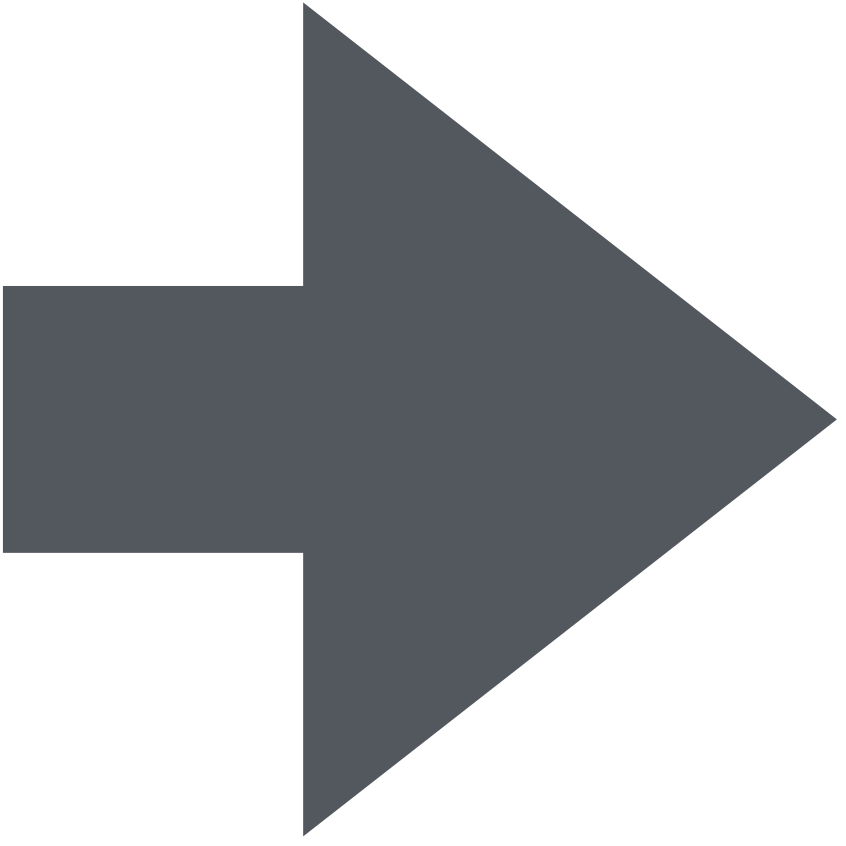
107

# Wrapped Data Race Detection

```
struct person(data_race_checker)
{
    std::string get_first_name() const
    {
        return first_name;
    }

    void set_first_name (std::string_view new_first)
    {
        first_name = new_first;
    }

    // Repeat for last_name

private:
    std::string first_name, last_name;
};
```

```
struct person
{
    std::string get_first_name() const
    {
        scoped_check<check_type::read> _ (check_state);
        return person_.get_first_name();
    }

    void set_first_name (std::string_view new_first)
    {
        scoped_check<check_type::write> _ (check_state);
        person_.set_first_name (new_first);
    }

    // Repeat for last_name

private:
    struct __person;
    __person person_;
    mutable check_state check_state;
};
```

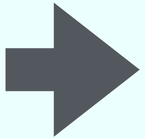


data_race_checker metaclass

# Wrapped Data Race Detection

data_race_checker metaclass

```
struct person(data_race_checker)
{
    std::string get_first_name() const
    {
        return first_name;
    }

    void set_first_name (std::string_view new_first)
    {
        first_name = new_first;
    }

    // Repeat for last_name

private:
    std::string first_name, last_name;
};
```



```
struct person
{
    std::string get_first_name() const
    {
        scoped_check<check_type::read> _ (check_state);
        return person_.get_first_name();
    }

    void set_first_name (std::string_view new_first)
    {
        scoped_check<check_type::write> _ (check_state);
        person_.set_first_name (new_first);
    }

    // Repeat for last_name

private:
    struct __person;
    __person person_;
    mutable check_state check_state;
};
```

# std Containers

## 16 Library introduction

### 16.4 Library-wide requirements

#### 16.4.6 Conforming implementations

##### 16.4.6.10 Data race avoidance

1 This subclause specifies requirements that implementations shall meet to prevent data race function shall meet each requirement unless otherwise specified. Implementations may pr other than those specified below.

2 A C++ standard library function shall not directly or indirectly access objects ([intro.multithread]) accessible by threads other than the current thread unless the objects are accessed directly or indirectly via the function's arguments, including `this`.

3 A C++ standard library function shall not directly or indirectly modify objects ([intro.multithread]) accessible by threads other than the current thread unless the objects are accessed directly or indirectly via the function's non-const arguments, including `this`.

4 [*Note 1*: This means, for example, that implementations can't use an object with static storage duration for internal purposes without synchronization because doing so can cause a data race even in programs that do not explicitly share objects between threads. — *end note*]

5 A C++ standard library function shall not access objects indirectly accessible via its arguments or via elements of its container arguments except by invoking functions required by its specification on those container elements.

6 Operations on iterators obtained by calling a standard library container or string member function may access the underlying container, but shall not modify it.

[*Note 2*: In particular, container operations that invalidate iterators conflict with operations on iterators associated with that container. — *end note*]

7 Implementations may share their own internal objects between threads if the objects are not visible to users and are protected against data races.

8 Unless otherwise specified, C++ standard library functions shall perform all operations solely within the current thread if those operations have effects that are visible to users.

9 [*Note 3*: This allows implementations to parallelize operations if there are no visible side effects. — *end note*]

## 23 Containers library [containers]

### 23.2 Requirements [container.requirements]

#### 23.2.3 Container data races [container.requirements.dataraces]

1 For purposes of avoiding data races ([res.on.data.races]), implementations shall consider the following functions to be `const`: `begin`, `end`, `rbegin`, `rend`, `front`, `back`, `data`, `find`, `lower_bound`, `upper_bound`, `equal_range`, `at` and, except in associative or unordered associative containers, `operator[]`.

2 Notwithstanding [res.on.data.races], implementations are required to avoid data races when the contents of the contained object in different elements in the same container, excepting `vector<bool>`, are modified concurrently.

3 [*Note 1*: For a `vector<int> x` with a size greater than one, `x[1] = 5` and `*x.begin() = 10` can be executed concurrently without a data race, but `x[0] = 5` and `*x.begin() = 10` executed concurrently can result in a data race. As an exception to the general rule, for a `vector<bool> y`, `y[0] = true` can race with `y[1] = true`. — *end note*]