# Complex numbers and templating

Victor Eijkhout, Susan Lindsey

Fall 2022

`last formatted: May 16, 2023`

# Complex numbers

# 1. Complex numbers

```
#include <complex>

complex<float> f;
f.re = 1.; f.im = 2.;
complex<double> d(1.,3.);

using std::complex_literals::i;
std::complex<double> c = 1.0 + 1i;

conj(c); exp(c);
```

# Complex Newton

# Exercise 1

Rewrite your Newton program so that it works for complex numbers:

```
complex<double> z{.5,.5};
while ( true ) {
  auto fz = f(z);
  cout << "f( " << z << " ) = " << fz << '\n';
  if (std::abs(fz)<1.e-10 ) break;
  z = z - fz/fprime(z);
}
```

You may run into the problem that you can not operate immediately between a complex number and an integer. Use `static_cast`.

# Templated functions

# 2. Templatized Newton, first attempt

You can templatize your Newton function and derivative:

```
template<typename T>
T f(T x) { return x*x - 2; };
template<typename T>
T fprime(T x) { return 2 * x; };
```

and then write

```
double x{1.};
while ( true ) {
  auto fx = f<double>(x);
  cout << "f( " << x << " ) = " << fx << '\n';
  if (std::abs(fx)<1.e-10 ) break;
  x = x - fx/fprime<double>(x);
}
```

# Exercise 2

Update your Newton program with templates. If you have it working for `double`, try using `complex<double>`. Does it work?

# Exercise 3

Use your complex Newton method to compute $\sqrt{2}$. Does it work?

# Exercise 4

Can you templatize your Newton code that used lambda expressions? Your function header would now be:

```
template<typename T>
T newton_root
   ( function< T(T) > f,
     function< T(T) > fprime,
     T init) {
```

You would for instance compute $\sqrt{2}$ as:

```
cout << "sqrt -2 = " <<
     newton_root<complex<double>>
          ( [] (complex<double> x) {
                return x*x + static_cast<complex<double>>(2); },
            [] (complex<double> x) {
              return x * static_cast<complex<double>>(2); },
            complex<double>{.1,.1}
            )
     << '\n';
```