P3619R1: Counter-examples for P2688R5

Sergey Anisimov
<s-anisimov-cxx@outlook.com>
Tymofii Kutlov
<tymofii.kutlov@outlook.com>
Vlad Serebrennikov
<serebrennikov.vladislav@gmail.com>

Example #1: `if` statement 1

C++17

```
std::tuple<int, int> f();
if (auto [a, b] = f(); a == 1) {
  a, b; // well-formed
} else {
  a, b; // well-formed
}
```

P2688R5

```
std::tuple<int, int> f();
if (f() match [1 let a, let b]) {
  a, b; // well-formed
} else {
  a, b; // unexpectedly ill-formed
}
```

Example #2: `if` statement 2

C++17

```
std::tuple<int, int> f();
bool j = should_bypass_check();
if (auto [a, b] = f(); a == 1 || j) {
  a, b; // well-formed
} else {
  a, b; // well-formed
}
```

P2688R5

```
std::tuple<int, int> f();
bool j = should_bypass_check();
if (f() match [1 let a, let b] || j) {
  a, b; // unexpectedly ill-formed
} else {
  a, b; // unexpectedly ill-formed
}
```

Example #3: Matching within template

C++17

```
template <typename T>
auto f1(T v) {
  return std::visit(
    overloaded{
      [](int){ return 1; },
      [](float){ return 2; },
      [](double){ return 3; },
    }, v);
}

// both are well-formed
f1(std::variant<int, float>{});
f1(std::variant<double, float>{});
```

P2688R5

```
template <typename T>
auto f2(T v) {
  // match expression cannot be
  // used for anything generic
  return v match {
    int: _ =>  1;
    float: _ => 2;
    double: _ => 3;
  };
}
// both are ill-formed
f2(std::variant<int, float>{});
f2(std::variant<double, float>{});
```

Example #4: Matching multiple types at once

Common code

```
using Variant = std::variant<
  std::array<int, 2>,
  std::array<int, 3>,
  std::array<float, 2>
>;
constexpr Variant v;
```

C++17

```
static_assert(std::visit(
  overloaded{
    [] <size_t N> (std::array<int, N>)
      { return true; },
    [] <size_t N> (std::array<float, N>)
      { return false; }
  }, v
));
```

P2688R5

```
static_assert(v match {
  std::array<int, 2>:   _ => true;
  std::array<int, 3>:   _ => true;
  std::array<float, 2>: _ => false;
});
```

P2688R5 + concepts

```
template<typename T, typename ValueT>
concept array_of = requires (T* x) {
  [] <size_t I> (std::array<ValueT, I>*){}(x);
};

static_assert(v match {
  array_of<int>:   _ => true;
  array_of<float>: _ => false;
});
```

Example #5: Attributes

Single identifier

```
42 match {
  // invented syntax, should be fine
  0 let x [[maybe_unused]] => true;
};
```

Whole match case

```
struct A {
  int a;
};
struct B {
  A b;
};

constexpr int attr = 24;

B{} match {
  // well-formed pattern (!)
  [[attr]] let x => true;
};
```