Doc. no. P2411r0
Date: 2021-07-12
Project: Programming Language C++
Audience: All
Reply to: Bjarne Stroustrup (bs@ms.com)

Thoughts on pattern matching

Bjarne Stroustrup

General observations

If we do this right, it will be a major improvement to C++ and relatively soon most code will look very different from today, making C++ attractive beyond its current user base. We should do this.

If we do this poorly, if will still be widely used and C++'s reputation for complexity will get another boost. Maybe experts would love it, but the net effect would be to lose users. We must avoid that.

If we provide a very attractive feature that has poor performance relative to alternatives, C++'s reputation for efficiency will be compromised. That would be a disaster.

Pattern matching offers a major opportunity to simplify C++ code through

- Generalization, providing a unified and simpler notation for what is currently many different notations for similar operations.
- New, powerful, simpler, and less error-prone ways of expressing code involving alternatives.
- Making code more generic through cleaner, more uniform, notation.

It is essential to provide a simple, elegant, and efficient way of using types that encapsulate their data through pattern matching. If that's not the case, the language, its library, and its user community will fracture between users of unencapsulated (algebraic) types and encapsulated types. In other words, between functional programming and object-oriented programming.

Specifics

Pattern Matching must absorb structured binding (as it was always meant to [P0144R0]). The mapping from an encapsulating type to a set of values used by pattern matching must be simple and declarative. The use of **get<>()** for structured binding is an expert-only mess. Any code-based, as opposed to declarative, mapping will have such problems in use and complicate optimization. We can do much better.

The 'is'-and-'as' notation [P2392] is cleaner and more general than the [P1371] and successors notation. For example, it eliminates the need to use the different notation styles for variant, optional, and any access. Uniform notation is the backbone of generic programming.

The use of the 'is'-and-'as' notation outside inspect expressions is a useful "over the horizon" design exercise for keeping the pattern matching design clean and general, but need not be ready for C++23.

I'm not against having such a generalization in C++23, but we need to be very careful that we have a complete solution to "the design puzzle." It would be unfortunate to have to spend a couple of years cleaning up major messes caused by lack of foresight. There will – as ever – be small improvements to do after the initial release based on post-release experience, but major mistakes can and must be avoided.

We must be careful not to complicate the pattern-matching design and notation by forcing rare special cases (from existing code) into the pattern-matching notion. I'm thinking of uses of **goto**, **reinterpret_cast**, and **variant<int,int,int>**. Such cases can be handled by more traditional techniques or by adapters. For example, when considering the 'as' operator, note that it is not intended to handle all existing casts, just the ones that are type safe.

We must try hard not to confuse familiarity (with other C++ facilities or pattern matching in other languages) with simplicity.

We have to be extremely focused and hard-working to get pattern-matching into C++23, and probably also lucky. Many would like to see an implementation and some use before approving a design.

We should avoid defining pattern-matching operations in terms of calls to existing library code. Doing so would become an obstacle to optimization, could force us to handle ancient warts, and could become an obstacle to the improvement of the library facilities used.

Final words

I am still of the opinion that after a general model of concurrency, a module version of the standard library, and library support for coroutines, pattern matching is the most promising addition to the language for the (relatively) near future [P2000].

References

- [P1371] B. Cardoso Lopes, S. Murzin, M. Park, D. Sankel, D. Sarginson, B. Stroustrup: Pattern Matching (R3). 2020-09.
- [P0144R0]: H. Sutter, B. Stroustrup. G. Dos Reis: Structured bindings. 2015-10-14.
- [P2000] M. Wong, Hinnant, R. Orr, B. Stroustrup, D. Vandevoorde: DIRECTION FOR ISO C++. 2020-07-15. Especially §5.3.
- [P2392] H. Sutter: Pattern matching using is and as. 2021-06-13.
- Original Patter Matching presentation: Y. Solodkyy, G. Dos Reis, and B. Stroustrup: Pattern Matching for C++. Evening session at WG21 meeting in Urbana-Champaign, Illinois. Nov. 2014.