Disallow _ Usage in C++20 for Pattern Matching in C++23

Document #:	P1469R0
Date:	2019-01-21
Project:	Programming Language C++ Evolution

Reply-to:	Sergei Murzin <smurzin@bloomberg.net> Michael Park <mcypark@gmail.com> David Sankel <dsankel@bloomberg.net> Dan Sarginson <dsarginson@bloomberg.net>

Contents

1 Introduction

We need to deprecate a rare usage of _ as an identifier in C++20 so it can be safely used for pattern matching [P1371] which is targeting C++23. The issue is simple, we'd like structured binding code as in,

```
auto [a, _] = std::make_pair(3, 4);
```

to have _ represent a wildcard pattern instead of binding the identifier _. There are a few ways to do this and we suggest what we think is the ideal engineering approach.

2 Various Approaches

2.1 Use something besides _ (BAD)

Why is _ so important when ? is available? Languages with pattern matching almost universally use _ as a wildcard pattern and popular libraries in C++ (like Google Test) do the same. It would be awkward and somewhat embarrassing if C++ were to not use such a ubiquitous token. Furthermore, because _ has so much existing widespread use, we expect people to use _ anyway, and accidentally bind the _ identifier.

__ is another possiblility, but it is difficult to recognize as a double underscore with many fonts and we expect significant confusion were we to use this.

2.2 Deprecate all or most uses of _ as an identifier (WORSE)

Another option is to deprecate all or most uses of _ as an identifier. We could, for instance, deprecate references to _ identifiers declared in block-scope. The migration cost for this is unfortunately quite a steep one. There are more instances of the _ token than `short` in the wild according to [ACTCD16]. Breaking all this code would be extremely costly for the C++ community.

2.3 Deprecate _ access when structured-binding bound (OKAY)

Finally, we could deprecate referencing _ identifiers that are bound as part of a structured binding. Because structured binding is a relatively new feature and most uses of _ as a structured binding identifier are expected to be wildcard uses anyway, the cost of fixing breakages would be low. This change to be a sweet spot for engineering value, but we need to act quickly if we want to take advantage of this for pattern matching in C++23.

2.4 Disallow _ access when structured-binding bound (GOOD)

Because usages of _ identifiers that are bound by structured binding are so rare, there is little engineering benefit in deprecating these usages instead of just making access illegal altogether. In fact, a deprecation period can actually increase cost as engineers (who ignore warnings) will have more opportunity to use _ identifiers bound by structured bindings.

3 Should we generalize this as in P1110R0?

[P1110R0] suggests that we make something like the wildcard (_) pattern accessible in many other places than patterns (e.g. in an enumerator). The authors of this paper find questionable engineering value for many of these suggestions and have no opinion on others. Our primary focus is on parity between structured bindings and pattern matching.

4 Conclusion

Right now we have an opportunity to prevent mistakes being made by using the wildcard that everyone expects for pattern matching. Let's deprecate or disallow access to _ variables if they are bound by structured binding.

5 References

[ACTCD16] Andrew Tomazos. Andrew's C/C++ Token Count Dataset 2016 (ACTCD16). http://www.tomazos.com/actcd16.pdf

[P1110R0] Jeffrey Yasskin and JF Bastien. 2018. A placeholder with no name. http://wg21.link/P1110R0

[P1371] Sergei Murzin, Michael Park, David Sankel, and Dan Sarginson. 2019. Pattern Matching. http://wg21.link/P1371