Some features of modern C++ Writing readable code

Programming Concepts in Scientific Programming

EPFL, Master class

November 13, 2017

C++ versions

- C++98
- C++11
- C++14
- C++17

Writing **readable** code: **auto**

```
std::vector<double> vec(10);
std::vector<double>::iterator it = vec.begin();
```

It is not convenient:

```
std::vector<double> vec(10);
auto it = vec.begin();
```

Writing **readable** code: range loops

```
std::vector<double> vec(10);
auto it = vec.begin();
auto end = vec.end();

for (; it != end; ++it) {
  std::cout << *it;
}
```

Writing **readable** code: range loops

So common that it is possible to write

```
std::vector<double> vec(10);

for (double p : vec) {
  std::cout << p;
}
```

Combined with the auto

```
std::vector<double> vec(10);

for (auto p : vec) {
  std::cout << p;
}
```

Writing **readable** code: Functors

```
struct MyFunctor {
  int operator()() { return 2; }
};
```

```
int main() {
  auto f = MyFunctor();
  std::cout << f() << std::endl;
}
```

Writing **readable** code: Functors

```
struct MyFunctor {
  int operator()(double v) { return v * 2; }
};
```

```
auto f = MyFunctor();

std::vector<double> vec;
for (auto d : vec) {
  auto res = f(d);
}
```

Writing **readable** code: Functors

```
struct MyFunctor {
  int operator()(double v) { return v * 2; }
};

template <typename V, typename T> void loop(V &vec, T f) {
  for (auto d : vec) {
    auto res = f(d);
  }
}

int main() {
  auto f = MyFunctor();
  std::vector<double> vec(10);
  loop(vec, f);
}
```

Writing **readable** code: Lambda functors

http://fr.cppreference.com/w/cpp/language/lambda

```
struct MyFunctor {
  MyFunctor(double a) : a(a) {}

  int operator()(double v) { return v * 2; }
  double a;
};
```

Calling:

```
  double a = 2.;
  MyFunctor f(a);
```

Replaced with:

```
  auto f_lambda = [a](double d) { return a * d; };
```

Writing **readable** code: Lambda functors

```
template <typename V, typename T> void loop(V &vec, T f) {
  for (auto d : vec) {
    auto res = f(d);
  }
}

int main() {
  std::vector<double> vec(10);
  loop(vec, [](double d) { return 2 * d; });
}
```

Writing **readable** code: Lambda functors

```
template <typename V, typename T> void loop(V &vec, T f) {
  for (auto d : vec) {
    auto res = f(d);
  }
}

int main() {
  int a = 2;
  std::vector<double> vec(10);
  loop(vec, [a](double d) { return d * a; });
}
```

Writing **readable** code: For each

```cpp
#include <algorithm>
#include <iostream>
#include <numeric>
#include <vector>

int main() {
  std::vector<double> v(10);
  std::iota(v.begin(), v.end(), 0);
  std::for_each(v.begin(), v.end(), [](double &x) { x = x * x; });
  std::for_each(v.begin(), v.end(),
                [](double x) { std::cout << x << std::endl; });
}
```