const correctness

```
auto& collection = store.create<ExampleMCCollection>("mc");
// collection -> ExampleMCCollection&
for (auto mc : collection) {
  mc.setEnergy(42); // OK
}

auto& const_collection = store.get<ExampleMCCollection>("mc");
// const_collection -> const ExampleMCCollection&
for (auto mc : const_collection) {
  mc.setEnergy(42); // NOT OK
}
```

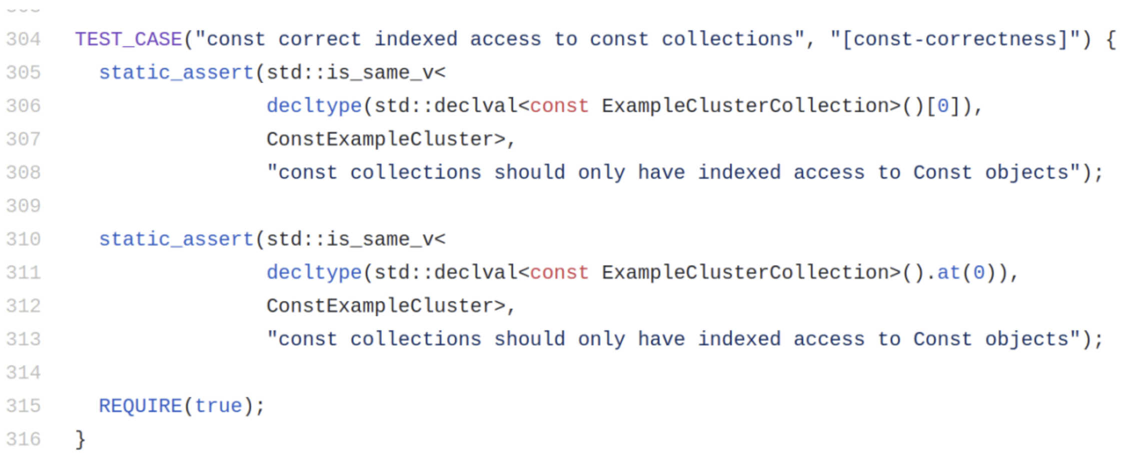
- In order to be const-correct, a `const` collection must not return any object that makes it possible to alter internal state of the collection through this object
- `const` assumes thread safe in c++ (>11). Not enforced by the compiler, but relied upon by STL
- Currently indexed access as well as iterator based access to collections always return mutable objects: AIDASoft/podio#176

Fix

- Make the indexed access method return a `Const` object from `const` collections
- Duplicate the `CollectionIterator` and also generate a `ConstCollectionIterator`
- Duplicate the `begin` and `end` members of the collections and make the `const` versions return a `ConstCollectionIterator`

Using `static_assert` to enforce things at compile time

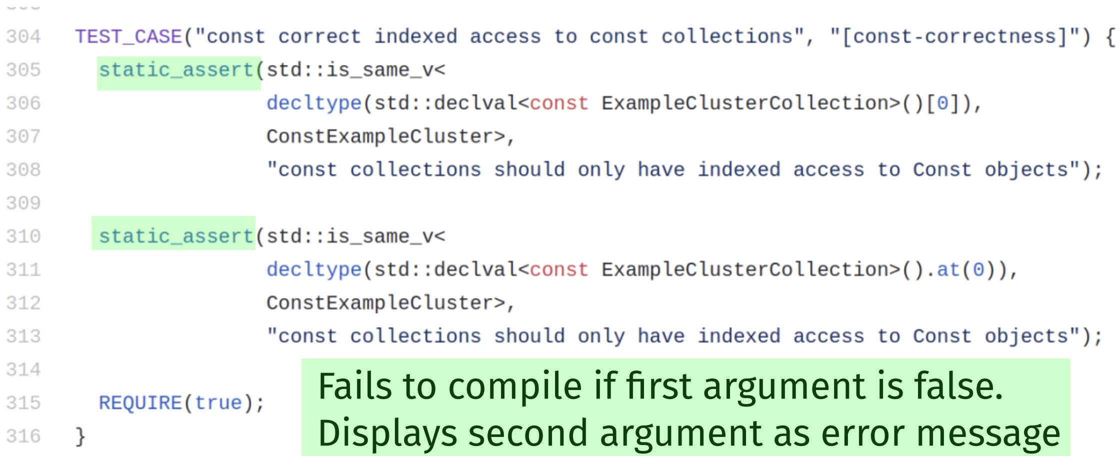
- “Compile-time” unittests. If it compiles it passed



```cpp
TEST_CASE("const correct indexed access to const collections", "[const-correctness]") {
  static_assert(std::is_same_v<
                decltype(std::declval<const ExampleClusterCollection>()[0]),
                ConstExampleCluster>,
                "const collections should only have indexed access to Const objects");

  static_assert(std::is_same_v<
                decltype(std::declval<const ExampleClusterCollection>().at(0)),
                ConstExampleCluster>,
                "const collections should only have indexed access to Const objects");

  REQUIRE(true);
}
```

Using `static_assert` to enforce things at compile time

- "Compile-time" unittests. If it compiles it passed

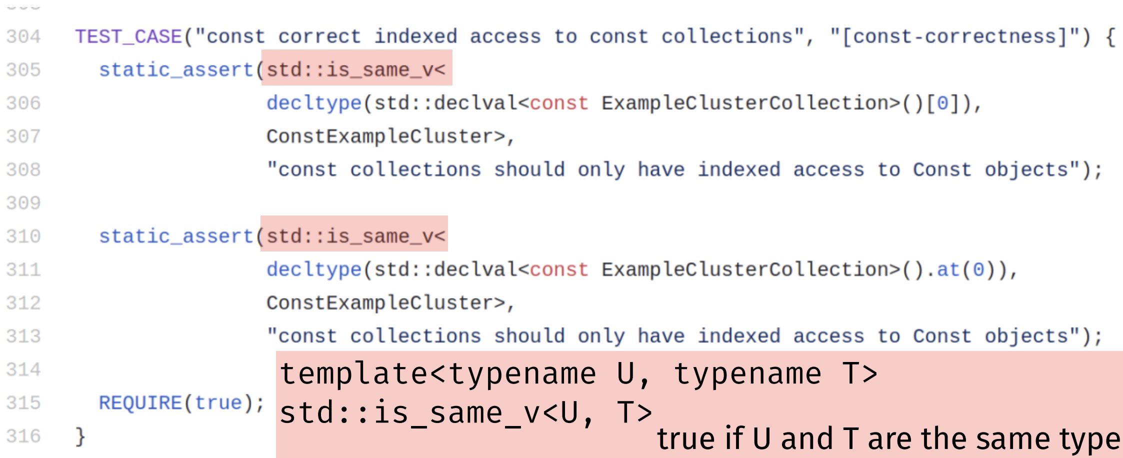


```cpp
TEST_CASE("const correct indexed access to const collections", "[const-correctness]") {
  static_assert(std::is_same_v<
                decltype(std::declval<const ExampleClusterCollection>()[0]),
                ConstExampleCluster>,
                "const collections should only have indexed access to Const objects");

  static_assert(std::is_same_v<
                decltype(std::declval<const ExampleClusterCollection>().at(0)),
                ConstExampleCluster>,
                "const collections should only have indexed access to Const objects");

  REQUIRE(true);
}
```

Fails to compile if first argument is false.
Displays second argument as error message

Using `static_assert` to enforce things at compile time

- “Compile-time” unittests. If it compiles it passed



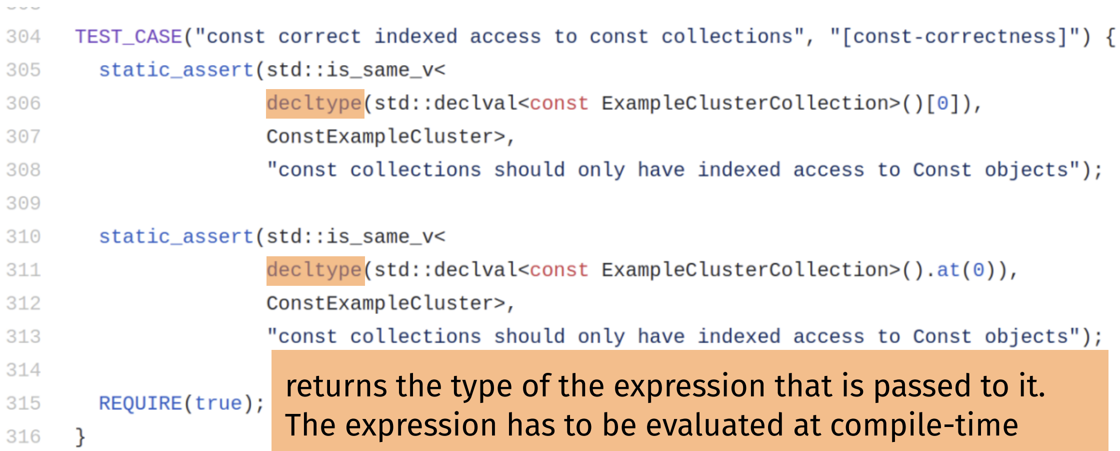
```cpp
TEST_CASE("const correct indexed access to const collections", "[const-correctness]") {
  static_assert(std::is_same_v<
                decltype(std::declval<const ExampleClusterCollection>()[0]),
                ConstExampleCluster>,
                "const collections should only have indexed access to Const objects");

  static_assert(std::is_same_v<
                decltype(std::declval<const ExampleClusterCollection>().at(0)),
                ConstExampleCluster>,
                "const collections should only have indexed access to Const objects");

  REQUIRE(true);
}
```

```cpp
template<typename U, typename T>
std::is_same_v<U, T>
```

true if U and T are the same type

Using `static_assert` to enforce things at compile time

- “Compile-time” unittests. If it compiles it passed

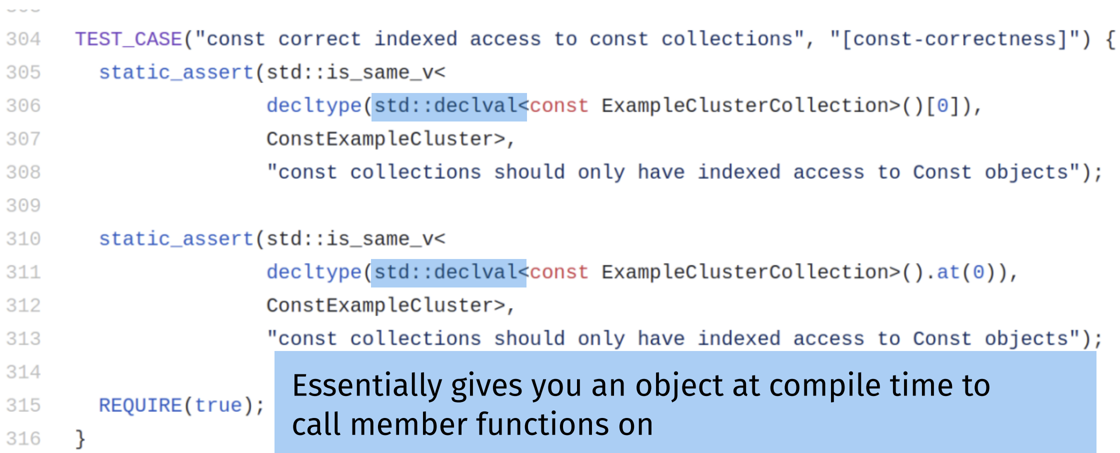


```cpp
TEST_CASE("const correct indexed access to const collections", "[const-correctness]") {
  static_assert(std::is_same_v<
                decltype(std::declval<const ExampleClusterCollection>()[0]),
                ConstExampleCluster>,
                "const collections should only have indexed access to Const objects");

  static_assert(std::is_same_v<
                decltype(std::declval<const ExampleClusterCollection>().at(0)),
                ConstExampleCluster>,
                "const collections should only have indexed access to Const objects");

  REQUIRE(true);
}
```

returns the type of the expression that is passed to it.
The expression has to be evaluated at compile-time

Using `static_assert` to enforce things at compile time

- “Compile-time” unittests. If it compiles it passed

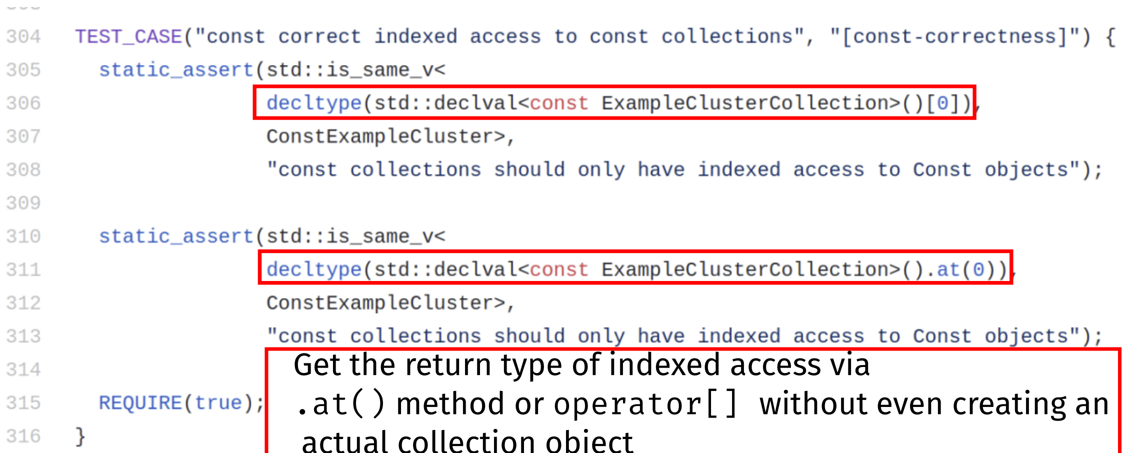


```cpp
TEST_CASE("const correct indexed access to const collections", "[const-correctness]") {
  static_assert(std::is_same_v<
                decltype(std::declval<const ExampleClusterCollection>()[0]),
                ConstExampleCluster>,
                "const collections should only have indexed access to Const objects");

  static_assert(std::is_same_v<
                decltype(std::declval<const ExampleClusterCollection>().at(0)),
                ConstExampleCluster>,
                "const collections should only have indexed access to Const objects");

  REQUIRE(true);
}
```

Essentially gives you an object at compile time to call member functions on

Using `static_assert` to enforce things at compile time

- “Compile-time” unittests. If it compiles it passed

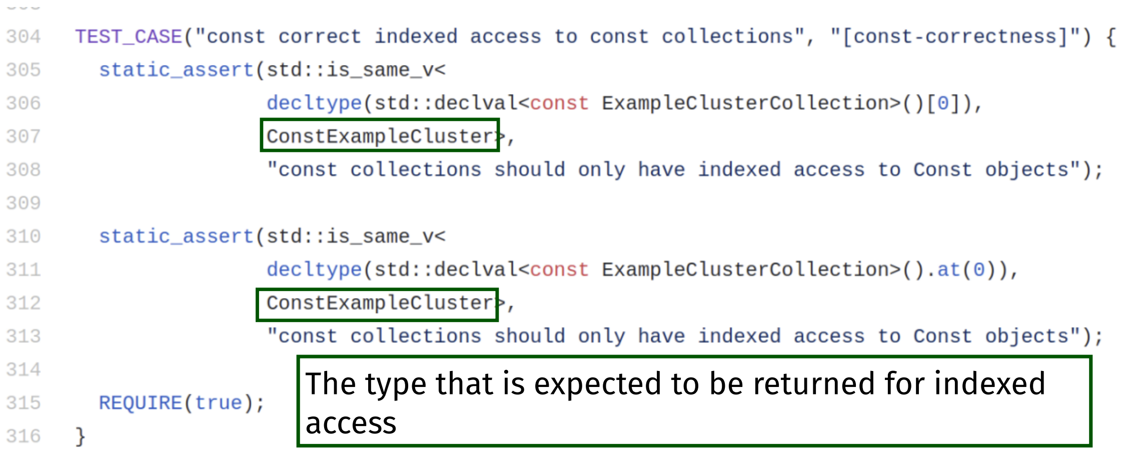


```
TEST_CASE("const correct indexed access to const collections", "[const-correctness]") {
  static_assert(std::is_same_v<
                decltype(std::declval<const ExampleClusterCollection>()[0]),
                ConstExampleCluster>,
                "const collections should only have indexed access to Const objects");

  static_assert(std::is_same_v<
                decltype(std::declval<const ExampleClusterCollection>().at(0)),
                ConstExampleCluster>,
                "const collections should only have indexed access to Const objects");

  REQUIRE(true);
}
```

Get the return type of indexed access via `.at()` method or `operator[]` without even creating an actual collection object

Using `static_assert` to enforce things at compile time

- “Compile-time” unittests. If it compiles it passed



```cpp
TEST_CASE("const correct indexed access to const collections", "[const-correctness]") {
  static_assert(std::is_same_v<
                decltype(std::declval<const ExampleClusterCollection>()[0]),
                ConstExampleCluster>,
                "const collections should only have indexed access to Const objects");

  static_assert(std::is_same_v<
                decltype(std::declval<const ExampleClusterCollection>().at(0)),
                ConstExampleCluster>,
                "const collections should only have indexed access to Const objects");

  REQUIRE(true);
}
```

The type that is expected to be returned for indexed access

Python ``equivalent";)

```
class ConstCorrectAccessTest(unittest.TestCase):
  def test_const_correct_access(self):
    collection = ExampleClusterCollection()
    self.assertTrue(isinstance(collection[0], ConstExampleCluster))
```

- But this needs an actual collection with at least one element
- Cannot check enforce this at compile time, if it fails it fails only at runtime