

## std::remove\_cvref

### Summary

---

#### Before C++20

```
typename std::remove_cv<
    typename std::remove_reference<T>::type>::type
std::remove_cv_t<std::remove_reference_t<T>>
```

#### After C++20

```
typename std::remove_cvref<T>::type
std::remove_cvref_t<T>
```

---

### Detail

std::decay<T> is often (mis)used to obtain the type name for a type that is potentially a reference, cv-qualified, or both:

```
template <typename T>
auto f(T&& t) {
    // Removes reference and cv-qualifiers...
    using type_name = std::decay_t<T>;

    // To avoid this problem...
    static_assert(!std::is_same_v<int&, int>);
    static_assert(!std::is_same_v<int const, int>);

    // When evaluating the following conditional expression.
    if constexpr (std::is_same_v<type_name, int>) {
        use_int(t);
    }

    // Similarly, a reference type is not an array...
    else if constexpr (std::is_array_v<type_name>) {
        use_array(t);
    }

    // Nor is it a function.
    else if constexpr (std::is_function_v<type_name>) {
        use_function(t);
    }

    else {
        use_other(t);
    }
}

int main() {
    auto const i = 42;
    auto const s = std::string();
```

```

    f(i); // T == 'int const&', type_name == 'int'; invokes 'use_int'.
    f(s); // T == 'std::string const&', type_name == 'std::string'; invokes 'use_other'.
}

```

However, `std::decay<T>` introduces decay semantics for array and function types:

```

int arr[3];

// T == 'int (&)[3]', but type_name == 'int*' due to array-to-pointer decay.
// 'std::is_array_v<int*>' is false; invokes 'use_other'.
f(arr);

void g();

// T == 'void (&)()', but type_name == 'void (*)()' due to function-to-pointer decay.
// 'std::is_function<void (*)()>' is false; invokes 'use_other'.
f(g);

```

In C++20, the standard library provides the `std::remove_cvref<T>` type trait to fulfill this purpose without introducing unwanted decay semantics:

```

template <typename T>
auto f(T&& t) {
    using type_name = std::remove_cvref_t<T>;

    ...
}

int main() {
    auto const i = 42;
    int arr[3];
    void g();

    f(0); // Same as before.
    f(i); // Same as before.

    f(arr); // T == 'int (&)[3]', type_name == 'int [3]'; invokes 'use_array'.
    f(g); // T == 'void (&)()', type_name == 'void()'; invokes 'use_function'.
}

```