```
              int yB0 = squareB(1 << 15);  // OK
        const int yB1 = squareB(1 << 15);  // OK
constexpr       int yB2 = squareB(1 << 15);  // OK

              int zC0 = squareB(1 << 16);  // Bug, zC0 is likely 0.
        const int zC1 = squareB(1 << 16);  // Bug, zC1  "    "   "
constexpr       int zC2 = squareB(1 << 16);  // Error, int overflow detected!
```

The compiler must evaluate the `squareB` function in the example above at compile time when it is used to initialize a **constexpr** variable and, consequently, report any UB that would arise during such evaluation as an error. Such is not the case for initialization of non**constexpr** variables even if they are **const**. In such cases, the initialization must happen **as if** it were evaluated at run time and the compiler might choose to do so. Therefore, when initializing a non**constexpr** variable of an integral type, the presence of UB will determine whether the variable will be a compile-time constant but will not lead to a compilation error.

Internal linkage

When a variable at file or namespace scope is either **const** or **constexpr** and nothing explicitly gives it **external linkage** (e.g., by being marked **extern**), it will have **internal linkage**, meaning that each **translation unit** will have its own copy of the variable.[3]

Oftentimes, only the values of such variables are relevant: Their initializers are seen, they are used at compile time, and there is no effect if different translation units use different objects having the same name and a different value. After compile-time evaluation is completed, the variables themselves will no longer be needed, and no actual address will be allocated for them at run time. Only in cases where the address of the variable is used will the effects of internal linkage be observable (and can lead to ODR violations).

Notably, **static** data members have external linkage except when inside an **unnamed namespace**. Therefore, if they are used in a way that requires they have an address allocated at run time, then a **definition** needs to be provided for them outside of their class irrespective of whether they are **constexpr**; see *Annoyances — **static** data members require external definitions* on page 314.

Use Cases

Alternative to enumerated compile-time integral constants

It is not uncommon to want to express specific integral constants at compile time — e.g., for precomputed operands to be used in algorithms, mathematical constants, configuration variables, or any number of other reasons. A naive, brute-force approach might be to hard-code the constants where they are used:

[3] In C++17, all **constexpr** ~~variables~~ are instead automatically **inline** as well, guaranteeing that there is only one instance in a program.