Chapter 9:
Epsilon Test

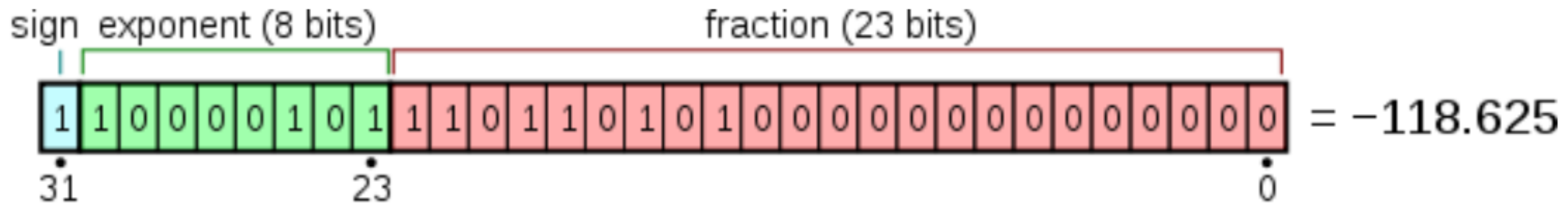
Mijin An
meeeeeejin@gmail.com

Why Floating-Point Numbers May Lose Precision

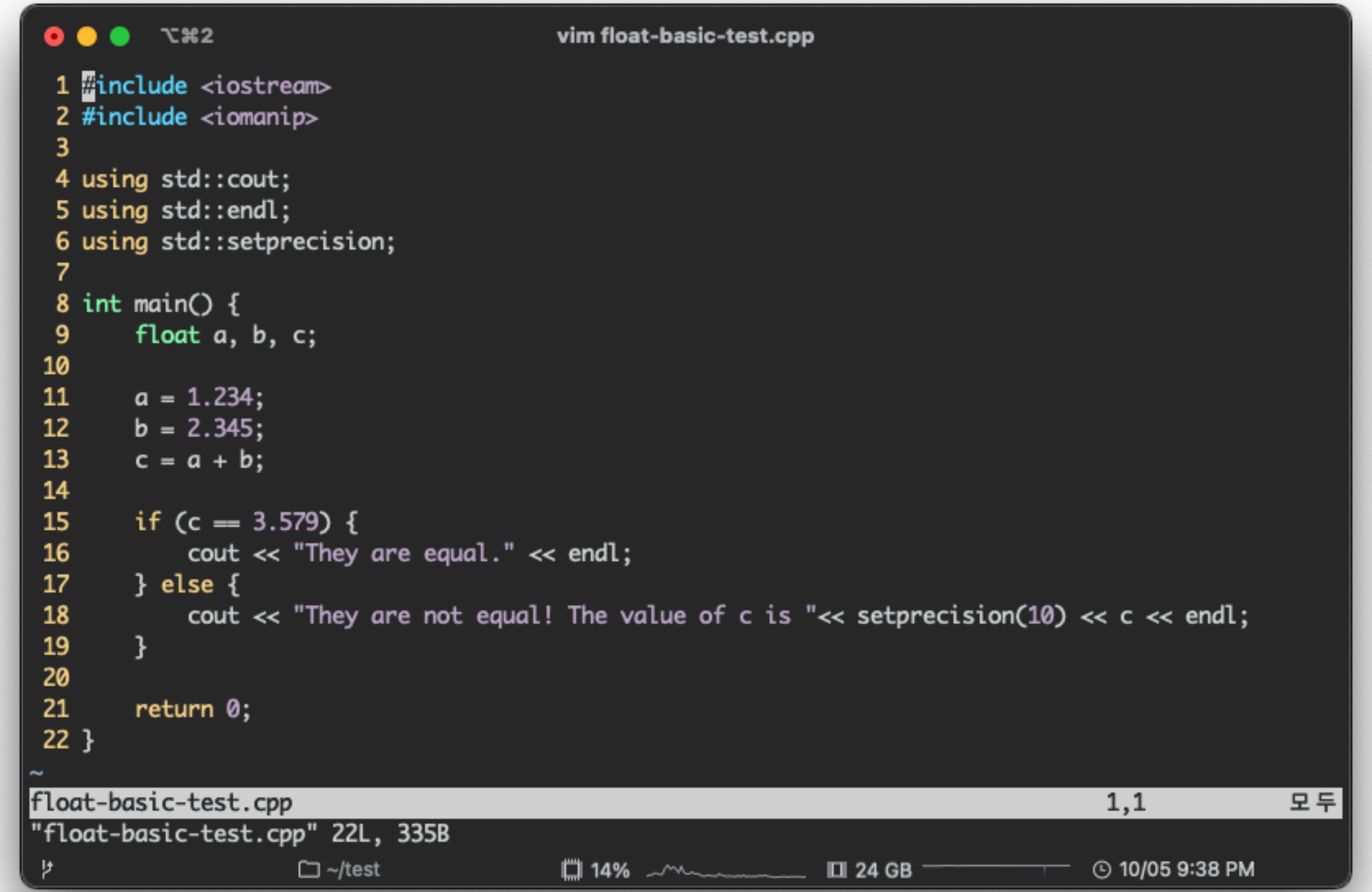
- Floating-point decimal values generally **do not have an exact binary representation**
  - A <u>side effect</u> of **how the CPU represents floating point data**
  - **Loss of precision** → Unexpected results

- To resolve the behavior:
  1. Ensure that the value is greater or less than what is needed
  2. Use a Binary Coded Decimal (BCD) library

- A technical standard for floating-point arithmetic: **IEEE 754**

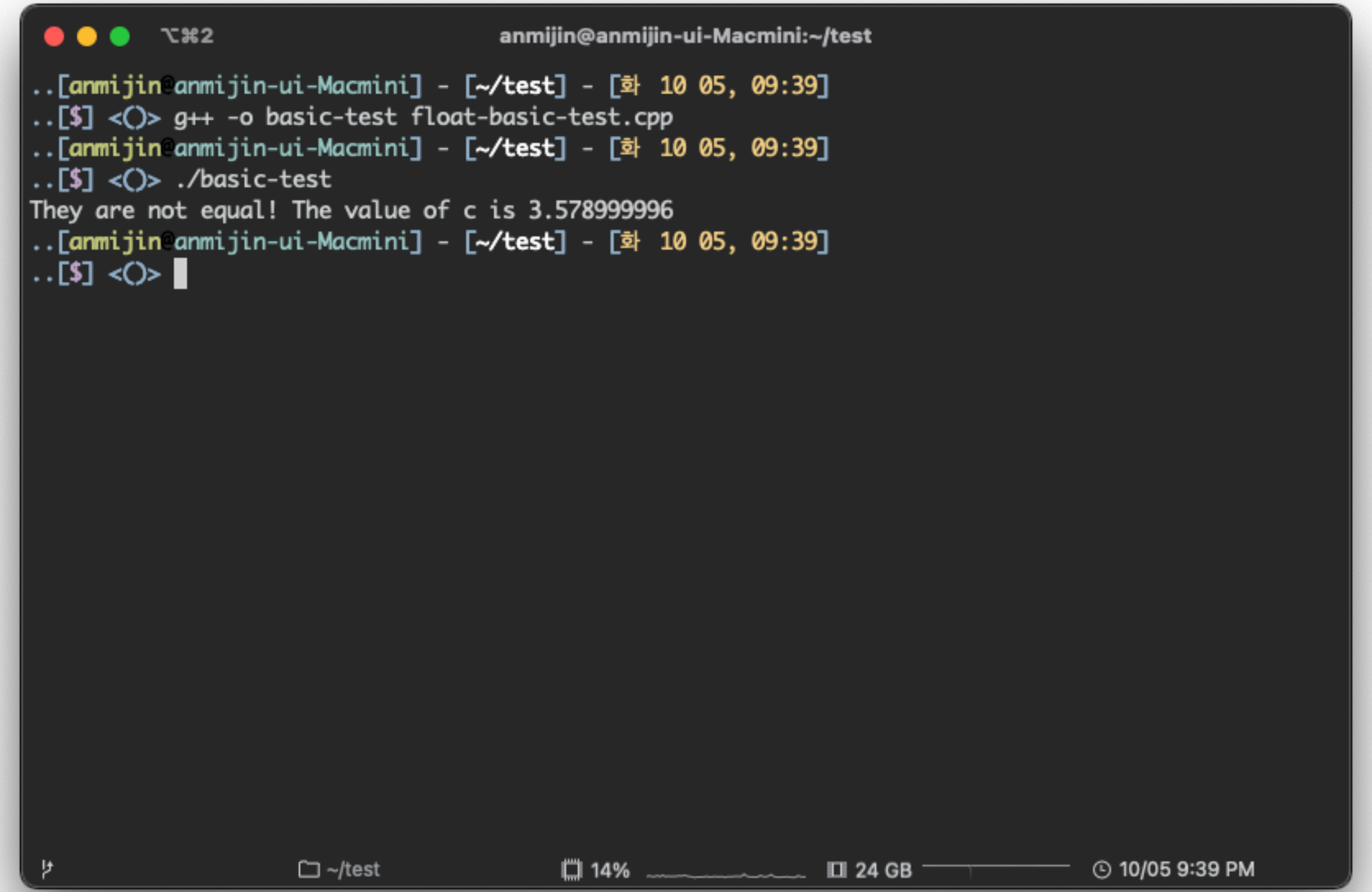


Basic Floating-Point Test



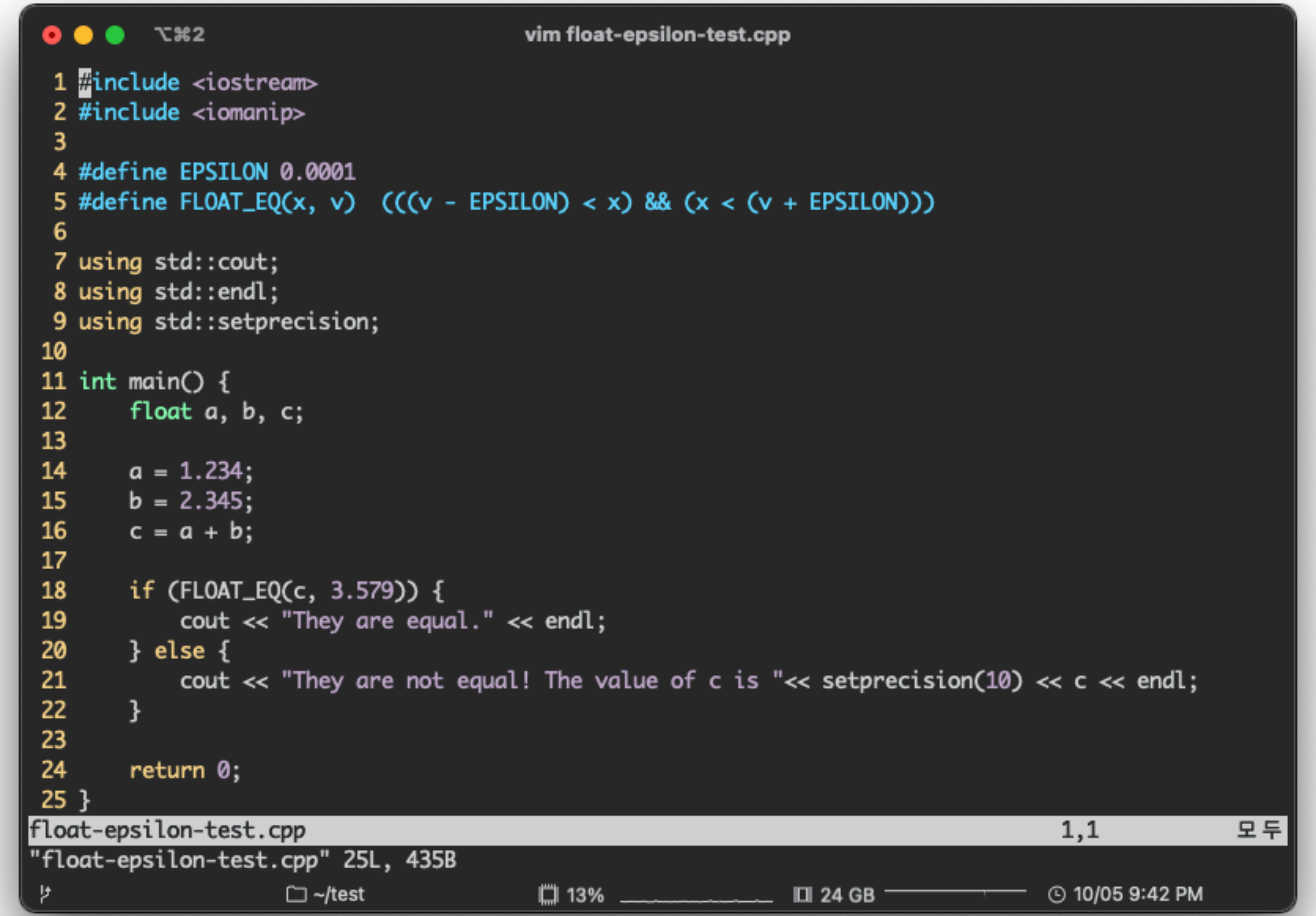
```cpp
#include <iostream>
#include <iomanip>

using std::cout;
using std::endl;
using std::setprecision;

int main() {
    float a, b, c;

    a = 1.234;
    b = 2.345;
    c = a + b;

    if (c == 3.579) {
        cout << "They are equal." << endl;
    } else {
        cout << "They are not equal! The value of c is "<< setprecision(10) << c << endl;
    }

    return 0;
}
~
float-basic-test.cpp                                                        1,1          모두
"float-basic-test.cpp" 22L, 335B
```

Basic Floating-Point Test



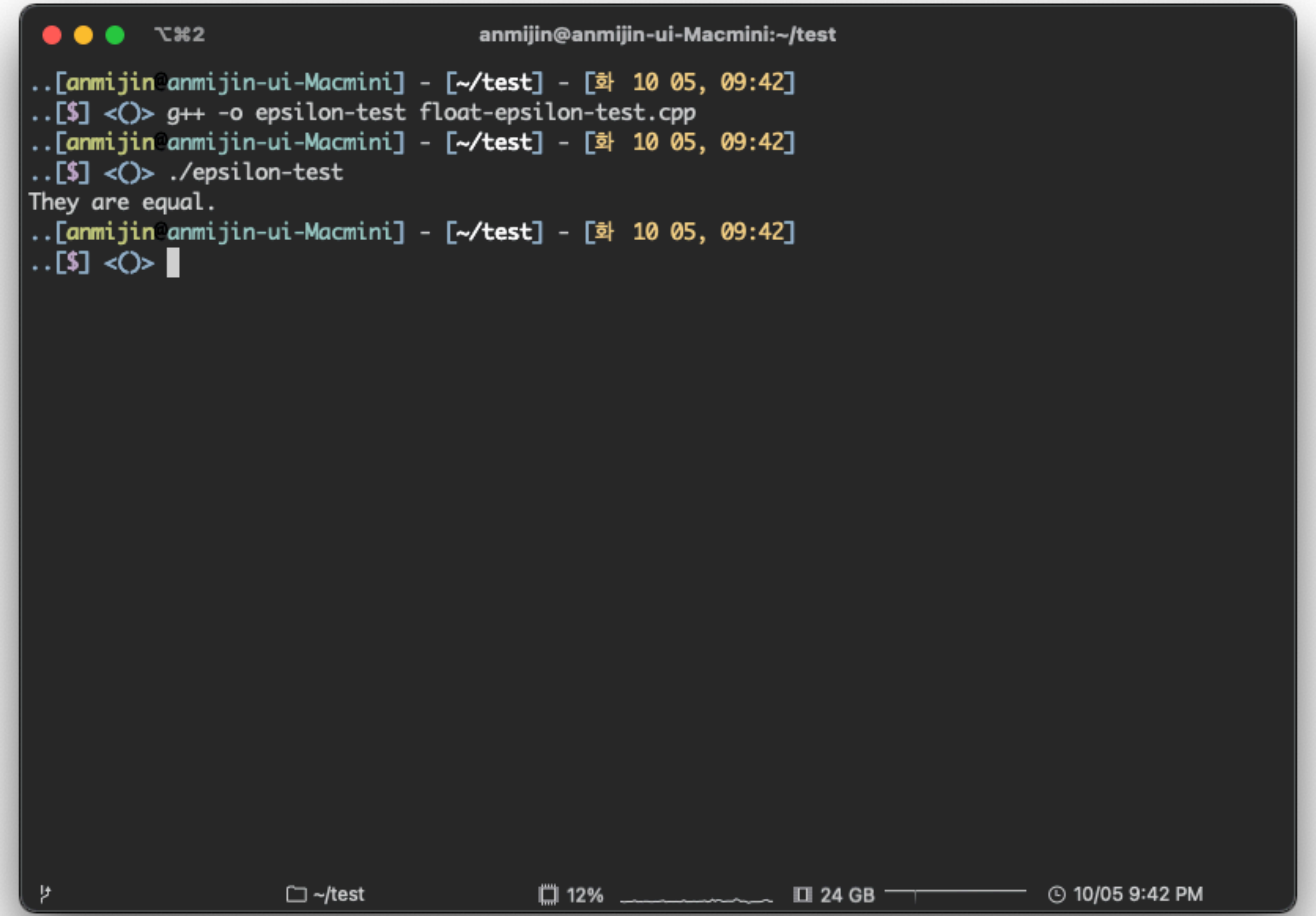
```
..[anmijin@anmijin-ui-Macmini] - [~/test] - [화 10 05, 09:39]
..[$] <()> g++ -o basic-test float-basic-test.cpp
..[anmijin@anmijin-ui-Macmini] - [~/test] - [화 10 05, 09:39]
..[$] <()> ./basic-test
They are not equal! The value of c is 3.578999996
..[anmijin@anmijin-ui-Macmini] - [~/test] - [화 10 05, 09:39]
..[$] <()>
```

Epsilon Test



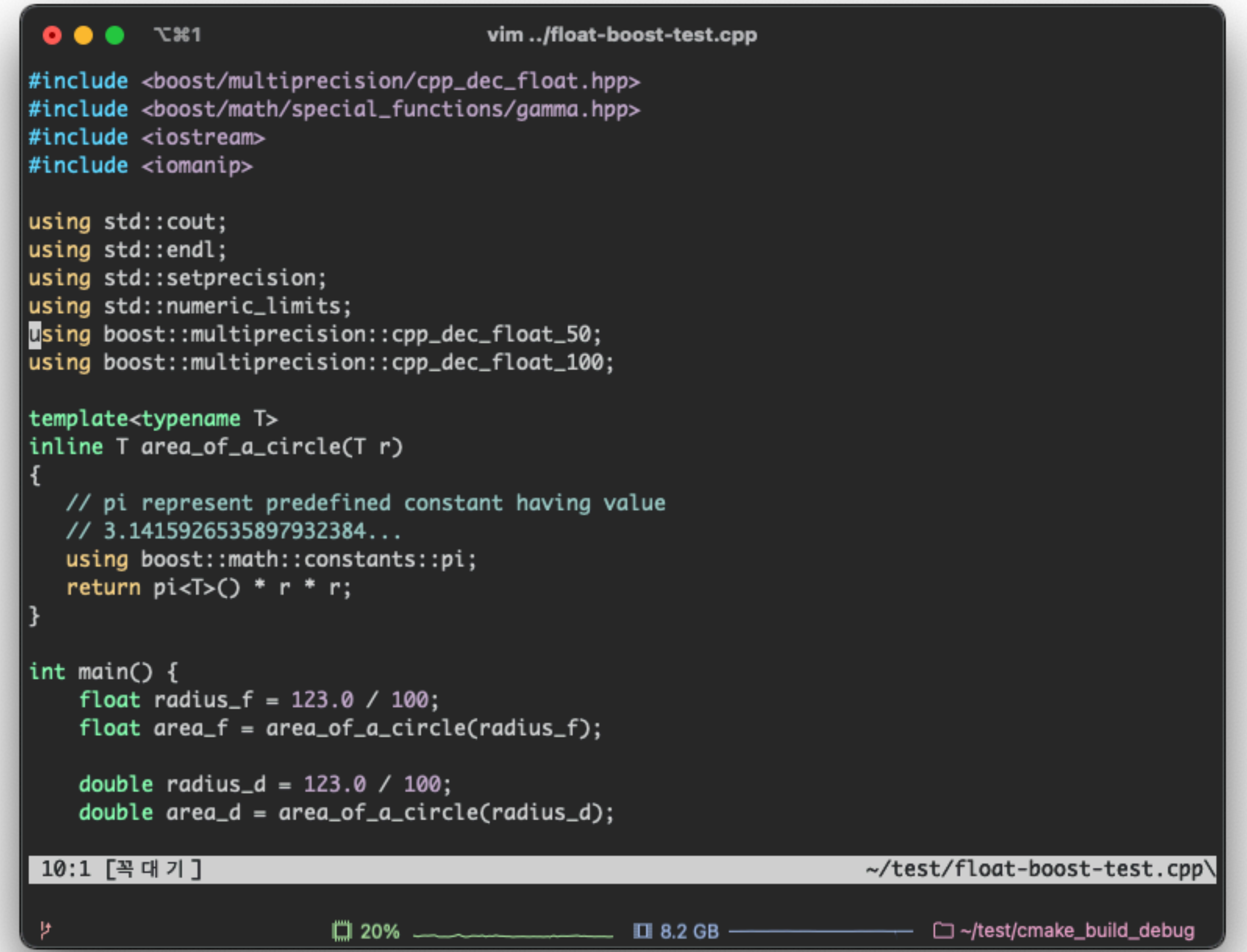
```
#include <iostream>
#include <iomanip>

#define EPSILON 0.0001
#define FLOAT_EQ(x, v)  (((v - EPSILON) < x) && (x < (v + EPSILON)))

using std::cout;
using std::endl;
using std::setprecision;

int main() {
    float a, b, c;

    a = 1.234;
    b = 2.345;
    c = a + b;

    if (FLOAT_EQ(c, 3.579)) {
        cout << "They are equal." << endl;
    } else {
        cout << "They are not equal! The value of c is "<< setprecision(10) << c << endl;
    }

    return 0;
}
float-epsilon-test.cpp                                          1,1         모두
"float-epsilon-test.cpp" 25L, 435B
```

Epsilon Test: Result



```
..[anmijin@anmijin-ui-Macmini] - [~/test] - [화 10 05, 09:42]
..[$] <()> g++ -o epsilon-test float-epsilon-test.cpp
..[anmijin@anmijin-ui-Macmini] - [~/test] - [화 10 05, 09:42]
..[$] <()> ./epsilon-test
They are equal.
..[anmijin@anmijin-ui-Macmini] - [~/test] - [화 10 05, 09:42]
..[$] <()>
```

Floating-Point Test With Boost Library

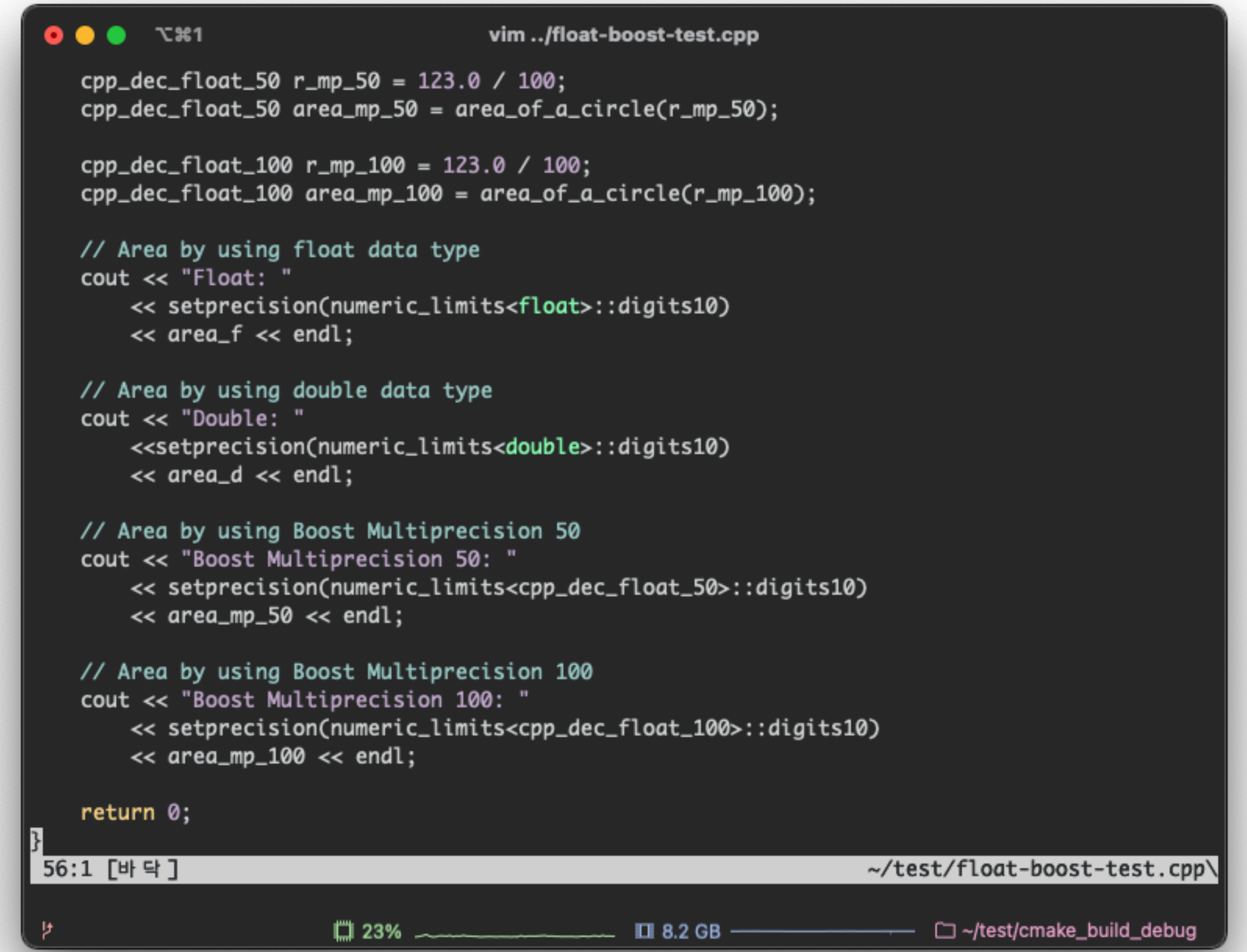


⌥⌘1 vim ../float-boost-test.cpp

```cpp
#include <boost/multiprecision/cpp_dec_float.hpp>
#include <boost/math/special_functions/gamma.hpp>
#include <iostream>
#include <iomanip>

using std::cout;
using std::endl;
using std::setprecision;
using std::numeric_limits;
using boost::multiprecision::cpp_dec_float_50;
using boost::multiprecision::cpp_dec_float_100;

template<typename T>
inline T area_of_a_circle(T r)
{
   // pi represent predefined constant having value
   // 3.1415926535897932384...
   using boost::math::constants::pi;
   return pi<T>() * r * r;
}

int main() {
    float radius_f = 123.0 / 100;
    float area_f = area_of_a_circle(radius_f);

    double radius_d = 123.0 / 100;
    double area_d = area_of_a_circle(radius_d);
```

```
10:1 [꼭대기]                                          ~/test/float-boost-test.cpp
```

20% 8.2 GB ~/test/cmake_build_debug



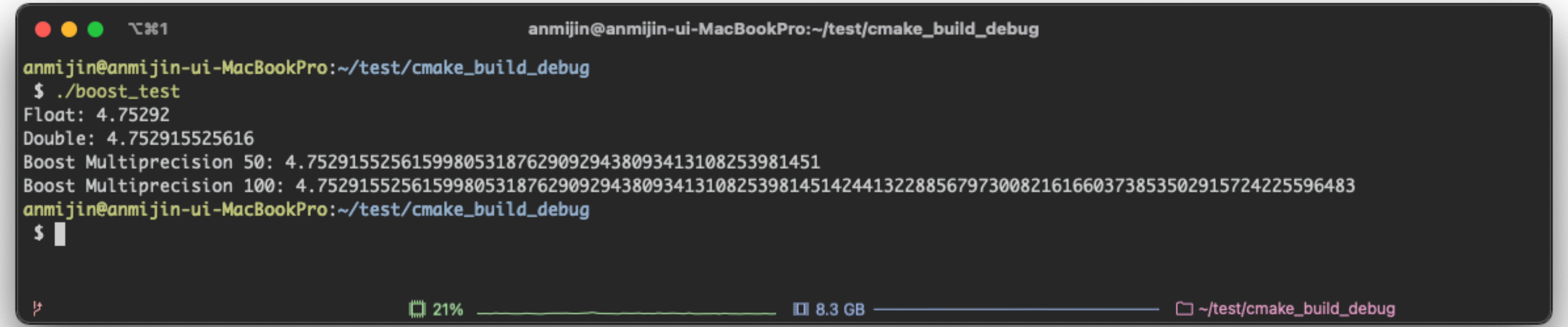
⌥⌘1 vim ../float-boost-test.cpp

```cpp
    cpp_dec_float_50 r_mp_50 = 123.0 / 100;
    cpp_dec_float_50 area_mp_50 = area_of_a_circle(r_mp_50);

    cpp_dec_float_100 r_mp_100 = 123.0 / 100;
    cpp_dec_float_100 area_mp_100 = area_of_a_circle(r_mp_100);

    // Area by using float data type
    cout << "Float: "
        << setprecision(numeric_limits<float>::digits10)
        << area_f << endl;

    // Area by using double data type
    cout << "Double: "
        <<setprecision(numeric_limits<double>::digits10)
        << area_d << endl;

    // Area by using Boost Multiprecision 50
    cout << "Boost Multiprecision 50: "
        << setprecision(numeric_limits<cpp_dec_float_50>::digits10)
        << area_mp_50 << endl;

    // Area by using Boost Multiprecision 100
    cout << "Boost Multiprecision 100: "
        << setprecision(numeric_limits<cpp_dec_float_100>::digits10)
        << area_mp_100 << endl;

    return 0;
}
```

```
56:1 [바닥]                                            ~/test/float-boost-test.cpp
```

23% 8.2 GB ~/test/cmake_build_debug

Floating-Point Test With Boost Library: Result



```
anmijin@anmijin-ui-MacBookPro:~/test/cmake_build_debug
 $ ./boost_test
Float: 4.75292
Double: 4.752915525616
Boost Multiprecision 50: 4.7529155256159980531876290929438093413108253981451
Boost Multiprecision 100: 4.7529155256159980531876290929438093413108253981451424413228856797300821616603738535029157242255596483
anmijin@anmijin-ui-MacBookPro:~/test/cmake_build_debug
 $ 
```

Reference

[1] "Why Floating-Point Numbers May Lose Precision", Microsoft, https://docs.microsoft.com/en-us/cpp/build/why-floating-point-numbers-may-lose-precision?view=msvc-160

[2] "Advanced C++ with boost library", GeeksforGeeks, https://www.geeksforgeeks.org/advanced-c-boost-library/

[3] "Floating-point Comparison", boost, https://www.boost.org/doc/libs/1_67_0/libs/math/doc/html/math_toolkit/float_comparison.html

[4] "IEEE 754", Wikipedia, https://ko.wikipedia.org/wiki/IEEE_754