On empty structs in the standard library

Vincent Reverdy
Collin Gress

Overview

Summary

A "universal" default empty structure is useful in multiple contexts. The standard library currently does not have one.

Examples

```
// Variant
struct monostate {}; // empty struct
struct S {S(int i) : i(i) {} int i;};
variant<monostate, S> variant;

// Conditional inheritance
struct empty_base {}; // empty struct
template <class T> struct derived
: conditional_t<is_class_v<T> && !is_final_v<T>, T, empty_base> {};

// Need for a template version
template <class...> struct empty_base_template {}; // empty struct
template <class... T> struct derived
: conditional_t<is_class_v<T> && !is_final_v<T>, T, empty_base_template<T>>... {};
```

Currently

- `tuple<>`: in `<tuple>`, has a `swap` member, is not intended to be a "universal" empty structure
- `monostate`: in `<variant>`, is considered to be a helper class for `variant`
- `blank`: the "universal" empty structure of the Boost libraries

Question

Need for a "universal" empty structure in `<type_traits>` or `<utility>`?

Design options

Option 1: do nothing

Do nothing and do not introduce a "universal" empty structure: let users keep creating their own.

Option 2: promote monostate

Make `monostate` the "universal" empty structure and transfer it from `<variant>` to `<type_traits>` or `<utility>`.

Option 3: one empty structure per use case

Add a new empty structure for each new use case that is being standardized such as a new `empty_base` for conditional inheritance.

Option 4: a universal empty structure on top of monostate

Add a new "universal" empty structure, with a different name than `monostate` in `<type_traits>` or `<utility>`.

Design of the template version

Motivation

```
// Need for a template version
template <class...> struct empty_base_template {};
template <class... T> struct derived
: conditional_t<is_class_v<T> && !is_final_v<T>, T, empty_base_template<T>>... {};
```

Design question

- `struct empty_struct {}`: the non-template and the template versions are two independent structures
- `using empty_struct = empty_struct_template<>`: the non-template version is an alias of the template version

Conclusion

What option?

- Option 1: do nothing
- Option 2: promote monostate
- Option 3: one empty structure per use case
- Option 4: a universal empty structure on top of monostate

Non-template and template versions

- the non-template and the template versions are two independent structures?
- the non-template version is an alias of the template version?

Bikeshedding

- `empty_struct`
- `empty`
- `blank`
- `nothing`
- `none`
- `null`
- `nil`
- `vacant`
- `primal`
- . . . and what about the template version?

Additional functionalities

- Comparison operators?
- Hash specialization?

Thank you for your attention