#include <iostream>

#include <iomanip>

#include <vector>

#include <numeric>

#include <chrono>

volatile int sink;

int main()

{

[std::cout](http://en.cppreference.com/w/cpp/io/cout) << [std::fixed](http://en.cppreference.com/w/cpp/io/manip/fixed) << [std::setprecision](http://en.cppreference.com/w/cpp/io/manip/setprecision)(9) << [std::left](http://en.cppreference.com/w/cpp/io/manip/left);

for (auto size = 1ull; size < 1000'000'000ull; size \*= 100) {

// record start time

auto start = std::chrono::high\_resolution\_clock::now();

// do some work

[std::vector](http://en.cppreference.com/w/cpp/container/vector)<int> v(size, 42);

sink = [std::accumulate](http://en.cppreference.com/w/cpp/algorithm/accumulate)(v.begin(), v.end(), 0u); // make sure it's a side effect

// record end time

auto end = std::chrono::high\_resolution\_clock::now();

[std::chrono::duration](http://en.cppreference.com/w/cpp/chrono/duration)<double> diff = end - start;

[std::cout](http://en.cppreference.com/w/cpp/io/cout) << "Time to fill and iterate a vector of " << [std::setw](http://en.cppreference.com/w/cpp/io/manip/setw)(9)

<< size << " ints : " << diff.count() << " s**\n**";

}

}

[std::chrono::duration](http://en.cppreference.com/w/cpp/chrono/duration)<double> diff = end - start;

[std::cout](http://en.cppreference.com/w/cpp/io/cout) << "Time to fill and iterate a vector of " << [std::setw](http://en.cppreference.com/w/cpp/io/manip/setw)(9)

<< size << " ints : " << diff.count() << " s**\n**"

;

Possible output:

Time to fill and iterate a vector of 1 ints : 0.000006568 s

Time to fill and iterate a vector of 100 ints : 0.000002854 s

Time to fill and iterate a vector of 10000 ints : 0.000116290 s

Time to fill and iterate a vector of 1000000 ints : 0.011742752 s

Time to fill and iterate a vector of 100000000 ints : 0.505534949 s

#include <iostream>

#include <type\_traits>

template<typename F, typename Class>

void ptr\_to\_member\_func\_cvref\_test(F Class::\*)

{

// F is an "abominable function type"

using FF = std::add\_pointer\_t<F>;

static\_assert([std::is\_same\_v](http://en.cppreference.com/w/cpp/types/is_same)<F, FF>, "FF should be precisely F");

}

struct S

{

void f\_ref() & {}

void f\_const() const {}

};

int main()

{

int i = 123;

int& ri = i;

typedef std::add\_pointer<decltype(i)>::type IntPtr;

typedef std::add\_pointer<decltype(ri)>::type IntPtr2;

IntPtr pi = &i;

[std::cout](http://en.cppreference.com/w/cpp/io/cout) << "i = " << i << "**\n**";

[std::cout](http://en.cppreference.com/w/cpp/io/cout) << "\*pi = " << \*pi << "**\n**";

static\_assert([std::is\_pointer](http://en.cppreference.com/w/cpp/types/is_pointer)<IntPtr>::value, "IntPtr should be a pointer");

static\_assert([std::is\_same](http://en.cppreference.com/w/cpp/types/is_same)<IntPtr, int\*>::value, "IntPtr should be a pointer to int");

static\_assert([std::is\_same](http://en.cppreference.com/w/cpp/types/is_same)<IntPtr2, IntPtr>::value, "IntPtr2 should be equal to IntPtr");

typedef [std::remove\_pointer](http://en.cppreference.com/w/cpp/types/remove_pointer)<IntPtr>::type IntAgain;

IntAgain j = i;

[std::cout](http://en.cppreference.com/w/cpp/io/cout) << "j = " << j << "**\n**";

static\_assert(![std::is\_pointer](http://en.cppreference.com/w/cpp/types/is_pointer)<IntAgain>::value, "IntAgain should not be a pointer");

static\_assert([std::is\_same](http://en.cppreference.com/w/cpp/types/is_same)<IntAgain, int>::value, "IntAgain should be equal to int");

ptr\_to\_member\_func\_cvref\_test(&S::f\_ref);

ptr\_to\_member\_func\_cvref\_test(&S::f\_const);

}

|  |  |
| --- | --- |
| [**is\_void**](https://en.cppreference.com/w/cpp/types/is_void)  (C++11) | checks if a type is void (class template) |
| [**is\_null\_pointer**](https://en.cppreference.com/w/cpp/types/is_null_pointer)  (C++14) | checks if a type is [std::nullptr\_t](http://en.cppreference.com/w/cpp/types/nullptr_t) (class template) |
| [**is\_integral**](https://en.cppreference.com/w/cpp/types/is_integral)  (C++11) | checks if a type is an integral type (class template) |
| [**is\_floating\_point**](https://en.cppreference.com/w/cpp/types/is_floating_point)  (C++11) | checks if a type is a floating-point type (class template) |
| [**is\_array**](https://en.cppreference.com/w/cpp/types/is_array)  (C++11) | checks if a type is an array type (class template) |
| [**is\_enum**](https://en.cppreference.com/w/cpp/types/is_enum)  (C++11) | checks if a type is an enumeration type (class template) |
| [**is\_union**](https://en.cppreference.com/w/cpp/types/is_union)  (C++11) | checks if a type is an union type (class template) |
| [**is\_class**](https://en.cppreference.com/w/cpp/types/is_class)  (C++11) | checks if a type is a non-union class type (class template) |
| [**is\_function**](https://en.cppreference.com/w/cpp/types/is_function)  (C++11) | checks if a type is a function type (class template) |
| [**is\_pointer**](https://en.cppreference.com/w/cpp/types/is_pointer)  (C++11) | checks if a type is a pointer type (class template) |
| [**is\_lvalue\_reference**](https://en.cppreference.com/w/cpp/types/is_lvalue_reference)  (C++11) | checks if a type is a *lvalue reference* (class template) |
| [**is\_rvalue\_reference**](https://en.cppreference.com/w/cpp/types/is_rvalue_reference)  (C++11) | checks if a type is a *rvalue reference* (class template) |
| [**is\_member\_object\_pointer**](https://en.cppreference.com/w/cpp/types/is_member_object_pointer)  (C++11) | checks if a type is a pointer to a non-static member object (class template) |
| [**is\_member\_function\_pointer**](https://en.cppreference.com/w/cpp/types/is_member_function_pointer)  (C++11) | checks if a type is a pointer to a non-static member function |