constexpr Lambda Functions

Lambda functions are now compatible with constant expressions.

Lambda expressions were introduced in C++11, and since that moment they've become an essential part of modern C++. Another significant feature of C++11 is the `constexpr` specifier, which is used to express that a function or value can be computed at compile-time.

In C++17, the two elements are allowed to exist together, so your lambda can be invoked in a constant expression context.

In C++11/14 the following code doesn't compile, but works with C++17:

C++ C++17

```
#include <iostream>
using namespace std;

int main () {
    constexpr auto SquareLambda = [] (int n) { return n*n; };
    static_assert(SquareLambda(3) == 9, "");
}
```

Since C++17 lambda expressions (their call operator `operator()`) that follow the rules of standard `constexpr` functions are implicitly declared as `constexpr`.

What are the limitations of `constexpr` functions?

Here's a summary (from 10.1.5 The constexpr specifier [dcl.constexpr]):

- they cannot be virtual
- their return type shall be a literal type
- their parameter types shall be a literal type

- their function bodies cannot contain: `asm` definition, a `goto` statement, try-block, or a variable that is a non-literal type or static or thread storage duration

In practice, in C++17, if you want your function or lambda to be executed at compile-time, then the body of this function shouldn't invoke any code that is not `constexpr`. For example, you cannot allocate memory dynamically or throw exceptions.

`constexpr` lambda expressions are also covered in the Other Changes Chapter and in a free ebook: C++ Lambda Story.

> *Extra Info:* The change was proposed in: P0170.

Head over to the next lesson to look at how to capture [*this] in Lambda Expressions.