Better Lambdas

Giovanni Dicanio
AUTHOR, SOFTWARE ENGINEER

https://blogs.msmvps.com/gdicanio

Overview

Quick intro/refresher on lambdas

Generic lambdas

Init-capture

No name

```
[](int a, double b) {
  // Do something …
}
```

Inline implementation

Lambda: Unnamed Function

```
vector<string> names{
  "Mike", "John", "Beth", "Austin",
  "Bob", "Cindy", "Elizabeth", "Connie"
};
```

Example: Sorting a String Vector

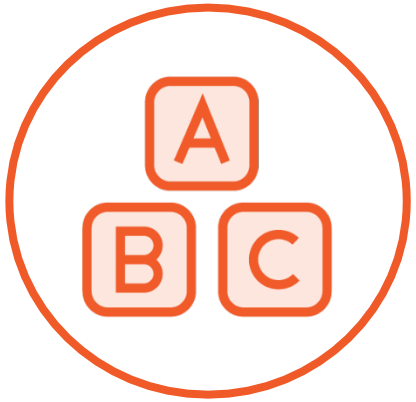
```
vector<string> names{
  "Mike", "John", "Beth", "Austin",
  "Bob", "Cindy", "Elizabeth", "Connie"
};

sort(begin(names), end(names));
```

Iterators and Sorting

bit.ly/VecSort

Example: Sorting a String Vector



```
vector<string> names{
  "Mike", "John", "Beth", "Austin",
  "Bob", "Cindy", "Elizabeth", "Connie"
};

sort(begin(names), end(names));
```

CUSTOM RULE

Example: Sorting a String Vector

Custom Sorting

Invoked by std::sort to figure out which string comes first

```
bool compare(string const& a, string const& b) {
  // Return true if a is ordered before b
}
```

Comparison Function

Compares two elements using a custom rule

Observing parameters

bit.ly/ObParam

CONST & : OBSERVING PARAMETERS

```
bool compare(string const& a, string const& b) {
  // Return true if a is ordered before b
}
```

Comparison Function

Compares two elements using a custom rule

```
bool compare(string const& a, string const& b) {
  // Return true if a is shorter than b
}
```

Example: Sorting Strings by Length

Shorter strings first

```
bool compare(string const& a, string const& b) {
        a.length()   b.length()
}
```

Example: Sorting Strings by Length

Shorter strings first

```
bool compare(string const& a, string const& b) {
        a.length() < b.length()
}
```

Example: Sorting Strings by Length

Shorter strings first

```
bool compare(string const& a, string const& b) {
  return a.length() < b.length();
}
```

Example: Sorting Strings by Length

Shorter strings first

PLUG IN std::sort

```
sort(begin(names), end(names), compare);
```

```
bool compare(string const& a, string const& b) {
  return a.length() < b.length();
}
```

Example: Sorting Strings by Length

Shorter strings first

```
sort(begin(names), end(names),  λ  );
```

Custom Sorting with Lambdas

```
sort(begin(names), end(names),
  [](string const& a, string const& b) {
    return a.length() < b.length();
  }
);
```

Custom Sorting with Lambdas

Shorter strings first

```
sort(begin(names), end(names),
  [](string const& a, string const& b) {
    return a.length() < b.length();
  }
);
```

LAMBDA INTRODUCER

Custom Sorting with Lambdas

Shorter strings first

```
sort(begin(names), end(names),
  [](string const& a, string const& b) {
    return a.length() < b.length();
  }
);
```

PARAMETER LIST

Custom Sorting with Lambdas

Shorter strings first

```
sort(begin(names), end(names),
  [](string const& a, string const& b) {
    return a.length() < b.length();
  }
);
```

BODY

Custom Sorting with Lambdas

Shorter strings first

```
sort(begin(names), end(names),
  [](string const& a, string const& b) {
    return a.length() < b.length();
  }
);
```

RETURN TYPE DEDUCED bool

Custom Sorting with Lambdas

Shorter strings first

LAMBDA CODE IN-PLACE

```
sort(begin(names), end(names),
  [](string const& a, string const& b) {
    return a.length() < b.length();
  }
);
```

Custom Sorting with Lambdas

Shorter strings first

```
sort(begin(names), end(names),
  [](string const& a, string const& b) {
    return a.length() < b.length();
  }
);
```

STRING VECTOR

Lambda for Custom Sorting

```
sort(begin(names), end(names),
  [](string const& a, string const& b) {
    return a.length() < b.length();
  }
);
```

Generic Lambdas

What if...

vector<unique_ptr<MyCoolClass>>

??

```
sort(begin(names), end(names),
  [](auto const& a, auto const& b) {
    return a.length() < b.length();
  }
);
```

Generic Lambdas

Use *auto*!

Demo

Generic Lambdas

CAPTURE LIST

```
[](auto const& a, auto const& b) {
  return a.length() < b.length();
}
```

Anatomy of a Lambda

PARAMETER LIST

```
[](auto const& a, auto const& b) {
  return a.length() < b.length();
}
```

Anatomy of a Lambda

```
[](auto const& a, auto const& b) {
  return a.length() < b.length();
}
```

BODY

Anatomy of a Lambda

```
int x{…};
double y{…};
…
[x, y](auto a, auto b) {
  // Do something…
}
```

CAPTURE LIST
Variables declared outside the lambda, *imported* inside it

Anatomy of a Lambda

Capture list

```
int x{…};
double y{…};
…
[x, y](auto a, auto b) {
  // Do something…
}
```

CAPTURE LIST

Variables declared outside the lambda, *imported* inside it

***x* and *y* captured and available here**

Anatomy of a Lambda

Capture list

```
int x{…};
double y{…};
…
[x, y](auto a, auto b) {
  // Do something…
}
```

CAPTURE LIST

x and *y* captured *by value*

Copies

of *x* and *y*

Anatomy of a Lambda

Capture list

```
int x{…};
double y{…};
…
[x, y](auto a, auto b) {
  // Do something…
}
```

CAPTURE LIST

x and *y* captured *by value*

Basic Rules for Parameter Passing in C++

bit.ly/CppParamRules

Anatomy of a Lambda

Capture list

```
int x{…};
double y{…};
…
[x, &y](auto a, auto b) {
  // Do something…
}
```

Anatomy of a Lambda

Capture list

```
int x{…};
double y{…};
…
[x, &y](auto a, auto b) {
  // Do something…
}
```

CAPTURE LIST

x by value

y by *reference*

Anatomy of a Lambda

Capture list

```
int x{…};
double y{…};
…
[x, &y](auto a, auto b) {
  // Do something…
}
```

CAPTURE LIST
x by value
y by *reference*

Access to the *original y*
(*not* a copy)

Anatomy of a Lambda

Capture list

```
int x{…};
double y{…};
…
[x, y, value = 64](auto a, auto b) {
  // Do something…
}
```

Init-capture

```
unique_ptr<X> p1 /* initialized to something… */;
unique_ptr<X> p2;
```

Movable Non-copyable

```
unique_ptr<X> p1 /* initialized to something… */;
unique_ptr<X> p2;

p2 = p1;
```

Movable Non-copyable

```
unique_ptr<X> p1 /* initialized to something… */;
unique_ptr<X> p2;

p2 = p1;
```

Movable Non-copyable

```
unique_ptr<X> p1 /* initialized to something… */;
unique_ptr<X> p2;

p2 = std::move(p1);
```

Movable Non-copyable

```
unique_ptr<X> p1 /* initialized to something… */;
unique_ptr<X> p2;

p2 = std::move(p1);
```

Movable Non-copyable

Transfer ownership

```
unique_ptr<X> p{…};

[ u{move(p)} ]( /* parameters */ ) {
  // Do something…
}
```

Init-capture

Capture by *move*

```
unique_ptr<X> p{…};

[ u{move(p)} ]( /* parameters */ ) {
  // Do something…
}
```

Init-capture

Capture by move

```
class Image {
…
  // Image pixels (R,G,B)
  unique_ptr<Pixel> m_data;
};
```

Image Class

Movable but non-copyable

```
class Image {
…
  // Image pixels (R,G,B)
  unique_ptr<Pixel> m_data;
};
```

Image Class

Movable but non-copyable

Init-capture with std::*move*

Demo

Init-capture

Summary

Quick lambda intro/refresher

Generic lambdas (auto)

Init-capture