Compile-time only constexpr-functions

Document Number:	**P0000 R0**
Date:	2017-12-02

Reply-to:	Andreas Fertig ⟨ isocpp@andreasfertig.info ⟩

Audience:	SG14, LWG

Contents

1 Introduction

The purpose of this document is to outline a new attribute `[[compile_time_only]]` to force `constexpr`-functions to only be evaluated at compile time only. This gives stronger guarantees on some cases.

2 Motivation

With the introduction of `constexpr` we have the potential to ask the compiler to evaluate a certain function at compile-time. The plus is, that this function does not cost run-time nor does it effect the size of the binary. However, `constexpr`-functions *can* be evaluated at compile-time only, if all requirements are satisfied. If not, they are handled like normal functions. Thereby increasing the size of the binary and execution is at run-time. Thus we get a run-time overhead which we neither want nor may be aware of them.

In embedded systems development it is sometimes vital to ensure that a desired function is *only* executed at compile-time. For instance when a supposedly compile-time only hash function is invoked. This code and the execute-time should only happen at compile-time.

There are also scenarios in which some tries to obfuscate data and has a function taking something like an obfuscation-key which should never be seen outside the compile process.

Solutions to express that a certain function is meant for compile-time only include pre- or postfixing this function. For instance a function named `hash` is then called `constexpr_hash`. This is bad for readability. Furthermore there is no tool or compiler support.

This design of `constexpr` is, that we do not have to implement all functions twice. Once for `constexpr` and once more in the non-`constexpr` case. This paper proposes the adoption of a new attribute: `[[compile_time_only]]`. This new attribute can be applied to a `constexpr`-function declaration. A function which such an attribute *and* `constexpr` triggers a compiler error, if it is not evaluated at compile-time.

3 Example

Consider the function `func()`. Depending on the parameter `x` it is evaluated at compile time or not.

```
1  constexpr int func(int x)
2  {
3    return 2 * x;
4  }
5
6  int main()
7  {
8    int rt = 2;
9
10   constexpr int a = func(1);    // (1) constexpr
11   constexpr int b = func(rt);   // (2) error
12             int c = func(1);    // (3) constexpr
13             int d = func(rt);   // (4) run-time
14 }
```

In case (1) `func` is `constexpr` and evaluated at compile-time. Case (2) is also ok, here the compiler announces an error that a `constexpr` variable needs a constant initializer. In case (3) the function `func` is evaluated at compile-time, while the variable `c` itself is still non-constexpr. Case (4) is which this proposal aims to improve.

Here the function `func` is marked `constepxr` but invoked in run-time context. This is not all the time obvious. In case when performance and code footprint matters it may not the intention to call this function in any non-`constexpr` context.

4 Alternatives

1. Always possible: do nothing.

2. Provide a type-trait `is_constexpr_invocation` which can then be used in a `static_assert`. This is additional coding effort for each `constexpr` function. It also can be written differently every time, which makes it harder to read and understand. This is without the discussion how such a type-trait could be implemented.

3. If [1] makes it, the proposed `constexpr` operator can be used in a `static_assert`. With that a function can be force to be compile-time only.

```
1 constexpr int func(int x)
2 {
3     static_assert(constexpr(), "Invoked in non-constexpr context");
4     return 2 * x;
5 }
```

However, it still has the disadvantage to not see it on the signature of the function and it causes typing repetition.

4. Provide a `constexpr`-overload-way. For example the parameters of a function can be marked `constexpr`. This can be seen as an overload to the variant without `constexpr`-marked parameters. A problem here may be, that with multiple parameters all of them must be `constexpr`. Otherwise there is a chance that the function is executed at run-time.

5 Acknowledgements

Thanks to Peter Sommerlad for suggesting the overload idea.

Bibliography

[1] Daveed Vandevoorde: "*The constexpr Operator*", P0595R0, 2017-02-02.
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/p0595r0.html