Natural Syntax: Keep It Simple

Gabriel Dos Reis
Microsoft

We have controversies surrounding the natural syntax for introducing a constrained template parameter in a way that is less verbose than the full-fledged template parameter declaration syntax. The natural syntax is necessary to keep simple cases simple. For example, the definition of the function `increment()` from the EoP [3, page 91] could be written using one of the Concepts TS syntaxes as

```
void increment(Iterator& x)
{
      x = successor(x);
}
```

where `Iterator` designates a concept, and the use of that name to declare the function parameter 'x' is interpreted as standing for a type template parameter. That declaration could equivalently be written as

```
template<Iterator I>
void increment(I& x)
{
      x = successor(x);
}
```

which is itself equivalent to the full-fledged constrained template declaration

```
template<Iterator I> requires Iterator<I>
void increment(I& x)
{
      X = successor(x);
}
```

Some part of the community wants some sort of visual clue to distinguish uses of concept names as type variables, from concrete types. See full discussions in [4, 5]. It should be noted that there is no actual grammatical ambiguity since every single use of a name is subject to name lookup, and the rules are unambiguous for C++ implementations to distinguish concept names from concrete type names. The issue here is whether some *visual* marker is needed to aid human reading. After a protracted dispute, EWG is headed toward some form of marker somewhere. At the Jacksonville meeting, a few proposals [1,2,6] were discussed in an EWG evening session that included "multi-choice" voting. The proposal P0734R0 was leading the pack; however a strong majority asked for specific concerns to be addressed, hence the suggestions in the paper P1079R0 which was circulated among committee members who expressed concerns at the Jacksonville evening session, prior to the pre-Rapperswil mailing deadline.

Another aspect of the controversies is how to interpret repeated mentions of a concept name in a declaration using the natural syntax. For example, could the `subtractive_gcd()` from from the EoP [3, page 78] be declared as

```
    EuclideanMonoid subtractive_gcd(EuclideanMonoid a, EuclideanMonoid b);
```

or does it need at least the form

```
    template<EuclideanMonoid T>
    T subtractive_gcd(T a, T b);
```

to state that both function parameters have the same type?

It should be emphasized that the natural syntax has no mission or requirement to express just about every declaration nuance that the full-fledged constrained template declaration permits. Nor should we embark on such mission, in my opinion.

I write this note to caution the C++ community and WG21 against what I see as a repeat of mistakes that we routinely and collectively make in WG21 design: verbose syntax and over-design that we come to regret later; as an example, I will mention requirement of "`typename`" for nested dependent types when it is obvious what is meant; thankfully that issue is now fixed for C++20 [7]. We are about to make similar mistakes, but at a larger scale if we don't correct course.

My suggestion is to keep the natural syntax simple. **Any modification to the Concepts TS syntax should be minimal, any new notational burdens should be placed on the less common use cases, and the common cases should be kept simple. Furthermore, visual clues that do not remove actual ambiguities should be optional, not compulsory.** On that basis, it strikes me that the proposal P0734 makes the wrong design tradeoffs; it leads to obtuse verbosity, confusion, and violates what I would consider basic language design principles. The solution presented in P1079R0 appears to me as a much better-balanced solution to the controversies. I urge WG21 to correct course, reconsider the issue and adopt the solutions proposed in P1079R0. I can live we a permitted (but not required) "auto" in the declaration of a function using the natural syntax if that helps people quickly identify that a function declaration declares a template. However, we should keep such permission simple and not embark on other complexities. I consider the brace notation for introducing template parameters a wrong and over-engineered solution to this problem, and actively harmful to the purpose of concepts for C++. Simplicity is a feature. Let's keep it simple. Let's bring Generic Programming to the masses.

References

1. Erich Keane, Adam Martin, Allan Deutsch: "A Case for Simplifying/Improving Natural Syntax Concepts"; WG21, P0782R0.
2. Thomas Koeppe: "*An adjective syntax for concepts*"; WG21, P0791R0.
3. Alexander Stepanov, Paul McJones: "*Elements of Programming*"; Addison-Wesley, 2009.
4. Bjarne Stroustrup: "*Answers to concept syntax suggestions*"; WG21, P0956R0.
5. Bjarne Stroustrup: "*A minimal solution to the concept syntax problems*"; WG21, P1079R0.
6. Herb Sutter: "*Concepts in-place syntax*"; WG21, P0734R0.
7. Nina Ranns, David Vandevoorde: "*Down with typename!*"; WG21 P0634R3.