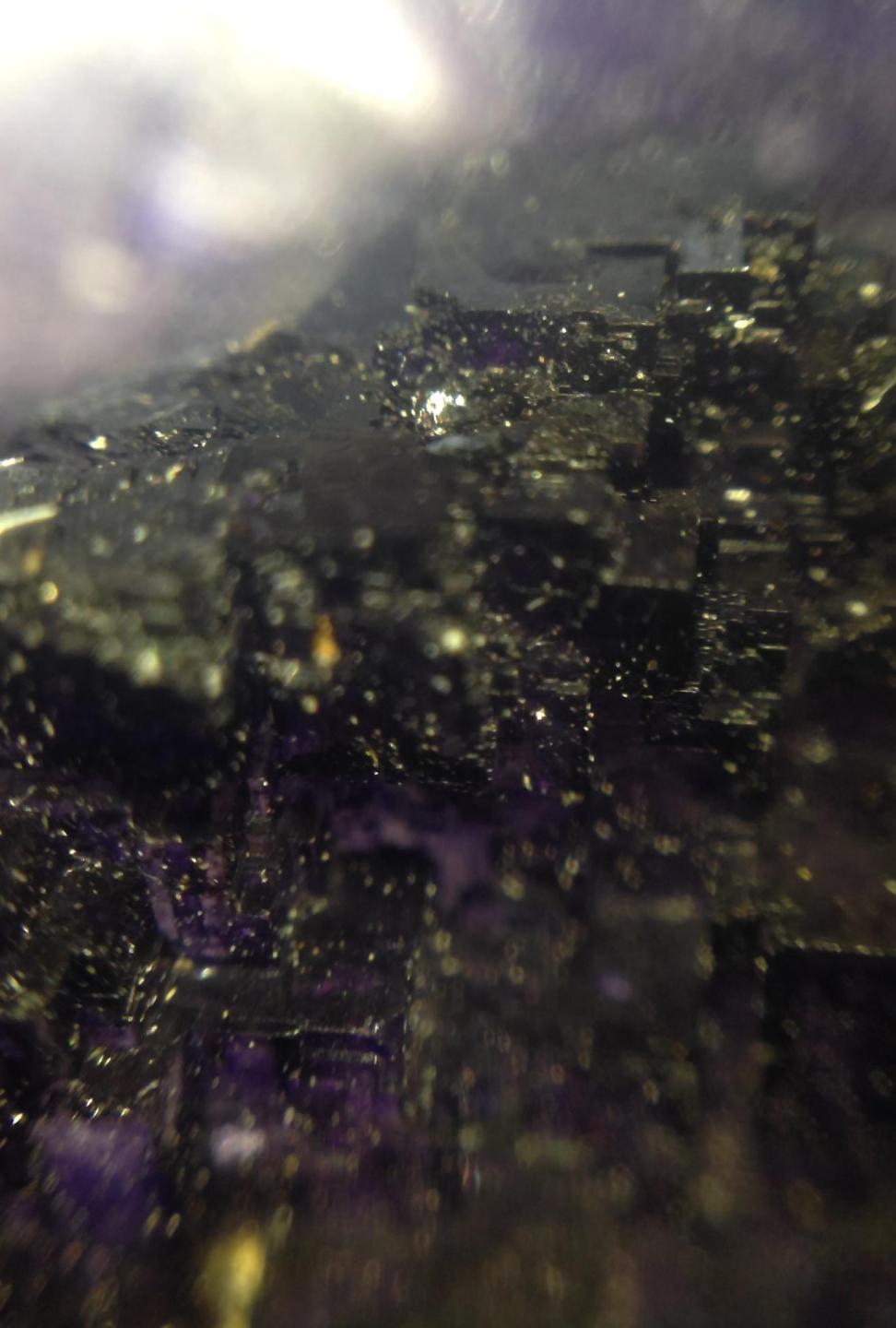
Does this code style look familiar to you?

```
2727     int add_int(int x, int y) {  
2728         int result = x + y;  
2729         return result;  
2730     }  
2731  
2732     double add_double(double x, double y) {  
2733         double result = x + y;  
2734         return result;  
2735     }  
2736  
2737     std::string add_string(std::string x, std::s  
2738         std::string result = x + y;  
2739         return result;  
2740     }  
2741  
2742     float add_float(float x, float y) {  
2743         float result = x + y;  
2744         return result;  
2745     }
```

# Templates to the rescue!

Automated code duplication!

```
2727 template<typename T>
2728     T add(T x, T y) {
2729         T result = x + y;
2730         return result;
2731     }
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
```



# C++ templates versus other languages

---

# member access in Python

**Nearly everything** is  
checked at **runtime!**

Lots of testing required



```
def foo(x):
    print(x.bar)

class A:
    def __init__(self):
        self.bar = "Blab"

a = A()
b = 99
foo(a)
foo(b)
```

# member access in C++ templates

Templates are checked  
at compile time!



```
template<typename T>
void foo(T t){
    std::cout << t.bar << std::endl;
}

struct A {
    std::string bar = "Blab";
};

int main(){
    auto a = A{};
    auto b = 99;
    foo(a);
    // foo(b); ERROR: request for member 'bar' in 't',
    // which is of non-class type 'int'

    return 0;
}
```

# Generic factory functions in Java

Only one function is generated!

Types are erased 😞

Simple things are impossible 🤦

```
public static <T> T create() {  
    T t = new T();  
    return t;  
}
```

```
Main.java:11: error: unexpected type  
        T t = new T();  
                           ^  
       required: class  
      found:    type parameter T  
      where T is a type-variable:  
          T extends Object declared in method <T>create()
```

## Template factory functions in C++

Types can be provided  
explicitly for **great good**

Our ECS system uses  
this extensively  
**Take a look** 😊

```
template<typename T>
T create(){
    auto t = T{};
    return t;
}

int main(){
    auto i = create<int>();
    auto d = create<double>();
    auto s = create<std::string>();

    return 0;
}
```

# Templates in C++

---

C++ is **statically typed**, and **all types must be known at compile time**

So how do templates work in C++?

- **Automated code duplication!** (*technically called monomorphization*)

Each time you provide a template function/class with a different type, **a different function/class is generated by the compiler!**

In other words, a template **is not** a class or function **until** you specify a type  
(*technically called instantiation*)

# Benefits of Templates in C++

---

- **Type checking**

- The compiler can inspect types and perform all the normal safety and correctness checks

- **Optimization**

- The compiler can generate faster code that is specific to each type

- **Expressivity**

- Templates are *extremely* powerful at doing many different things

# Downsides of Templates in C++

---

- **Slow compile times**

- Templates add extra work for the compiler

- **Code bloat**

- “automatic code duplication” is exactly that – the size of compiled programs increases for every new template instantiation

- **Complexity**

- Template code compiles differently from normal code – understanding and fixing errors can be difficult





# The Dark Side of C++

---

## *Undefined Behaviour*

# C++ is not safe

---

- C++ lets you break the rules of the language
- When you break the rules, *anything* can happen. 
- A good C++ programmer knows how not to break the rules

```
1 #include <iostream>
2
3 int main() {
4     std::cout << "Start ---" << std::endl;
5     char ch; // Oops! Forgot to initialize :-
6     std::cout << ch << std::endl;
7     std::cout << "Finish ---" << std::endl;
8     return 0;
9 }
10
11
12
13
14
15
16
```

```
Start ---
Finish ---
```

Undefined Behaviour  
means:  
*Your code  
may do  
nothing*

---

```
1 #include <iostream>
2
3 int main(){
4     int i;
5     double d;
6     bool b;
7     uint8_t u;
8     std::cout << i << '\n';
9     std::cout << d << '\n';
10    std::cout << b << '\n';
11    std::cout << u << '\n';
12 }
```

```
0
0
0
```

Undefined Behaviour means:  
Your code  
*may do what you believe it should*

---

```
1 #include <iostream>
2
3 int main(){
4     int i;
5     double d;
6     bool b;
7     uint8_t u;
8     std::cout << i << '\n';
9     std::cout << d << '\n';
10    std::cout << b << '\n';
11    std::cout << u << '\n';
12 }
```

```
0
6.95255e-310
0
```

Undefined Behaviour  
means:

Your code  
*may do what*  
you believe it  
should

---

*...until you change your  
compiler settings*

```
1 #include <iostream>
2
3 int main(){
4     int i;
5     double d;
6     bool b;
7     uint8_t u;
8     std::cout << i << '\n';
9     std::cout << d << '\n';
10    std::cout << b << '\n';
11    std::cout << u << '\n';
12 }
```

```
718172376
0
0
```

Undefined Behaviour means:  
Your code  
*may do what you believe it should*

---

*...until you change your compiler settings  
...or try a different compiler*

```
// Entry point
int main() {
    int* ptr = nullptr;
    std::cout << *ptr; X
```

### Exception Thrown

Exception thrown: read access violation.  
**ptr** was **nullptr**.

#### [Copy Details](#)

##### [▲ Exception Settings](#)

Break when this exception type is thrown

Except when thrown from:

Undefined Behaviour means:

YOUR CODE MAY CRASH WITH A HELPFUL ERROR MESSAGE

```
48
49
50
51    // Entry point
52    int main() {
53        return 0;
54    }
55
56
57
```

Undefined Behaviour means:

YOUR CODE MAY CRASH FOR NO EXPLAINABLE REASON

xhash

salmon

std::Hash&lt;\_Traits&gt;

\*\_Find\_hint&lt;\_Keyty&gt;(const \_Nodeptr \_Hint, const \_Keyty &amp; \_Ke

```

1650     protected:
1651         template <class _Keyty>
1652         _NODISCARD _Hash_find_last_result<_Nodeptr> _Find_last(const _Keyty& _Keyval, const size_t _Hashval) const {
1653             // find the insertion point for _Keyval and whether an element identical to _Keyval is already in the container
1654             const size_type _Bucket = _Hashval & _Mask;
1655             _Nodeptr _Where = _Vec._Mypair._Myval2._Myfirst[(_Bucket << 1) + 1]._Ptr; X
1656             const _Nodeptr _End = _List._Mypair._Myval2._Myhead;
1657             if (_Where == _End) {
1658                 return {_End, _Nodeptr{}};
1659             }
1660
1661             const _Nodeptr _Bucket_lo = _Vec._Mypair._Myval2._Myfirst[_Bucket << 1]._Ptr;
1662             for (;;) {
1663                 // Search backwards to maintain sorted [_Bucket_lo, _Bucket_hi] when !_StandardOrdering
1664                 if (!_Traitsobj(_Keyval, _Traits::_Kfn(_Where->_Myval))) {
1665                     if !_CONSTEXPR_IF (!_Traits::_Standard) {
1666                         if (_Traitsobj(_Traits::_Kfn(_Where->_Myval), _Keyval)) {
1667                             return {_Where->_Next, _Nodeptr{}};
1668                         }
1669                     }
1670
1671                     return {_Where->_Next, _Where};
1672                 }
1673
1674                 if (_Where == _Bucket_lo) {
1675                     return {_Where, _Nodeptr{}};

```

## Exception Thrown

Exception thrown: read access violation.

this-&gt;\_Vec.\_Mypair.\_Myval2.\_Myfirst was 0x110111011011A.

## Copy Details

## ▲ Exception Settings

 Break when this exception type is thrown

Except when thrown from:

 salmon.exe[Open Exception Settings](#) | [Edit Conditions](#)

# Undefined Behaviour means:

YOUR CODE MAY CRASH FOR NO EXPLAINABLE REASON

```
1 #include <iostream>
2
3 bool fn() {
4     // Oops! Forgot to return :-)
5 }
6
7 int main() {
8     std::cout << "Start ---" << std::endl;
9     if (fn()) {
10         std::cout << "fn() returned true\n";
11     } else {
12         std::cout << "fn() returned false\n";
13     }
14     std::cout << "Finish ---" << std::endl;
15     return 0;
16 }
```

```
Start ---
```

```
Start ---
```

```
bash: line 7: 1737 Segmentation fault      (core dumped) ./a.out
```

Undefined Behaviour means:  
Your code may run and do something completely unexplainable

---

```
3963 #ifndef OPENSSL_NO_HEARTBEATS
3964 int
3965 tls1_process_heartbeat(SSL *s)
3966 {
3967     unsigned char *p = &s->s3->rrec.data[0], *pl;
3968     unsigned short hbtype;
3969     unsigned int payload;
3970     unsigned int padding = 16; /* Use minimum padding */
3971
3972     /* Read type and payload length first */
3973     hbtype = *p++;
3974     n2s(p, payload);
3975     pl = p;
3976
3977     if (s->msg_callback)
3978         s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
3979                         &s->s3->rrec.data[0], s->s3->rrec.length,
3980                         s, s->msg_callback_arg);
```

This is a pointer to an array



This *should* be the length  
of that array. It is not.

## Undefined Behaviour means:

YOUR CODE MIGHT RUN FINE, BUT  
HACKERS CAN STEAL YOUR  
PASSWORDS

The code will crash if  
I read an array out of  
bounds, right? 😬 😬 😬

```
3963 #ifndef OPENSSL_NO_HEARTBEATS
3964 int
3965 tls1_process_heartbeat(SSL *s)
3966 {
3967     unsigned char *p = &s->s3->rrec.data[0], *pl;
3968     unsigned short hbtype;
3969     unsigned int payload;
3970     unsigned int padding = 16; /* Use minimum padding */
3971
3972     /* Read type and payload length first */
3973     hbtype = *p++;
3974     n2s(p, payload);
3975     pl = p;
3976
3977     if (s->msg_callback)
3978         s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
3979                         &s->s3->rrec.data[0], s->s3->rrec.length,
3980                         s, s->msg_callback_arg);
```

This is a pointer to an array



This *should* be the length  
of that array. It is not.



# Undefined Behaviour means:

YOUR CODE MIGHT RUN FINE, BUT  
HACKERS CAN STEAL YOUR  
PASSWORDS

answer:  
Hard maybe

# The Heartbleed Bug



The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet. SSL/TLS provides communication security and privacy over the Internet for applications such as web, email, instant messaging (IM) and some virtual private networks (VPNs).

The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to encrypt the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users.

Undefined  
Behaviour  
means:

YOUR CODE MIGHT RUN FINE, BUT  
HACKERS CAN STEAL YOUR  
PASSWORDS

Reading an array out of bounds  
does not guarantee anything



# Definitions of Undefined Behaviour

---

- “Renders the entire program meaningless if certain rules of the language are violated.” [1]
- “There are no restrictions on the behaviour of the program” [1]
- “Compilers are not required to diagnose undefined behaviour [...], and the compiled program is not required to do anything meaningful.” [1]
- “Because **correct C++ programs are free of undefined behaviour**, compilers may produce unexpected results when a program that actually has UB is compiled with optimization enabled” [1]
- If a program encounters UB when given a set of inputs, there are no requirements on its behaviour, “**not even with regard to operations preceding the first undefined operation**” [2]



[1] <https://en.cppreference.com/w/cpp/language/ub>

[2] C++20 Working Draft, Section 4.1.1.5

# Undefined Behaviour in simpler terms

---

If you do something wrong in C++, literally anything can happen your code runs.

This includes:

- Your code runs and does nothing 🤔
- Your code runs as you expect it to 😊
- Your code crashes with a helpful error message 😊
- Your code crashes for no explainable reason 😰
- Your code runs and does something *weird* 😮
- Your code runs as you expect it to, but fails later at the worst possible moment 😱
- Your code passes all tests, but hackers can steal your passwords 😭
- Demons will come flying out of your nose



# Common causes of Undefined Behaviour

- Reading from an **uninitialized variable**
- Reading an array **out of bounds**
- **Dereferencing a null pointer**
- **Dereferencing an invalid pointer** that doesn't point to any object
- Dereferencing a pointer to an object with an incompatible type
- Forgetting to return a value from a `non-void` function
- `delete`-ing dynamically allocated memory twice

Many infamous software bugs and vulnerabilities are due to  
Undefined Behaviour!

# Why does C++ have Undefined Behaviour? This sounds **terrible**!

---

- Undefined Behaviour *simplifies* compilation (and language design)
  - Compilers can (and do!) assume that Undefined Behaviour never happens
  - Compilers don't need to do extra work to ensure safety
  - The concept of Undefined Behaviour was inherited from C
  - Detecting all types of Undefined Behaviour in C++ is **impossible**.

# What Undefined Behaviour means for you

---

- **The C++ language cannot be learned by trial-and-error.**
- Read **good C++ books** and **reliable documentation** to best learn to avoid Undefined Behaviour
  - See <https://isocpp.org/get-started> for introductions, examples, tutorials, guidance, and books
  - See <https://en.cppreference.com/w/> for language and standard library documentation
- If you **write safe code** to begin with, you will **waste less time debugging** 
- **Read compiler warnings** and **increase your compiler's warning level**
  - We've already turned on extra warnings in the starter code for you
- **When in doubt**, write your own safety checks

# Avoiding Undefined Behaviour with safety checks

## Using the assert() macro

Documentation:  
[https://en.cppreference.com  
/w/cpp/error/assert](https://en.cppreference.com/w/cpp/error/assert)

Use `assert (condition)` to test your assumptions

- In **debug mode**, halts the program *immediately* with a helpful error message if *condition* is false
  - **Use your debugger!** It will take you right to the problem!
- In **release mode**, does **nothing**.
  - Useful for optimization (fast code)
  - **Not useful for input validation!**
- Use `assert` to **test your assumptions** about your own code and to find **unrecoverable errors**
- For problems that can be fixed, such as a user pressing the wrong button, use a different error reporting mechanism (e.g. returning error codes)

```
1 #include <iostream>
2
3
4 int main() {
5     int x = 88;
6     int* ptr = nullptr;
7     for (int i = 0; i < 100 && !ptr; ++i) {
8         for (int j = 0; j < 100 && !ptr; ++j) {
9             if (i*i + j*j == x) {
10                 ptr = &x;
11             }
12         }
13     }
14     std::cout << *ptr << std::endl;
15     return 0;
16 }
17
```

```
bash: line 7: 29756 Segmentation fault      (core dumped) ./a.out
```

```
1 #include <iostream>
2 #include <cassert>
3
4 int main() {
5     int x = 88;
6     int* ptr = nullptr;
7     for (int i = 0; i < 100 && !ptr; ++i) {
8         for (int j = 0; j < 100 && !ptr; ++j) {
9             if (i*i + j*j == x) {
10                 ptr = &x;
11             }
12         }
13     }
14     assert(ptr != nullptr);
15     std::cout << *ptr << std::endl;
16     return 0;
17 }
```

```
a.out: main.cpp:14: int main(): Assertion `ptr != nullptr' failed.
bash: line 7: 30544 Aborted                  (core dumped) ./a.out
```

```
1 #include <iostream>
2 #include <vector>
3
4
5 class A {
6 public:
7     A() : m_items{1, 2, 3, 5, 7, 11, 13, 17, 19} {}
8 int getItem(int index){
9
10    return m_items[index];
11 }
12 private:
13     std::vector<int> m_items;
14 };
15
16 int main() {
17     auto a = A{};
18     std::cout << a.getItem(0) << std::endl;
19     std::cout << a.getItem(13) << std::endl;
20     return 0;
21 }
```

1  
0



```
1 #include <iostream>
2 #include <vector>
3 #include <cassert>
4
5 class A {
6 public:
7     A() : m_items{1, 2, 3, 5, 7, 11, 13, 17, 19} {}
8     int getItem(int index){
9         assert(index >= 0 && index < m_items.size());
10        return m_items[index];
11    }
12 private:
13     std::vector<int> m_items;
14 };
15
16 int main() {
17     auto a = A{};
18     std::cout << a.getItem(0) << std::endl;
19     std::cout << a.getItem(13) << std::endl;
20     return 0;
21 }
```

```
1
a.out: main.cpp:9: int A::getItem(int): Assertion `index >= 0 && index < m_items.size()' failed.
bash: line 7: 32109 Aborted                  (core dumped) ./a.out
```

```
1 #include <iostream>
2 #include <cmath>
3 #include <exception>
4
5 int main() {
6     auto x = 0.0;
7     std::cin >> x;
8     std::cout << "x is " << x << std::endl;
9
10
11
12     std::cout << "sqrt(x) is " << std::sqrt(x) << std::endl;
13     return 0;
14 }
15
```

```
x is -22
sqrt(x) is -nan
```

```
1 #include <iostream>
2 #include <cmath>
3 #include <exception>
4
5 int main() {
6     auto x = 0.0;
7     std::cin >> x;
8     std::cout << "x is " << x << std::endl;
9     if (x < 0.0) {
10         throw std::runtime_error("Oops! Please enter a non-negative number, thanks! :-)");
11     }
12     std::cout << "sqrt(x) is " << std::sqrt(x) << std::endl;
13     return 0;
14 }
```

```
x is -22
terminate called after throwing an instance of 'std::runtime_error'
  what(): Oops! Please enter a non-negative number, thanks! :-)
bash: line 7: 3873 Done                      echo "-22"
                                ^~~~~~
3874 Aborted                  (core dumped) | ./a.out
```

```
24 int main() {
25     showLoginPrompt();
26     if (getUserCommand() == DatabaseAction::Drop){
27         auto uc = getUserCredentials();
28         std::cout << "LOG: " << uc << " wants to delete the database" << std::endl;
29         assert(uc == User::Admin);
30         deleteTheEntireDatabase();
31     }
32     return 0;
33 }
```

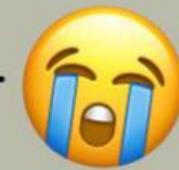
```
Welcome to Database Management System (Development Version 9.04.12)
LOG: Guest wants to delete the database
a.out: main.cpp:29: int main(): Assertion `uc == User::Admin' failed.
bash: line 7: 14871 Done
    14872 Aborted                  (core dumped) | ./a.out
```

```
24 int main() {
25     showLoginPrompt();
26     if (getUserCommand() == DatabaseAction::Drop){
27         auto uc = getUserCredentials();
28         std::cout << "LOG: " << uc << " wants to delete the database" << std::endl;
29         assert(uc == User::Admin);
30         deleteTheEntireDatabase();
31     }
32     return 0;
33 }
```

```
Welcome to Database Management System (Release 10.05.71)
```

```
LOG: Guest wants to delete the database
```

```
LOG: The database was successfully deleted. Everything is gone.
```



```
24 int main() {
25     try {
26         showLoginPrompt();
27         if (getUserCommand() == DatabaseAction::Drop){
28             auto uc = getUserCredentials();
29             std::cout << "LOG: " << uc << " wants to delete the database" << std::endl;
30             if (uc != User::Admin) {
31                 throw AuthenticationError{};
32             }
33             deleteTheEntireDatabase();
34         }
35     } catch (const std::exception& e){
36         std::cout << "ERROR: " << e.what() << std::endl;
37     }
38     return 0;
39 }
```

Welcome to Database Management System (Release 10.05.71)

LOG: Guest wants to delete the database

ERROR: Authentication failed



# *In conclusion:*

- C++ is not safe
  - *Undefined Behaviour* is weird
  - *Undefined Behaviour* must be avoided
  - Learn before you code
  - Safety checks make life better
-

