



C++20

C++ ready for Python programmers?

Ronan Keryell (rkeryell@xilinx.com)

Xilinx Research Labs, San José, California

2020/01/15 @ Xilinx Software Developers Forum



Happy new year C++20!



ACM Turing award 2017: John L. Hennessy & David A. Patterson

A New Golden Age for Computer Architecture

John L. Hennessy, David A. Patterson, Communications of the ACM, February 2019, Vol. 62 No. 2, Pages 48-60

[https://cacm.acm.org/magazines/2019/2/
234352-a-new-golden-age-for-computer-architecture/fulltext](https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext)
Award lecture: <https://www.youtube.com/watch?v=3LVeEjsn8Ts>

- ▶ “*Those who cannot remember the past are condemned to repeat it*”,
George Santayana, 1905
- ▶ “*What we have before us are some breathtaking opportunities disguised as insoluble problems*”, John Gardner, 1965
- ▶ ↗ Agile Hardware & Software Development
 - Open Architectures, inspired by the success of open source software

*“The next decade will see a Cambrian explosion of novel computer architectures,
meaning exciting times for computer architects in academia and in industry.”*



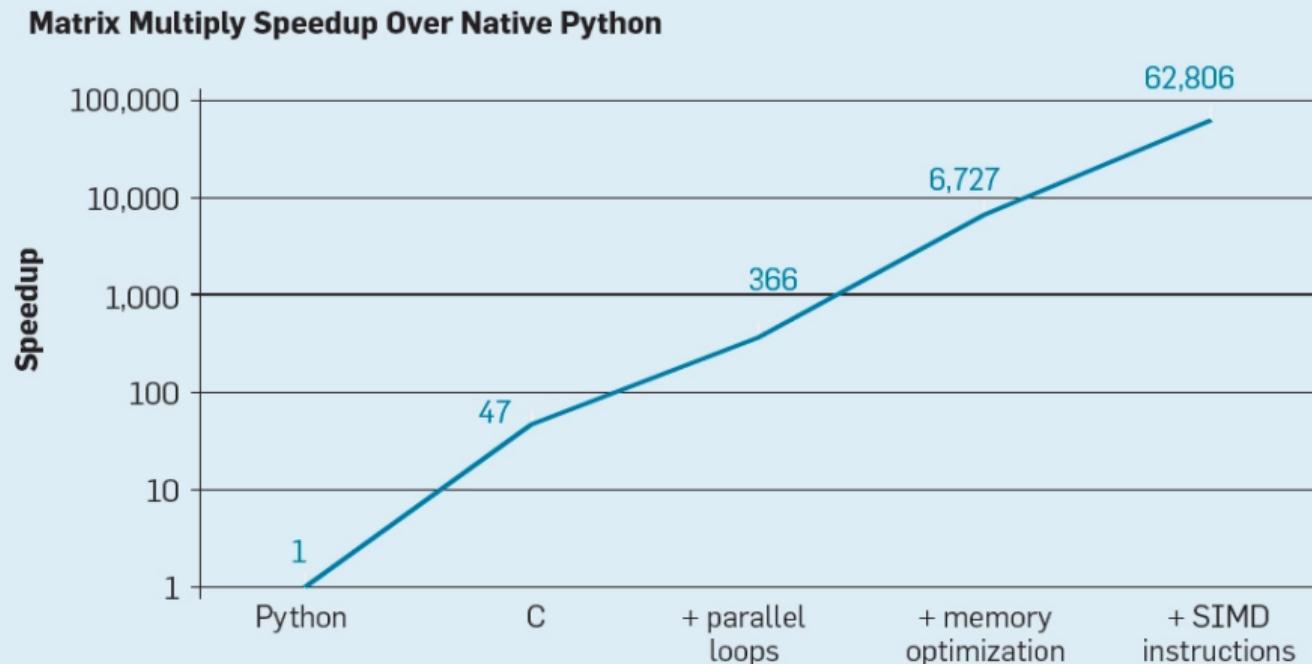
All Programmable [with hardware mind set]

Programming style

- ▶ Anything resolves to writing 0 and 1 in (configuration) memory
 - Great unification achieved !
 - Well done !
 - Problem solved !
- ▶ Actually done by Eckert, Mauchly and Von Neumann around 1944-1950... ☺



Huge programming dilemma: there's plenty of room at the top!



Charles E. Leiserson, Neil C. Thompson, Joel S. Emer, Bradley C. Kuszmaul, Butler W. Lampson, Daniel Sanchez, and Tao B. Schardl, "There's plenty of room at the top: What will drive growth in computer performance after Moore's Law ends?" unpublished manuscript submitted for publication, 2019.

The programmer's dilemma

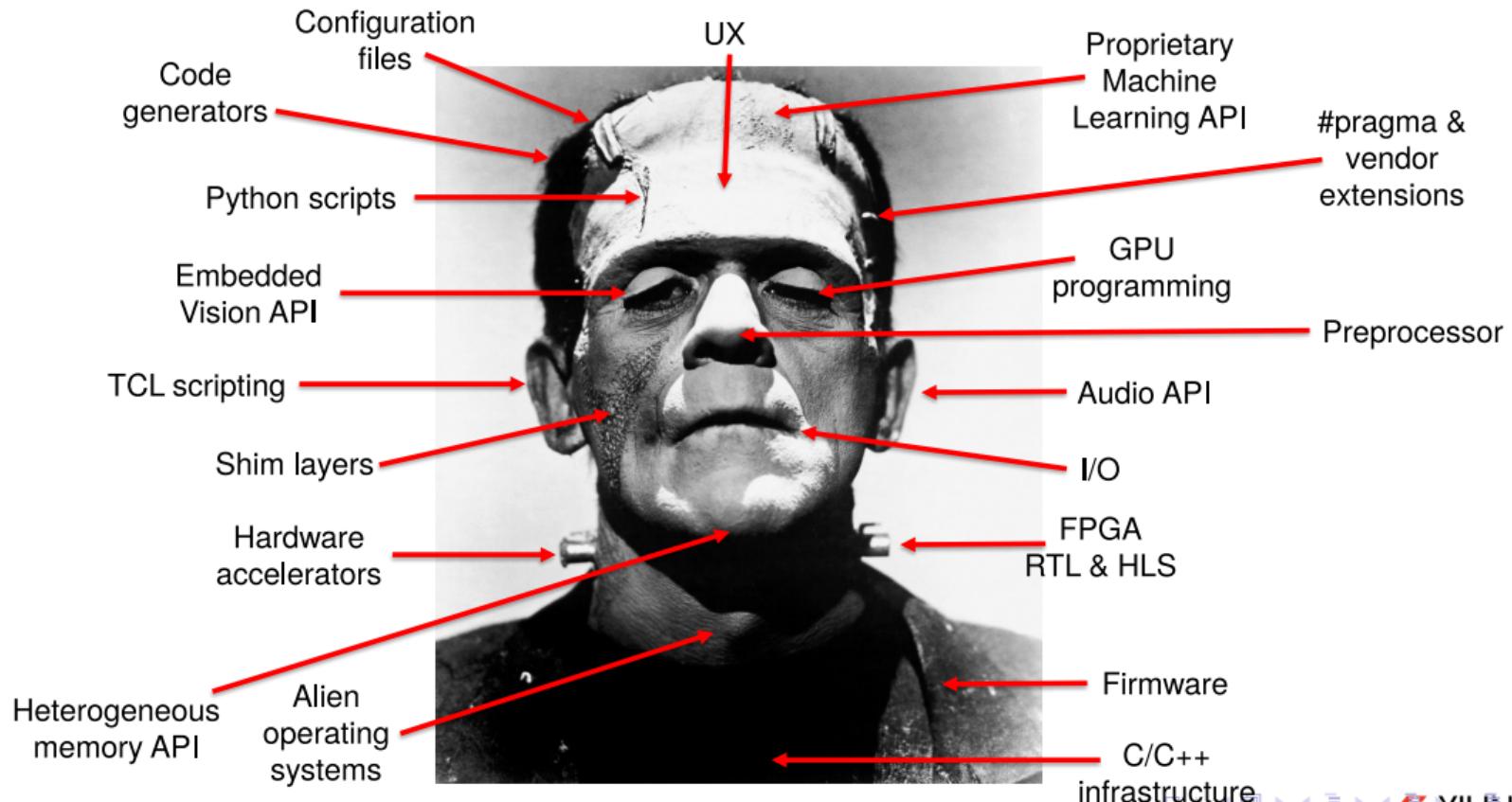
- ▶ Ubiquitous programming
 - Productivity & glue languages
 - Python
 - TCL
 - Perl
 - AWK
 - Shell
- ▶ Resource and performance optimization
 - Optimization/infrastructure/library/... languages
 - Assembly
 - VHDL
 - Verilog
 - C
 - C++

Real applications

- ▶ Any big enough application involves both productivity & optimization
- ▶ No one-size-fits-all language ☺
 - Involve different languages & different programming styles
- ▶ Lucky users can use scripting languages leveraging optimized libraries written with optimization languages
 - PYNQ
 - Machine Learning
 - ...
- ▶ Annoying to spend time switching languages
- ▶ Would be nice to have C++ experience less painful and with broader application domain...



Heterogeneous computing [software mind set nightmare]



Remember C++ ?

2-line description by Bjarne Stroustrup

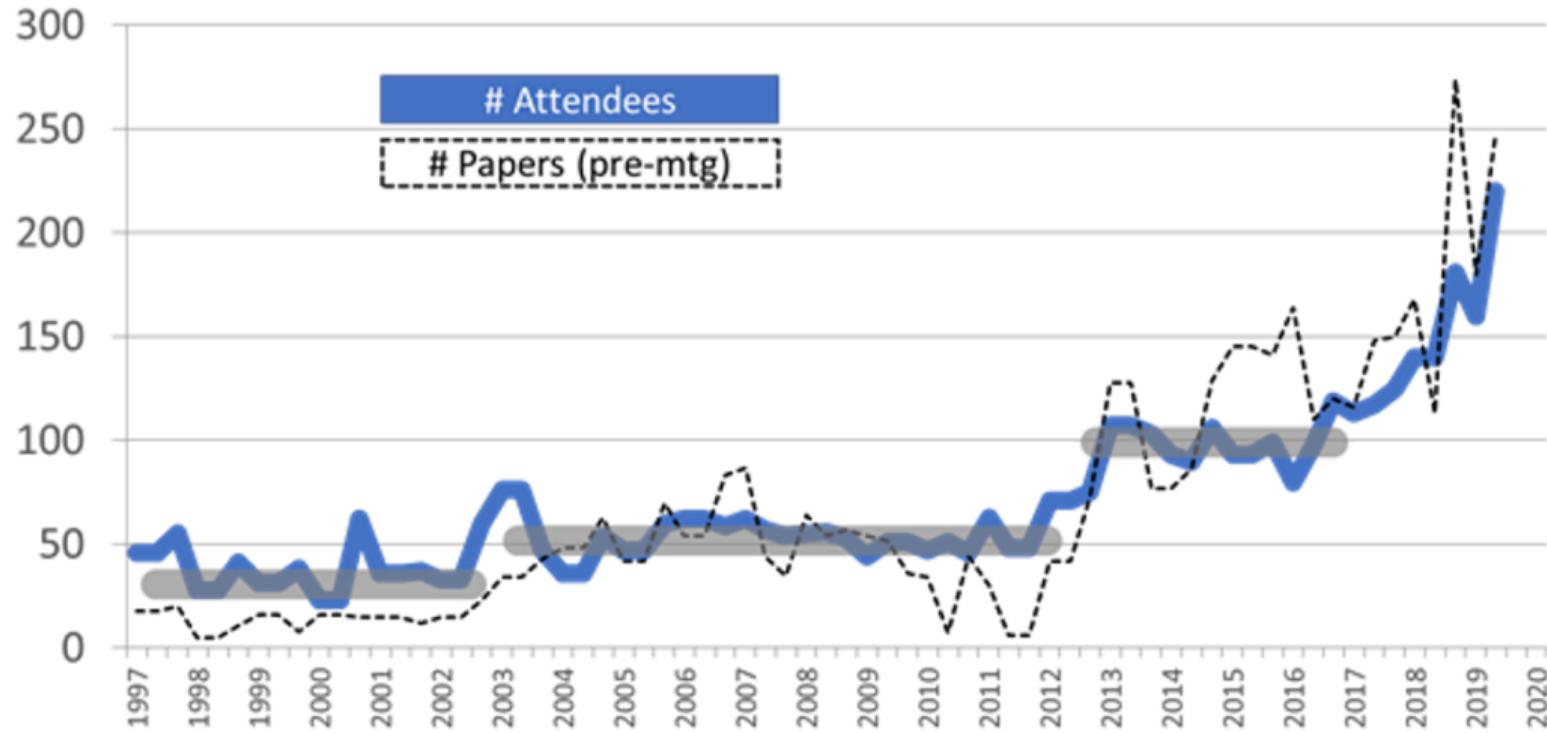
- ▶ Direct mapping to hardware
- ▶ Zero-overhead abstraction

But why C++?

- ▶ Very successful & ubiquitous language
 - Millions of programmers
 - Billions of users: any use of cars, IoT, IT, smartphone, Internet... depend on C/C++
- ▶ Interoperability: seamless interaction with embedded world, libraries, OS, C...
- ~~ Unique existing position in embedded system to control the full stack!!!
- ▶ **Full-stack**: combine both low-level aspects with high-level programming, multi-paradigm
 - Pay only for what you need
- ▶ Open-source production-grade compilers (GCC & Clang/LLVM) & tools
 - Implemented **before** specification
 - ... because it is too complex to do otherwise!!!
- ▶ Generic programming + classes can be used to define Domain Specific Embedded Language (DSEL)
 - Not directly targeting heterogeneous computing...
 - But extensible through classes (~ DSEL)
 - Xilinx AIE Cardano SDF framework
- ▶ Also extensible with **#pragma** and attributes (already in Xilinx Vitis for FPGA & AIE)

¿ And what about modern C++ ?

ISO C++ committee proposal papers & attendance



XILINX

1 C++ version/3 years ~ Parallelizing C++ committee itself!

- ▶ Language rebooted in 2011
 - ~~> Larger attraction & interest
- ▶ Is there any precedent of programming language developed cooperatively by of 200+ people democracy? ☺
- ~~> Use... parallelism ! ☺ <https://isocpp.org/std/the-committee>
 - Generic working groups
 - Core Working Group. Library Working Group, Evolution Working Group, Library Evolution Working Group
 - Specialized Study Groups
 - **SG1, Concurrency:** Olivier Giroux (Nvidia). Concurrency, parallelism, SIMD, memory model... SYCL is a candidate here!
 - **SG6, Numerics:** Lisa Lippincott (Tanium). Numerics topics, fixed point, decimal floating point, fractions, arbitrary precision...

- **SG7, Compile-time programming:** Hana Dusíková (Avast). Compile-time reflection & programming
- **SG14, Game Development & Low Latency:** Michael Wong (Codeplay). Embedded systems, heterogeneous computing
- **SG19, Machine Learning** Michael Wong (Codeplay). Better support for array, matrix, linear algebra...
- **SG20, Education:** JC van Winkel (Google). Produce guidance for modern course materials for C++ education

▶ Production of Technical Specifications before standardization

- Implemented in several compilers
- Allow experiments in the fields
- Provide feedback to the ISO C++ Committee

Make C++ more complex to make it... simpler!

- ▶ Modern C++ quite simpler to use
- ▶ But still compatible with old C++ & C
 - Globally more complex for... implementers!
- ▶ Keep **end-user only focused on simpler modern features**
- ~~ ISO C++ committee working on **C++ Core Guidelines**
 - <http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines>
- ▶ Subliminal messages
 - Learn modern C++... and teach students or customers! ☺
 - Forget about what you know from old C/C++ ☺



Modern Python/Modern C++/Old C++

► Python 3.8

```
v = [ 1, 2, 3, 5, 7 ]  
for e in v:  
    print(e)
```

► C++17

```
std::vector v { 1, 2, 3, 5, 7 };  
for (auto e : v)  
    std::cout << e << std::endl;
```

► C++03

```
std::vector<int> v;  
v.push_back(1);  
v.push_back(2);  
v.push_back(3);  
v.push_back(5);  
v.push_back(7);  
for (std::vector<int>::iterator i =  
     v.begin(); i != v.end(); ++i)  
    std::cout << *i << std::endl;
```



C++ auto because my laziness deserves it!

- ▶ Python

```
x = 123  
s = "hello"
```

- ▶ C++ auto (and some user-defined literals)

```
auto x = 123;  
auto s = "hello"; // A C string  
using namespace std::literals; // User-defined literals super-power  
auto constexpr a_std_string = "hello"s; // C++20  
auto & an_alias_to_x = x;  
auto size = 0b1001; // C++14  
auto big_num = 18'446'744'073'709'550'592lu; // C++14  
auto d = 3h + 10min + 5ns;  
auto const newYearsEve = 31d/December/2019; // C++20
```

- ▶ Also useful for Voldemort types (cannot be directly named...)

- Returning some locally defined types `auto f () { struct S { int i; }; return S { 42 }; }`
- Lambdas have anonymous types

Back to Python...

~ Modern C++ : like Python but with speed and type safety

- ▶ Python 3.x (interpreted):

```
def add(x, y): return x + y
print(add(2, 3))      # 5
print(add("2", "3")) # 23
print(add(2, "Boom")) # Fails at run-time :-(
```

- ▶ Same in C++14 but compiled + static compile-time type-checking:

```
auto add = [] (auto x, auto y) { return x + y; };
std::cout << add(2, 3) << std::endl;          // 5
std::cout << add("2"s, "3"s) << std::endl; // 23
std::cout << add(2, "Boom"s) << std::endl; // iii Does not compile !!! :-(
```

Without using templated code! ~~template <typename>~~ ☺



Back to Python...

~ Modern C++ : like Python but with speed and type safety

- ▶ Python 3.x (interpreted):

```
def add(x, y): return x + y
print(add(2, 3))      # 5
print(add("2", "3")) # 23
print(add(2, "Boom")) # Fails at run-time :-(
```

- ▶ Same in C++14 but **compiled + static compile-time type-checking**:

```
auto add = [] (auto x, auto y) { return x + y; };
std::cout << add(2, 3) << std::endl;          // 5
std::cout << add("2"s, "3"s) << std::endl; // 23
std::cout << add(2, "Boom"s) << std::endl; // !!! Does not compile !!! :-(
```

Without using templated code! ~~template <typename>~~ ☺



□ () {} C++ lambda emoji for functional programming

▶ Python feature

```
def f():
    c = 10
    # Capture of c inside the lambda
    return lambda x : x + c

l = f()
print(l(1), f()(20)) # 11 30
```

▶ C++14 generic lambda (function object)

```
#include <iostream>
auto f() {
    auto c = 10;
    // = to capture c by copy
    return [=] (auto x) { return x + c; };
}
int main() {
    auto l = f();
    std::cout << l(1) << ' ' << f()(20) << std::endl; // 11 30
}
```

- <https://godbolt.org/z/MxZpPk>
 - Look at CppInsights in link above to understand the syntactic sugar
- More features with C++17 & C++20...

Generic variadic lambdas & operator interpolation

```
#include <iostream>
#include <string>
using namespace std::string_literals;
// Define an adder on anything.
// Use new C++14 generic variadic lambda syntax
auto add = [] (auto... args) {
    // Use new C++17 operator folding syntax (here from left)
    return (... + args);
};

int main() {
    std::cout << "The result is: " << add(1, 2, 3) << std::endl;
    std::cout << "The result is: " << add("begin"s, "end"s) << std::endl;
}
```

Without using templated code! ~~template <typename>~~ ☺

Try this example on <https://wandbox.org/permlink/LambcvLqmyaSV4JL>



Algorithms

- ▶ High-level generic algorithms (sort, map-reduce...)
- ▶ 0-cost abstraction (templated header library)
 - Efficient implementations
 - Avoid unclear raw loops and spaghetti coding
- ▶ C++20 provides vector (SIMD) & parallel execution policies
- ▶ C++20 adds bit manipulation (`std::rotl`, `std::popcount`, `std::endian...`)

<https://en.cppreference.com/w/cpp/algorithm>

<https://en.cppreference.com/w/cpp/numeric>



Real “typical” problem in XSJ B4F2 (Mont-)Blanc Area

FOR FAVOR NO BOREE
PLEASE DO NOT ERASE.

Find values for the letters to get the largest value for the sum (OUCH)
Each letter must have a unique value.

$$\begin{array}{r} \text{S T U B} \\ + \text{T O E} \\ \hline \text{O U C H} \end{array}$$

B=6
F=3
C=4
O=9
S=8
T=7
U=4

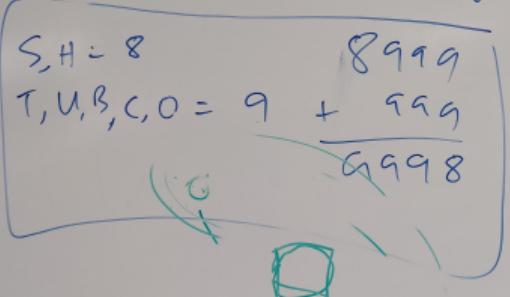
scintillating

$S, H = 8$
 $T, U, B, C, O = 9$

$$\begin{array}{r} 8999 \\ + 999 \\ \hline 9998 \end{array}$$

Correct

$$\begin{array}{r} 8756 \\ + 793 \\ \hline 9549 \end{array}$$





Solution in 20 lines & 15ms on my laptop

(I)

```
#include <algorithm>
#include <iostream>
#include <numeric>

// Convert digits into base-10 number
auto base_10_value = [] (auto... digits) {
    std::array a { digits... };
    return std::accumulate(a.cbegin(), a.cend(), 0,
                          [] (auto sum, auto x) { return sum*10 + x; });
};

int main() {
    std::array<int, 10> digits;
    std::iota(digits.begin(), digits.end(), 0);
    auto & [ B, C, E, H, O, S, T, U, _8, _9 ] = digits;
    decltype(digits)::value_type max = 0;
    do {
        auto a = base_10_value(S, T, U, B);
        auto b = base_10_value(T, O, E);
        auto r = base_10_value(O, U, C, H);
        if (a + b == r && r > max)
            max = r;
    } while (std::next_permutation(digits.begin(), digits.end()));
    std::cout << max << std::endl;
}
```

- ▶ <https://godbolt.org/z/g8-8TA>
- ▶ `make CXX=clang++-10 CXXFLAGS="-O3 -std=c++2a" quizz`



Solution in 20 lines & 15ms on my laptop

(II)

► Performance

```
perf stat ./quizz/quizz
```

```
9543
```

```
Performance counter stats for './quizz/quizz':
```

15.57 msec	task-clock	#	0.986 CPUs utilize
0	context-switches	#	0.000 K/sec
0	cpu-migrations	#	0.000 K/sec
145	page-faults	#	0.009 M/sec
64,917,402	cycles	#	4.170 GHz
218,597,685	instructions	#	3.37 insn per cycle
28,311,377	branches	#	1818.644 M/sec
200,191	branch-misses	#	0.71% of all branches

```
0.015781652 seconds time elapsed
```

```
0.015802000 seconds user
```

```
0.000000000 seconds sys
```

- $10! = 3628800$ iterations $\rightsquigarrow \frac{64,917,402}{10!} \approx 17.9$ cycles/iteration (including I/O)



Solution in 20 lines & 15ms on my laptop

(III)

► Clean solution

- No raw loop
 - Rely on standard algorithms from the STL
- Generic programming without `template` keyword
 - Use generic lambdas
- No explicit memory allocation/deallocation
- No pointer-based spaghetti code
- Spoiler alert: C++20 ranges provide shorter solution...



Most vexing parse in C++

- ▶ Old C++03 syntax based on constructor calls

```
std::string s("hello world!");
```

- ▶ I personally find this unclear when browsing code (for my heavy C background...)
 - It looks like a normal function call...
 - but this is a variable declaration!

- ▶ https://en.wikipedia.org/wiki/Most_vexing_parse

```
struct Timer { };
struct TimeKeeper {
    TimeKeeper(const Timer& t) {}
    int get_time() { return 0; }
};
int main() {
    TimeKeeper tk(Timer()); // what is tk?
    return tk.get_time();
}
```

<https://godbolt.org/z/SZnkE7>

Most vexing parse in C++

- ▶ Old C++03 syntax based on constructor calls

```
std::string s("hello world");
```

- ▶ I personally find this unclear when browsing code (for my heavy C background...)
 - It looks like a normal function call...
 - but this is a variable declaration!

- ▶ https://en.wikipedia.org/wiki/Most_vexing_parse

```
struct Timer { };
struct TimeKeeper {
    TimeKeeper(const Timer& t) {}
    int get_time() { return 0; }
};
int main() {
    TimeKeeper tk(Timer()); // what is tk?
    return tk.get_time();
}
```

<https://godbolt.org/z/SZnkE7>

- ▶ Function declaration for a function `tk` that returns an object of type `TimeKeeper` and has a single (unnamed) parameter that is a pointer to function returning an object of type `Timer` (and taking no input)

Uniform initialization and list initialization since C++11

- ▶ https://en.cppreference.com/w/cpp/language/list_initialization

```
struct Timer { };
struct TimeKeeper {
    TimeKeeper(const Timer& t) {}
    int get_time() { return 0; }
};
int main() {
    // tk is a TimeKeeper initialized from a default Timer
    TimeKeeper tk { Timer {} };
    return tk.get_time();
}
```

<https://godbolt.org/z/szyBPT>

- ▶ Prefer { and } when possible and keep (and) for explicit constructor calls

```
std::vector<int> v1 { 2, 3, 4 }; // Vector with elements 2, 3, 4
std::vector<int> v2 { 2 }; // Vector with element 2
std::vector<int> v3(2); // Vector with elements 0, 0
std::vector<int> v3(3, 42); // Vector with elements 42, 42, 42
```



Extended with designated initializer with C++20

https://en.cppreference.com/w/cpp/language/aggregate_initialization

```
// Before C++20, like in C
struct A { int x; int y; int z; };
A b { .x = 1, .z = 2 }; // b.y initialized to 0
// Since C++20, combining with default values
struct B {
    string a;
    int b = 42;
    int c = -1;
};
A { .c = 21 }; // Initializes a with {} (default constructor), then
                // initializes b with = 42, then initializes c with = 21
```

Named parameters (using designated initializer in C++)

► Cool Python feature

```
def print_point(x = 3, y = 42):
    print(x, y)
print_point()          # 3 42
print_point(1, 2)      # 1 2
print_point(x = 7)     # 7 42
print_point(y = 8)     # 3 8
print_point(x = 7, y = 8) # 7 8
```

► <https://brevzin.github.io/c++/2019/12/02/named-arguments> (Barry Revzin)

```
#include <iostream>
struct point {
    int x = 3;
    int y = 42;
};
void print_point(point p = {}) { std::cout << p.x << ' ' << p.y << std::endl; }
int main() {
    print_point();                      // 3 42
    print_point({ 1, 2 });              // 1 2
    print_point({ .x = 7 });            // 7 42
    print_point({ .y = 8 });            // 3 8
    print_point({ .x = 7, .y = 8 });    // 7 8
}
```

<https://godbolt.org/z/FfE8pn>

- In C++20, it works also for template parameters... ☺

Tuple (product type)

- ▶ “Product type” in algebraic types (computer science jargon)
- ▶ Python tuple: immutable heterogeneous sequence

```
def t():
    return (1, 5.6, "hello")

tup = t()
print(tup[0], tup[1], tup[2]) # 1 5.6 hello
```

- ▶ Useful to lazily gather data (passing arguments, returning several values at once)
- ▶ C++11 provides std::tuple <https://en.cppreference.com/w/cpp/utility/tuple> and std::pair <https://en.cppreference.com/w/cpp/utility/pair>

```
#include <iostream>
#include <tuple>
auto t() {
    return std::tuple { 1, 5.6, "hello" };
}
int main() {
    auto tup = t();
    std::cout << std::get<0>(tup) << ' '
        << std::get<1>(tup) << ' '
        << std::get<2>(tup) << std::endl;
}
```

<https://godbolt.org/z/A4jB5G>

- ▶ Both Python & C++ syntax would be cumbersome if there was no



Tuple (product type)

- ▶ “Product type” in algebraic types (computer science jargon)
- ▶ Python tuple: immutable heterogeneous sequence

```
def t():
    return (1, 5.6, "hello")

tup = t()
print(tup[0], tup[1], tup[2]) # 1 5.6 hello
```

- ▶ Useful to lazily gather data (passing arguments, returning several values at once)
- ▶ C++11 provides std::tuple <https://en.cppreference.com/w/cpp/utility/tuple> and std::pair <https://en.cppreference.com/w/cpp/utility/pair>

```
#include <iostream>
#include <tuple>
auto t() {
    return std::tuple { 1, 5.6, "hello" };
}
int main() {
    auto tup = t();
    std::cout << std::get<0>(tup) << ' '
        << std::get<1>(tup) << ' '
        << std::get<2>(tup) << std::endl;
}
```

<https://godbolt.org/z/A4jB5G>

- ▶ Both Python & C++ syntax would be cumbersome if there was no...

Structured binding

► Python tuple unpacking

```
def t():
    return (1, 5.6, "hello")

i, f, s = t()
print(i, f, s) # 1 5.6 hello
```

► C++

```
#include <iostream>
#include <tuple>
auto t() {
    return std::tuple { 1, 5.6, "hello" };
}
int main() {
    auto [i, f, s] = t(); // C++17, works also with structures and arrays
    std::cout << i << ',' << f << ',' << s << std::endl; // 1 5.6 hello
    // Old C++11 way
    int ii; float ff; const char *ss; // Need to declare first
    std::tie(ii, ff, ss) = t();
    std::cout << ii << ',' << ff << ',' << ss << std::endl; // 1 5.6 hello
}
```

https://godbolt.org/z/wGYuU_

Sum type (algebraic type) std::variant

- ▶ Python is very dynamic and agile (and slow): native sum type!

```
v = 1  
v = 5.6  
v = "hello"
```

Use dynamic typing, dynamic allocation and garbage collection...

- ▶ C++: *only pay for what you need*

<https://en.cppreference.com/w/cpp/utility/variant>

```
#include <iostream>  
#include <variant>  
int main() {  
    std::variant<int, double, char> v; // C++17: declare allowed types ahead of time  
    v = 1;  
    std::cout << std::get<int>(v) << std::endl; // 1  
    v = 5.6;  
    std::cout << std::get<double>(v) << std::endl; // 5.6  
    std::cout << std::get<char>(v) << std::endl; // Throw 'std::bad_variant_access'  
}
```

<https://godbolt.org/z/WXiGjE>

- Replace typically C structure + tag in a nicer/safer way
- Does not use dynamic allocation

Wilder C++ sum type with std::any

- ▶ Closer to Python variable
- ▶ C++: *only pay for what you need*
 - Can hold any type, not only among predefined ones
 - More expensive: add dynamic allocation & RTTI for not declaring allowed type ahead
 - https://en.wikipedia.org/wiki/Run-time_type_information

<https://en.cppreference.com/w/cpp/utility/any>

```
#include <any>
#include <iostream>
int main() {
    std::any v; // C++17
    v = 1;
    std::cout << std::any_cast<int>(v) << std::endl; // 1
    v = 5.6;
    std::cout << std::any_cast<double>(v) << std::endl; // 5.6
    std::cout << std::any_cast<char>(v) << std::endl; // Throw 'std::bad_variant_access'
}
```

<https://godbolt.org/z/e6Z53v>

- Still no need for garbage collector ☺

Dealing with optionality

- ▶ Python provides None

```
def compute():
    if no_error:
        return 42
    else:
        return None
v = compute()
if v is not None:
    print(v) # Process v...
```

- ▶ C++17 introduced std::optional

<https://en.cppreference.com/w/cpp/utility/optional>

```
#include <iostream>
#include <optional>

std::optional<int> compute() {
    if (no_error)
        return 42;
    return {};
```

```
} int main() {
    auto v = compute();
    // v is a std::optional<int>, not an int...
    if (v)
        std::cout << *v << std::endl;
}
```

- ▶ Recurring examples in Xilinx code base

```
T * p == nullptr;
p = new T; // or malloc()
if (p)
    // OK, we have a thing, work on it...
if (p)
    delete p; // or free()
```

- Spaghetti code, unsafe, memory leaks, no exception safety... ☹



Dealing with optionality

- ▶ Python provides None

```
def compute():
    if no_error:
        return 42
    else:
        return None
v = compute()
if v is not None:
    print(v) # Process v...
```

- ▶ C++17 introduced std::optional

<https://en.cppreference.com/w/cpp/utility/optional>

```
#include <iostream>
#include <optional>

std::optional<int> compute() {
    if (no_error)
        return 42;
    return {};
```

```
} int main() {
    auto v = compute();
    // v is a std::optional<int>, not an int...
    if (v)
        std::cout << *v << std::endl;
}
```

- ▶ Recurring examples in Xilinx code base

```
T * p == nullptr;
p = new T; // or malloc()
if (p)
    // OK, we have a thing, work on it...
if (p)
    delete p; // or free()
```

- ▶ Spaghetti code, unsafe, memory leaks, no exception safety... ☺



Text formatting

- ▶ Python <https://docs.python.org/3/library/string.html?highlight=format>

```
print ("Hello {} {}\n".format("world", 42)) # Hello world 42!
```

- ▶ C++20

```
#include <iostream>
#include <format>
int main() {
    std::cout << std::format("Hello {} {}\n", "world", 42);
} // Hello world 42!
```

- <https://en.cppreference.com/w/cpp/utility/format/format>
- Reuse the format specification in Python!!! ☺



Move semantics

- ▶ Python use references and garbage collector

```
SIZE = 1 << 26
def vector_inc(x):
    result = [0] * len(x)
    for i in range(len(x)):
        result[i] = x[i] + 1
    return result
x = [0] * SIZE
v = vector_inc(x)
print (v[0])
```

- ▶ C++03 (& C style)

```
#include <iostream>
#include <vector>
std::size_t constexpr size = 1 << 29;
void vector_inc(std::vector<int> &result,
                const std::vector<int> &x) {
    result.resize(x.size());
    for (std::size_t i = 0; i < x.size(); ++i)
        result[i] = x[i] + 1;
}
```

```
int main() {
    std::vector<int> x(size);
    std::vector<int> v;
    vector_inc(v, x);
    std::cout << v[0] << std::endl;
}
```

- ▶ C++17 with move semantics & copy elision

```
#include <iostream>
#include <vector>
std::size_t constexpr size = 1 << 29;
std::vector<int> vector_inc(const std::vector<int> &x) {
    std::vector<int> result(x.size());
    for (std::size_t i = 0; i < x.size(); ++i)
        result[i] = x[i] + 1;
    // Use Named Return Value Optimization or "move" constructor
    return result;
}
int main() {
    std::vector<int> x(size);
    // v (re)use dynamic storage of result: no copy
    std::vector<int> v = vector_inc(x);
    std::cout << v[0] << std::endl;
}
```



Ranges in C++

- ▶ Python <https://docs.python.org/3/howto/functional.html>
- ▶ One of the most exciting features of C++20
 - Yet another completely new programming style!
 - Functional programming and composability

Xilinx HR tooling problem

- ▶ Xilinx hiring managers have to use Taleo to handle candidates
- ▶ Does not work correctly when more than 50 candidates
 - Not possible to extract résumés
- ▶ But HR people can use Talemetry to extract candidate list with 1 URL for each résumé
 - ↗ A lot of mouse clicks per candidate... ☹
- ▶ Need a tool to fetch all the résumés on my computer
- ▶ Obvious use case for Python...

CSV HR document

First Name	Last Name	Email	Home Phone	City	State	Country	Profile Url
Xzwla,Nhrnmfknb,fahgpkywe.yiblv@pxreq.lcq,+9	(738) 390 0177,Vbv	Xbmcmpd,KZ,EY,hcfds://bkvje.lakfqphiu.bbb/fgimfbz/628c8169-433y-00vk-17c8-3s0ce15624rt					
Osufn,Pkagalvm,mmngxqkf48@tyulr.dvv,+3	(564) 991 6254,Yhgoems,MC,VU,mlept://gndqd.fxrqmzzd.ls.uwznukkf/16277617-922r-02cw-289s-4g6092be1b2n						
Uwroxb,Fapveghhu,eazlrczu4944@anqwy.equ,+2	(177) 489 7972,Kgu	Zkzf,IV,HG,pzttf://mwzin.qhnuzrwpj.iil/fojrkdw/lxat3pp-1158-56ly-jb75-413vz4400188					
Gqlqyip,Emuldzeo,mqrzukbd@iivfbbl.vwq,+5	(862) 849 1167,Lqcoeczd,PI,QU,laiku://gcjlp.ugxtssxgl.mey/hzgrpvqg/c5wjmk60-95e3-48h1-406w-5n7618o7f484						
Vcand,Vyamjdg,lzjeue96@tonnn.btr,+7	(897) 013 8364,Rfpnbl,TD,XQ,lykqmq://hrzvn.dgayhsmmc.tjd/ciwubvpm/se54s50w-2t26-47jk-y51z-1wxg62jb7110						
PNKOWI,BDM,engrzbtvt14016@joihx.dba,+9	(120) 169 0312,Tuxzz Nnzw,QS,XA,uinwg://koksv.bopxhwalv.mlv/swmomopc/w49049w6-3f89-15ae-eu8r-3k54d7273647						
Fhhtcejnh,Ykqx,orgucfayk.pijzaaaqld@rxtxes.ljk,+6	(048) 761 1046,Algjn,LG,GJ,xpbvs://mzjms.vlakcloc.fjf/uaewngks/jw23j44w-3h87-11zg-svd3-3mu63kn7y86						
Mcuqfk,Usduism,elaeit_hotqozx@jsnwt.ant,+6	(285) 086 6741,Lgacxgzg,PI,LT,glsjl://futty.uigdldwmu.rez/ukerddjm/z304kjx0-9jnt-27gk-cq76-3q928d0e33m6						
Zwik,Ulcx,jzgfyuwf@guivx.xme,,WI,yjoju://zdeqc.zprenfujt.fml/vwpvgzyf/32f7625v-40p9-45d7-tzu1-3y53tm5f1ytd							
Cheocs,Aezeicwpok,gryeed4410@cifda.did,+9	(247) 883 7498,Cahtugje,WV,YR,dtptq://jzmfh.dxntipegv.chc/obewpmft/36370907-8ddk-47tv-5672-5xhy4f18614e						
Jvhseymx,Nrotxl,fngsqzghsbkgla@aujfj.vmo,+5	(255) 653 5285,Vhb	Qdju,VR,EZ,umavx://hecuk.pqmbfozq.qtz/icusncoj/585r2ba0-9d0x-101c-6yfl-5v5682d2a348					
Pnwiz,Go,hujxcjn4@orbqqjsw.sva,+8	(763) 741 8494,Efvdkmfd,ON,JU,isvvi://tmrxw.wnpafjzlu.qlha/ffwqeqmx/833jadv6-2589-57sc-j4kx-6hbv87bb3t06						
Tgbvoj Qrsgj,Jkomot,yqe963972@scccviyqr.jxi,+5	(625) 095 3396,Xzwxs,AJ,JZ,obojo://11ijm.etotjmolm.uek/ftvabgi/708h6613-8816-35mg-go04-100171k2j862						
Gipo,Csxpl Sfwmzz,nndlalvattfowys@ykbjobjb.wyf,+7	(936) 732 1202,Ckwuyyc,QR,EF,lrzzq://myixc.grtwfqdyj.zpa/oquydkm/66ro52ko-185i-75zh-8xhy-3yk3p137r9j						
Heuglvtjmez,GkibtL,uzcrgpykuqpo524220@mujf.mma,+4	(235) 772 7938,Scconv Xpeqfp,BD,HL,jvjin://rrhoi.ggvnpbmrni.ted/ocggdivc/nv079928-2001-69wu-s7z7-9u882b6a07q9						
Qwgxa,Afxxgqdi,cd_cywfqgis@cmg.sy,+22	4478813801455,Qéfurt ou Uaqjku,,IX,wstlu://hhhti.gnztaskmc.zwt/rwdzbqbs/64yn8mq2-2a2z-85as-94d0-6y484g0x25z6						
Fwegf,Igvafbv,ismvpzhvgpsnq@qiuyo.kpn,+4	(800) 047 8626,,OL,HO,vcmvg://bdvuc.abmtuigp.mlq/efyfvjkj/37319eir-5095-76no-14q8-1ck2277366v4						
Vefgzs,Edsaf,hbdolz@jysnpfrs.gtd,+9	(034) 565 9704,Pjcpajm,IR,IP,yqyrx://zrejx.yfjivktez.kpn/bswuumcl/x3779541-20ef-73jb-jlkh-3s69m6661795						
Jhnyhc,Rocy,pyui.jqxzlnw@zgzm.imy,+1	(414) 108 3668,Ufm	Rnbklqj,MT,NN,cwaiq://lcbxr.hqnqckrqg.iux/zdaqiqw/s8cm79dr-540e-81ya-dn5x-3w222ab792f					
Rveaex,Vydx_Exudzqp,tcpkyo.yhv179@rozkhdy.qoy,+0	(922) 670 0661,Nnjzfar,IN,HX,ykzgu://jruhv.iutharpg.dag/mwipsilt/312bz895-1768-99zo-468e-9o5161gq830z						
Jk,Zggsw,up.hng@zstof.riq.rxy,+7	(312) 409 8134,Vwgsij,QM,AV,dyhzz://gkbyx.ivxndjdp.nri/xrzkwqg/22yu66y8-5818-32zh-b15m-3mr060ck8w3q						
Ednducw,Oil9,rfrccenbuf@waxzq.lyj,+9	(256) 234 7387,Rhomn Oslxciw,PO,HB,cdprp://ntvpv.nrbdcxwsh.yxc/ubuyhask/1h1322fx-83i5-94id-k803-0o2018257dv6						
Kswbh,Ckzlukh,usccgdfhqeze@crkkw.yef,+1	(522) 061 8090,Haqyq,BE,RO,ejvkt://dtwpy.antltvtfr.hzj/bxo1lxue/25m951j9-1342-86oo-c4p9-5mx430qn0a4d						
Tcvvwml,Iredgh,xkekred0dviediq.wi.qrn,+4	(910) 399 4282,,VA,GF,vpdqa://zejss.wkagbbscw.lbj/ubfmp1wl/wabi3co1-83d9-29h7-u1px-5m3447772892						
Oébhumiwzc,Feydti,dknjnjrit.aoigpa@nsg.owq.fn,+28	118049872,Ityga-Zfq-Fjsol,,ZD,wauew://usrtu.cbndwpcmk.cnq/naaoiinh/yufuxn71-6st4-31jx-h1q1-4d8v5949t6qy						
Fcvncv,Jtjuoyj,jg98788@vitlb.wzh,+7	(045) 364 9694,Brunuq,PW,ML,frnut://eamvy.hbphsmgyb.isf/dvnusehd/6t28f7dx-5r47-25nn-0008-7p94v771w166						
Jtjznq,Jqpqh,wqkof.nw10ktasf.mew.iub,+1	(409) 654 7552,Prnnwu,OC,JJ,gwmuc://spmuw.mhynnbbpg.cfy/pehepvd़/37u2v86h-6i1h-51xr-b13e-6138o928n200						
Ulhu,Sqqpzy,lhpuv.dfevmxj@rhfpaz.uzk,+5	(788) 322 9824,Yjxboytg,FL,DK,mltcy://vzffo.hausaulhl.ipb/qinjrenm/0j4723xm-4r60-04et-gdyi-9x49c3u36904						
Cmque,Eammtwl,kfuuv.dfevmxj@rhfpaz.uzk,+5	(788) 322 9824,Yjxboytg,FL,DK,mltcy://vzffo.hausaulhl.ipb/qinjrenm/0j4723xm-4r60-04et-gdyi-9x49c3u36904						
Ailguwmg,Vhcanjhk,xaykalx.goyqfbxt@qqeolhy.em,+86	675936652,Kpwu,,JT,mbcyl://kjytb.smcpuuvsj.mvs/twhstxb/17075g6e-1g6b-63to-8384-8ith37wn2221						



Fetch candidates with C++20

```
#include <filesystem>
#include <fstream>
#include <iostream>
#include <regex>
// Waiting for C++20 library
// #include <span>
#include <string>
#include <boost/process.hpp>
#include <range/v3/all.hpp>
// TODO add to range-v3/include/range/v3/view.hpp
#include <range/v3/view/remove.hpp>
namespace bp = boost::process;
using namespace ranges;

int main(int argc, char *argv[]) {
    // Go into a safer C++20 world
    // std::span args { argv + 1, argc - 1};
    // Skip the first argument which is actually the program name
    for (auto i = 1; i < argc; ++i) {
        // Open each file from the names passed as argument
        auto file = std::ifstream { argv[i] };
        // Create a local subdirectory and work into it
        auto constexpr subdirectory = "CV";
        std::filesystem::create_directory(subdirectory);
        std::filesystem::current_path(subdirectory);

        // Iterate on all the lines but the first header one
        for (const auto & line : getlines_view { file } | views::tail) {
            std::cout << "Handling " << line << std::endl;
        }
    }
}
```

```
auto columns = line
// Remove There are still some raw '\r' because getlines_view cannot
// handle yet MS-DOS text files in a unix environment
| views::remove('\r')
// Assume CSV fields separated by ','.
| views::split(',')
// Put the result to a vector so we can easily access the last element
| to<std::vector>;
// Convert the last range of characters into a string
auto url = to<std::string>(*columns.crbegin());
// Rewrite the candidate URL to the candidate résumé URL
auto resume_url = url + "/resume/download";
resume_url = std::regex_replace(resume_url,
                               std::regex { "/profiles/" },
                               "/candidate/");
// Use curl to do the real fetching, to handle HTTP protocol
bp::system(bp::search_path("curl"),
           // Use the name advertised by the server to store the file
           "--remote-name",
           // Telemetry impose some browsers... :-((
           // Just lie by imitating some other one
           "--user-agent",
           "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:70.0) Gecko/20100101 Firefox/70.0");
resume_url);
```

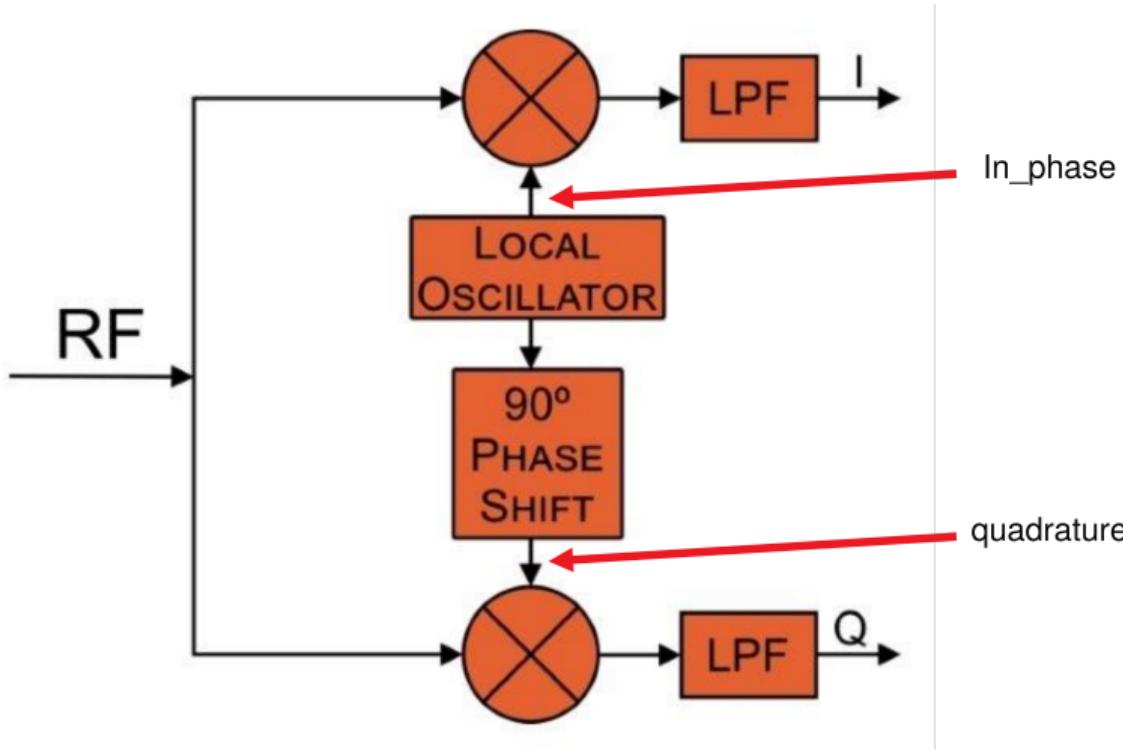
https://github.com/keryell/fetch_resumes

Example: radio receiver



<https://xkcd.com/1363>

Simple example: generate sequence in quadrature



<https://www.allaboutcircuits.com/textbook/radio-frequency-analysis-design/radio-frequency-demodulation/understanding-quadrature-demodulation>

Generate sequence in quadrature: the Python 3.8 way

```
#! /usr/bin/env python3
import itertools
import sys

samples = 8
if samples % 4 != 0 :
    sys.exit("sample numbers need to be a multiple of 4")

table = [ t/(samples - 1) for t in range(samples) ]

def tabulate(x):
    return table[x]

r, r2 = itertools.tee(itertools.cycle(range(samples)), 2)
in_phase = map(tabulate, r)
quadrature = map(tabulate, itertools.islice(r2, int(samples/4), None))
output = itertools.zip_longest(in_phase, quadrature)
for i, q in output:
    print ("{:f} {:f}".format(i, q))
```

Generate sequence in quadrature: compare the C++20 way

```
#include <iostream>
#include <range/v3/all.hpp>
using namespace ranges;
auto constexpr samples = 8;
static_assert(samples % 4 == 0, "sample numbers need to be a multiple of 4");
auto constexpr generate_table() {
    std::array<float, samples> a;
    for (int i = 0; i < samples; ++i)
        a[i] = i/(samples - 1.f);
    return a;
}
// C++20 constinit: execute generate_table() at compile time!!!
auto constinit table = generate_table();
auto tabulate = [] (auto v) { return table[v]; };

int main() {
    auto r = views::ints | views::take(samples) | views::cycle;
    auto in_phase = r | views::transform(tabulate);
    auto quadrature = r | views::drop(samples/4) | views::transform(tabulate);
    auto output = views::zip(in_phase, quadrature);
    for (auto [i, q] : output) {
        std::cout << i << ' ' << q << std::endl;
    }
}
```

<https://godbolt.org/z/GSgtXt>

C++20 features useful for hardware people (& software people...)

- ▶ Features for higher-level (pointer- & loop-less) programming: containers, algorithms, ranges...
 - Explicit memory accesses & management are painful to optimize/recognize/debug (HLS, polyhedral models...)
- ▶ Performance: parallelism, executors, co-routines, SIMD...
- ▶ Functional programming
 - Higher-order functions that accept functions
 - Program (and *hardware!*) composition: fpga_pipeline_executor(convol, relu, max_pool2);
- ▶ Type-safety at compile-time
 - Do not detect problems at run time (Python...)
 - Program optimized & safe for the real safety-critical use case (ADAS AUTOSAR & MISRA
 ↗ C++17, fly-by-wires...)
- ▶ Generic programming
 - Type-independent code like in Python

```
auto miraculous_adder = [] (auto ... args) { return (... + args);};
```
- ▶ Metaprogramming
 - Transform/generate/introspect code at compile time
 - Adaptable computing ↗ optimized hardware execution

```
boost::hana::int_c<10>.times.with_index([&] (auto i) { std::cout << f(i) << std::endl;});
```



C++20 down to assembly code specialization

```
#include <cassert>
#include <type_traits>
constexpr auto sum = [](auto arg, auto... args) {
    if (std::is_constant_evaluated()) {
        return (arg + ... + args);
    } else {
        [&](auto value) {
            asm("leal (%1, %2), %0"
                : "=r" (arg)
                : "r" (arg),
                  "r" (value));
        }(args), ...);
        return arg;
    }
};
static_assert(6 == sum(1, 2, 3));
int main(int argc, char *[])
{
    assert(10 == sum(argc, 2, 3, 4));
}
```

- ▶ Remember inline assembly???
- ▶ Now with compile-time specialization without code change!!!
- ▶ <https://godbolt.org/z/CJsKZT>
- ▶ C is just *le passé...* ☺
- ▶ Inspired by Kris Jusiak @krisjusiak Oct 24 2019
 - <https://twitter.com/krisjusiak/status/1187387616980635649>
 - C++20: Folding over inline-assembly in `constexpr`



C++ resources

- ▶ Start with <https://isocpp.org>
- ▶ <https://cppreference.com> for good hand-on reference on specific features
- ▶ Socialize in C++ meetups, such as <https://www.meetup.com/ACCU-Bay-Area>
- ▶ More and more C++ conferences, some have video on-line
 - CppCon <https://www.youtube.com/user/CppCon/videos> (specially for Longmont people!)
- ▶ My preferred book in 2019: *Functional Programming in C++*, Ivan Čukić
<https://www.manning.com/books/functional-programming-in-c-plus-plus>
- ▶ C++ Core Guidelines
<http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines>

Conclusion

- ▶ Heterogeneous computing is everywhere and here to stay
 - More demand for finer resource control without throwing programming comfort!
- ▶ Real applications require collaboration of several languages
 - Python and other modern productivity languages are cool!
 - *Modern versions* of old languages can be cool too! ☺
- ▶ C++20 is not a collection of individual features but a synergy of features
 - I presented just a few points of C++20... There are quite more!
 - Still compatible with old C & C++
 - Mature ecosystem + huge modernizing effort without killing the legacy
 - Modern C++ can address the full stack from assembly to higher-level multi-paradigm
 - C++20 ranges are close to AXI4 streams and Cardano SDF window buffers
- ▶ For beginners: many Python constructions can be directly translated into C++20
 - Can give performance improvement out-of-the box for critical parts
- ▶ You are not happy with C++?
 - Do it the standard way! Participate to the standards to have a real impact! ☺
 - ISO C++ is an elitist democracy: only what is implemented and working is ratified
 - SYCL standard as part of Khronos effort to improve ISO C++
- ▶ Stop programming in JurassiC++! Make Xilinx attractive for new software hire!

ACM Turing award 2017: John L. Hennessy & David A. Patterson
 All Programmable [with hardware mind set]
 Huge programming dilemma: there's plenty of room at the top!
 The programmer's dilemma
 Real applications
 Heterogeneous computing [software ~~mind set~~ nightmare]
 Remember C++ ?
 But why C++?
 ISO C++ committee proposal papers & attendance

1 C++ version/3 years ↗ Parallelizing C++ committee itself!
 Make C++ more complex to make it... simpler!
 Modern Python/Modern C++/Old C++
 C++ auto because my laziness deserves it!
 Back to Python...

↗ Modern C++ : like Python but with speed and type safety
 Back to Python...

↗ Modern C++ : like Python but with speed and type safety
 [] () {} C++ lambda emoji for functional programming
 Generic variadic lambdas & operator interpolation
 Algorithms
 Real "typical" problem in XSJ B4F2 (Mont-)Blanc Area
 Solution in 20 lines & 15ms on my laptop
 Most vexing parse in C++

3	Most vexing parse in C++	27
4	Uniform initialization and list initialization since C++11	28
5	Extended with designated initializer with C++20	29
6	Named parameters (using designated initializer in C++)	30
7	Tuple (product type)	31
8	Tuple (product type)	32
9	Structured binding	33
10	Sum type (algebraic type) std::variant	34
12	Wilder C++ sum type with std::any	35
13	Dealing with optionality	36
14	Dealing with optionality	37
15	Text formatting	38
16	Move semantics	39
	Ranges in C++	40
	Xilinx HR tooling problem	41
	CSV HR document	42
17	Fetch candidates with C++20	43
	Example: radio receiver	44
18	Simple example: generate sequence in quadrature	45
19	Generate sequence in quadrature: the Python 3.8 way	46
20	Generate sequence in quadrature: compare the C++20 way	47
21	C++20 features useful for hardware people (& software people...)	48
22	C++20 down to assembly code specialization	49
23	C++ resources	50
26	Conclusion	51
	You are here !	52