`visit<R>`: Explicit Return Type for `visit`

Document #:	D0655R0
Date:	2017-10-14
Project:	Programming Language C++ Library Evolution Group

Reply-to:	Michael Park <mcypark@gmail.com> Agustín Bergé <agustinberge@gmail.com>

1 Introduction

This paper proposes to allow visiting `variant`s with an explicitly specified return type.

2 Motivation and Scope

Variant visitation requires invocation of all combinations of alternatives to result in the same type, such type is deduced as the visitation return type. It is sometimes desirable to explicitly specify a return type to which all the invocations are implicitly convertible to, as if by `INVOKE<R>` rather than `INVOKE`:

```
struct process {
  template <typename I>
  auto operator()(I i) -> O<I> { /* ... */ };
};

variant<I1, I2, I3> input = /* ... */;

// mapping from a `variant` of inputs to a `variant` of results:
auto output = visit<variant<O<I1>, O<I2>, O<I3>>>(process{}, input);

// coercing different results to a common type:
auto result = visit<common_type_t<O<I1>, O<I2>, O<I3>>>(process{}, input);

// visiting a `variant` for the side-effects, discarding results:
visit<void>(process{}, input);
```

In all of the above cases the return type deduction would have failed, as each invocation yields a different type for each alternative.

3 Impact on the Standard

This proposal is a pure library extension.

4 Proposed Wording

Add to **§23.7.2 [variant.syn]** of N4687 [1]:

```
  // 23.7.7, visitation
  template <class Visitor, class... Variants>
    constexpr see below visit(Visitor&&, Variants&&...);
+  template <class R, class Visitor, class... Variants>
+    constexpr R visit(Visitor&&, Variants&&...);
```

Add to **§23.7.7 [variant.visit]** of N4687 [1]:

```
+template <class R, class Visitor, class... Variants>
+  constexpr R visit(Visitor&& vis, Variants&&... vars);
```

Requires: The expression in the *Effects*: element shall be a valid expression of the same type and value category, for all combinations of alternative types of all variants. Otherwise, the program is ill-formed.

Effects: Let `is...` be `vars.index()...`. Returns `INVOKE<R>(forward<Visitor>(vis), get<is>(forward<Variants>(vars))...);`.

Throws: `bad_variant_access` if any `variant` in `vars` is `valueless_by_exception()`.

Complexity: For `sizeof...(Variants) <= 1`, the invocation of the callable object is implemented in constant time, i.e. it does not depend on `sizeof...(Types)`. For `sizeof...(Variants) > 1`, the invocation of the callable object has no complexity requirements.

5 Design Decisions

This paper draws inspiration from `bind`. // TODO(mpark)

References

[1] 2017. Working Draft, Standard for Programming Language C++. *N4687*. Retrieved from http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4687.pdf