

## Section 1.2 C++14

## Variable Templates

```

#include <iostream> // std::cout

template <int N>
const int fib = fib<N - 1> + fib<N - 2>; // OK, compile-time const

template <> const int fib<2> = 1; // OK, compile-time const
template <> const int fib<1> = 1; // OK, compile-time const

int main()
{
    std::cout << fib<4> << '\n'; // guaranteed to print out 3
    std::cout << fib<5> << '\n'; // guaranteed to print out 5
    std::cout << fib<6> << '\n'; // guaranteed to print out 8

    return 0;
}

```

Note that replacing each of the three **const** keywords with **constexpr** in the example above also achieves the desired goal and would also be applicable to nonintegral constants.

**Annoyances****Variable templates do not support template template parameters**

Although a class, variable, or function template can accept a (class) **template template parameter**, no equivalent construct is available to accept *variable* or *function* templates<sup>3</sup>:

```

template <typename T> T vt(5);

template <template <typename> class>
struct S { };

S<vt> s1; // Error

```

Providing a wrapper **struct** around a **variable template** might therefore be necessary in case the **variable template** needs to be passed to an interface accepting a **template template parameter**:

```

template <typename T>
struct Vt { static constexpr T value = vt<T>; };

S<Vt> s2; // OK

```

---

<sup>3</sup>A method to increase consistency between variable templates and class templates when used as template template parameters has been proposed for C++23; see **pusz20a**.