Interview with Bjarne Stroustrup
24th Feb 2020 – 19:00 GMT+1 Paris

Beginning – General questions

1° How does a programming language is born ? How did you create C++ ? Explain us the story !

> C++, like most of the languages I like best, started with a need, a problem. I didn't particularly want to design a programming language. I wanted to build a system, but I decided that I couldn't build that system – a versions of Unix to run as a distributed system – with the existing languages. I needed a language that could do low-level things like memory managers, device drivers, and process schedulers. I also needed a language that could do high-level things like specifying components of a program and how they communicated. No language could do both, so I built C++. No. I never got to build my distributed system, but others built such systems using C++.

2° Are you surprise by the success of C++ ?

> Yes, definitely. I was – and am – quite amazed. There never was a time where C++ didn't have several better financed and heavily marketed competitors. Many aspects of the design were not fashionable, just responses to real-word challenges. Yet the C++ community grew and still grows. C++ serves real needs.

3° C++ is ISO. It is crucial and important ? To maintain an ecosystem and have the guarantee it runs on various platforms.

> For C++, the ISO standards committee (WG21) is essential. We don't have a rich owner and to collaborate, competing organizations need an open forum like WG21 in which to collaborate. Where else can people from competing organizations sit down and solve problems together? Having a standards committee doesn't guarantee anything; only the quality of the standard give success. It is crucial that the standard is based on consensus, so that all compiler and standard-library providers find it worth-while to adhere to it. In Prague, the vote for C++20 was 79-0 in favor! That means that C++20 will become official in the fall. Most parts of C++20 are already shipping in various compilers and much is in production use. C++20 is not Science Fiction.

4° Making enhancement to a programming language is difficult. We saw it with Sun and Oracle with Java, the Kotlin case, making the risk to break compatibility. What is your feeling with C++ ?

> Stability is an important feature, especially for people who build systems than must last for decades. C++ has evolved into a much more expressive and performant language than what I had in the 1980s, but we have been very careful to maintain a great degree of compatibility. Everybody wants a smaller, simpler language. They also want more features. Furthermore, most C++ developers insists that their old code should not be broken. Delivering all three is impossible. I try to maintain compatibility while adding ways to simplify the use of C++.

5° C++ has evolved among years with a rapid growth during the last decade. It is a good thing ?

Yes. New challenges emerge and new ideas are needed to meet them. A living language changes to keep up with a changing world. That is true for both natural and artificial language. Also, I always knew that C++ wasn't yet the best match for my ideals. For example, in 1981 I wrote that generic programming was needed. I conjectured that macros could be used to meet that need, but that didn't scale, so I implemented templates. However, templates met only part of my ideals for generic programming. Only with 'concepts' are we getting close.

Another example is concurrency and parallelism where we had to standardize a strongly typed threads-and-locks level of support. As a systems programming language, C++ had to provide direct support to what the operating systems offered. Then, we have to offer better standard support for lock-free programming, and only later could we address the higher-level models of concurrency and parallel algorithems. C++20 provides parallel algorithms and C++23 is likely to provide a general model of concurrency.

Over the ysear, C++ has also provided many facilities to simplify programming. 'Make simple things simple !' is a common comment. In modern C++, we have to write far less code to manage resources, to express general algorithms, to write simple loops, etc. than in older versions of C++. We canno simplify the language without breaking code, but we can simplify the use of the language.

6° What is your best award ? Tell us more.

The Charles Stark Draper Prize from the US National Academy of Engineering (https://en.wikipedia.org/wiki/Charles_Stark_Draper_Prize). It's one of the world's highest honors for an engineer. It was founded to compensate for the fact that there is no Nobel Prize for engineering. Very few computer scientists have received 'The Draper' ; it is for all fields of engineering. The engineers always appreciated my work and I am proud of having done something that's useful at a large scale.

7° Can you give us your feeling when young developers think about C++ as old legacy stuff compared to Rust, Kotlin and more ?

Everyone like something new and shiny and we'd all like to ignore the messy real-world complexities, many of which relates to past work. I'm somewhat amused when someone accuses me of borrowing something from Rust that I invented decades earlier. Every successful language will become "legacy" and have to deal with older ideas and older code; the alternative is failure.

8° What is the most important features in language that will be popular for you ?

The most important single feature in C++ is the class with constructors and destructors to handle resource management. That's a feature that distinguishes C++ from most other language and it is foundational to C++ programming. After that, I'll point to templates with concepts to support generic programming.

It is typically a mistake to focus on individual language features. I often characterize C++ like this

- A static type system with equal support for built-in types and user-defined types
- Value ***and*** reference semantics

- Systematic and general resource management (RAII)
- Support for efficient object-oriented programming
- Support for flexible and efficient generic programming
- Support for compile-time programming
- Direct use of machine and operating system resources
- Concurrency support through libraries (where necessary implemented using intrinsics)

This is for C++, of course ; other languages are designed for different application domains and different developer populations, so they have different design criteria.

Core C++ Questions

- assignment:
  - Is move semantic a good thing (what about the fuzzy &&) ?

Move semantics is a very good thing in that it completes C++'s resource management model by allowing resources to be move from scope to scope with little or no overhead and without fiddling with pointers or explicit resource management. This dramatically simplifies code and completes the developments started by the return value optimization (which I implemented in 1984). I almost exclusively use move semantics implicitly through return statements where it is safe and implicit once you have defined move constructors and move assignments. I am very suspicious about explicit use of && and std::move() because such use is error-prone and usually unnecessary.

- `class`:
  - Still the heart of c++ ; express what you think !
  - Do we need enhancements to classes ?

Classes with constructors and destructors (when needed) are the heart of C++. I think they are pretty good and don't need significant improvements.

- `concept`:
  - Tell us more about C++ 20 concepts

When I designed templates, I wanted three things:

- Generality – to allow people to do more that I could imagine
- Zero overhead
- Precisely-specified interfaces

I didn't know how to do all three simultaneously, so we got just the first two. However, the first two were enough for templates to become a major success, but also caused a lot of problems and confusion. "Concepts" gives us the that last point: precisely-specified interfaces without run-time overheads or restrictions on what you ca express; it's simply compile-time predicate logic on properties of types and values.

Finally, we can say just

sort(v);

And get a decent error message if **v** isn't sortable. We might declare **sort()** like this:

void sort(sortable_range auto&);

That **auto** is redundant, but many standards committee members felt that leaving it out would confuse people because then they wouldn't be able to see that **sort()** was a template.

- constructor:
  - A good constructor is easy to use. It's not easy.

???

- Coroutines
  - Tell us more about C++ 20 coroutines

Coroutines was part of the original conception of C++ and the very first C++ library (when C++ was still called "C with Classes") provided coroutines. I'm very pleased to see them back. Coroutines makes many forms of otherwise complicated code simple by allowing state state of a computation to be automatically saved between computations. The runtime representation of C++20 coroutines is very small and efficient so that they can be used to handle hundreds of thousands of asynchronous computations in a single program. They are already in heavy use in Facebook and Microsoft.

- derived `class`:
  - then you realize if it's well designed… when you use it or derive it.
  - Do we need enhancements to this stuff?

No. I don't think we need to add to the derived class mechanisms.

- destructor:
  - When the dirty job is done for you when you have called Close() or not…

Probably my favorite C++ feature. It is key to general resource management. Just managing memory isn't sufficient. We need to handle non-memory resources such as file handles, locks, and database connections.

- exception:
  - I am not an exception guy but I like the catch all… (…)

Well, I am "an exception guy" and I don't mind "catch(…)."

Exception used with RAII immensely simplifies code and eliminates whole classes of errors at very minor costs. Sometimes, it provides speedups compared with code littered with tests of error codes. It keeps exceptional control flows from complicating the source code.

Most problems with exceptions come from people who use try-catch as simply a form of if-then-else or in a codebase that is such a mess of pointers and manual resource management that exceptions become a source of errors and run-time cost. Another class of problems

comes in systems that don't try to catch errors and in tiny systems where any form of run-time support can crowd out needed functionality.

- `for`:
    - for-range is quite cool. Tell us more !

What more is there to tell ? Traversing a range is one of the most common actions in our code so we should have a simple and relatively safe way of expressing it. Most of the common errors with traditional C for loops cannot be made with range-for and it is easier to optimize. Together with **std ::span()** is eliminates most buffer-overflow style errors.

- function:
    - lambda is good. I appreciate a lot for STL stuff.

Yes. They have become key to callbacks of all sorts, including specifying arguments for algorithms parameterized with predicates and other operations. Lambdas are often faster and more convenient than alternatives. Note that in C++, a lambda is simply a convenient notation for defining and instantiating a function object.

- operator:
    - operator overloading is quite good ; it's wonderful.

Indeed. I couldn't do without overloading (for function and operators). They are one of the key to generic programming and to many forms of elegant code.

- `public`, `private`, and `protected`:
    - the separation between things can be marvelous, and sometimes a friend comes at the rescue!

Indeed, though I don't find all that many uses for "protected" these days.

- `template`:
    - the hard stuff of C++
    - with the experience, it's the pure beauty of abstraction, with the specialization feature.
    - Do we need enhancements to templates ?

We need to polish a few rough corners on the use of contracts, but otherwise I think we are fine for now. For example, I'd like to be able to constrain template arguments in variable definitions:

```
pair<integral,derived_from<Shape>*> p = {27,new Circle{p,30}};
```

With concepts many of our designs will become simpler and easier to use.

If we don't need to constrain, we can simplify

```
pair p = {27,new Circle{p,30}};    // p becomes a pair<int,Circle*>
```

We have had template argument deduction since C++17.

- `virtual`:

- We could not live without it…

Well, quite often we can. Class hierarchies and virtual functions are great, but somewhat overused.

Various questions

1° When I teach C++ lessons, I tell my students that C++ is everywhere. A friend of mine was in New-York in June and saw the 66 seconds about “the engine of everything” with you in picture. I was so happy. It’s so the truth. From medical to military, from databases to operating systems, from virtual machines (Java, NET) to Office suite, from industry to video games : EVERYWHERE ! Herb Sutter told “the world is built on C++”. What about to be a 40-years old personage (C++) and be so simple : no starlight, no ads, no commercial machine, no ownership and a wide adoption by all tech companies. How do you feel when you hear that and because you know the truth ?

Obviously, I feel happy when I see C++ being so widely used to do well. I also feel a bit scared because it is a great responsibility. C++ really is just about everywhere. Mostly it is invisible as part of some system that we depend on, say our phone, our television set, our movies and games, our car, our bank, our camera, the farm that produces our food.

Though it was an honor, and not an ad for C++, seeing myself on a three-story-tall video display in Times Square was a bit spooky.

2° When tech people think about C++, they think about C and pointers. I know that, my students tell me. When they realize that smart pointers can do the job, with RAII, they are with no voices. All they have learned with Java and NET is useless.

Well, those other languages have their place, but yes, it’s sad that many have a completely warped view of C++. I wrote “A Tour of C++” to help people get a feel for modern C++. It has the obvious advantage of being thin. If you have already mastered programming, you can read it over a weekend.

3° Some weeks ago, on the MVP Channel, I send a passionate post about Rust popular usage and that some part of the kernel may be written in Rust for Windows Operating systems. Some researcher (who has never written an O.S) was arguing that. Herb replied me and I liked the way he wrote it. He called those guys the believers: like the project singularity, longhorn all those R&D efforts but cost effort. Longhorn was a trauma for Microsoft and was stopped at Vista pre-released. It was planned to incorporate NET stuff into the OS at kernel ad user layers. Windows Division reported that NET was to weak, not robust, not reliable and to slow and asked the Developer Division to address these issues. Because, it was not possible, WinDiv ignore DevDiv and it’s still the war 15 years later. How do you feel when all new students learn only Java and NET at school? In my opinion, it’s a shame. What is your feeling with those so-called productive languages like Java and NET/C#?

I don’t do language comparisons. They are hard to do well. I think many would benefit from being taught modern C++. Sadly, much C++ teaching is stuck in the 1980s. The C++ standards committee (WG21) now have a study group on education, aiming at providing some easily accessible guidelines for teaching.

4° How did you planned to write the Core C++ Guidelines with Herb Sutter on isocpp github ? Can you give us the under the scene aspect of that ? It could be Addison Wesley C++ Series book but it’s free. Was it wanted ? The price to be again the number one is to be free ?

I was working on a set of guidelines for Morgan Stanley based on the advice sections in my books (e.g., “A Tour of C++”) with the aim of making them widely available. It occurred to me that I could not be the only one doing that. After all, C++11 and C++14 were bringing many new people to modern C++ and they needed guidlines. So I asked around and Herb had started similar work at Microsoft. Obviously, to be able to collaborate freely and to share our work on a timely basis, we made it an open-source project. The guidelines and the tiny support library (GSL) are on Github:

- https://github.com/isocpp/CppCoreGuidelines
- https://github.com/microsoft/gsl

The (essential) static analysis support is mainly in Visual Studio with a bit of support in Clang Tidy.

This is an ambition project. For starters we want to ensure a style of C++ that’s type-safe and resource-safe. Beyond that, we’d like to eliminate needless complex use and common performance problems.

The Core Guidelines is another good way of looking at modern C++. Don’t read it all at once, though. It’s meant to be used with a static analyser, but the introduction can be quite helpful and the individual guidelines can be used much as an FAQ. Obviously, people are encouraged to contribute. It’s not just Herb and me. We have many contributors and several regular editors.

5° Some people in marketing, said some years ago that C++ is “unsafe and unsecure” ? I explain them that smart pointers, auto , nullptr, vector ,string do the job and it’s a non-sense to emit such a sentence. What is your opinion again a so stupid sentence?

Safety and security are system properties. Some people seem obsessed with the possibility of language insecurities, but every language that can manipulate hardware directly will have some. Any language that can use the operating systems API directly have some. If I wanted to break into a computer, I wouldn’t start messing with C++, there are easier ways in.

That said, most of the things that people complain about can be avoided in modern C++; just look at the C++ Core Guidelines (and enforce them). The feature you mention are among the features that helps.

6° The engine of everything. It’s so cool. C++ is a diamond tool. For people who need to allocate dynamic memory it fits, for people who need constant memory it fits, for building all kind of software fits. With the success of native tooling and specially LLVM or GCC backend, a series of languages adopt the native style because it’s that that works well and works best. What are you feeling with the eco-system of programming languages that go native?

Hard to say because there is not just one such ecosystem. It is actually nice to see all those new languages built on a C++ infrastructure.

7° Some companies invest in Java, Kotlin, others Rust, others Go, others Swift. What do you think of diversity ?

It’s good to see developments in programming languages and programming techniques. I just wish more people would realize that there is no language that is best for everything and

everybody and try to learn from the best in all and benefit from the variety of languages where they have their strength. C++ is often “the one to beat.” There is a reason for that: C++ is pretty good for many things, efficient, and stable. Any language that survives for a couple of decades will have some of C++’s problems.

8° Everything on my pc laptop is made with C/C++ : Windows, Sysinternals Suite, Office (Word, Excel, PowerPoint, Outlook), Chrome, VLC, Winamp, Gimp, Acrobat Reader, SQL Server, my Video games. ALL. Is there best achievements? C++ rulez the world.

Personally, I am happiest with C++’s role in science and engineering: Think of CERN, the human genome project, the Mars Rovers.

Also, C++ is spreading into new application domains. For example, through TensorFlow, it is the foundation for most AI/ML.

Additional questions

Can I pick up some sentences you will tell us to put in the foreword of my C++ 20 french book and explain it was extracted from an interview you gave with me and Programmez Magazine ? Do you give me your authorization ? The book will be shipped in May 2020.

ChristopheP