Die vielen Varianten von Konstantheit in modernem C++

Rainer Grimm
Schulung und Mentoring

Varianten von Konstantheit

Varianten

- `const`
- `const_cast`
- `constexpr`
- `consteval`
- `constinit`
- `is_constant_evaluated`

Unterschiede

- Funktion
- Variablen

Varianten von Konstantheit

Varianten

- `const`
- `const_cast`
- `constexpr`
- `consteval`
- `constinit`
- `is_constant_evaluated`

Unterschiede

- Funktion
- Variablen

`const`

Const Correctness: Verwende das Schlüsselwort `const`, um zu verhindern, dass Objekte mutiert werden.

➡ `const` ist ein Qualitätsmerkmal für das Programm.

Ein konstantes Objekt

- muss initialisiert werden.
- kann nicht verändert werden.
- kann nicht Opfer eines Data Races werden.
- kann nur `const`-Mitgliedsfunktionen aufrufen.

const

Per Default sollen konstante Zeiger und Referenzen übergeben werden

```
void getCString(const char* cStr);
void getCppString(const std::string& cppStr);
```

Ausnahme: Nichtkonstante Zeiger und Referenzen

```
void modifyCString(char* cStr);
void modifyCppString(std::string& cppStr);
```

➡ Ein- und Ausgabeparameter

Varianten von Konstantheit

Varianten

- `const`
- `const_cast`
- `constexpr`
- `consteval`
- `constinit`
- `is_constant_evaluated`

Unterschiede

- Funktion
- Variablen

const_cast

`const_cast` erlaubt es, `const` oder `volatile` zu einer Variable hinzufügen oder zu entfernen.

Verwende keinen C-Cast (`int i = (int) myValue;`), da dieser eine Reihe von Casts anwendet:

`static_cast` → `const_cast` → `reinterpret_cast`

Das Ändern eines als `const` deklarierten Objekts durch Entfernen seiner Konstantheit ist undefiniertes Verhalten.

modifyingConst.cpp

Varianten von Konstantheit

Varianten

- `const`
- `const_cast`
- `constexpr`
- `consteval`
- `constinit`
- `is_constant_evaluated`

Unterschiede

- Funktion
- Variablen

constexpr

Konstante Ausdrücke

- können zur Compilezeit ausgewertet werden.
- geben dem Compiler einen tiefen Einblick.
- sind implizit thread-sicher.

- Variablen

```
constexpr double myDouble = 5.2;
const int myInt = 5;
```

  - sind implizit konstant.
  - sind implizit thread-sicher. ➡ Ein Data Race erfordert einen gemeinsamen veränderbaren Zustand.

`const/constexpr`-Variablen machen es einfach ein concurrent Programm zu analysieren.

constexpr

- Funktionen

```
constexpr int gcd(int a, int b) {
    while (b != 0){
        auto t = b;
        b = a % b;
        a = t;
    }
    return a;
}
```

  - muss jede Abhängigkeit zur Compilezeit auflösen können.
  - können Variablen haben, die initialisiert werden müssen.
  - kann keine `statischen` und `thread_local`-Variablen haben.

  - haben das Potenzial, zur Compilezeit ausgeführt zu werden.
  - sind rein.

constexpr

- Reine Funktionen (Mathematische Funktionen)
  - geben für die gleichen Argumente das gleiche Ergebnis zurück.
  - besitzen keine Seiteneffekte.
  - ändern nicht den Status des Programms.

- Vorteile
  - Einfach zu testen und zu refaktorieren
  - Lassen sich in beliebiger Reihenfolge ausführen
  - Können automatisch parallelisiert werden
  - Erlauben alte Ergebnisse zwischenzuspeichern

constexpr

- Benutzerdefinierte Typen

```
struct MyDouble {
    double myVal;
    constexpr MyDouble(double v): myVal(v){}
    constexpr double getVal(){return myVal;}
};
```

  - müssen mindestens einen `constexpr`-Konstruktor besitzen.
  - können `constexpr` und nicht-`constexpr` Mitgliedsfunktionen haben.

constexpr

C++20 unterstützt die `constexpr`-Container `std::vector` und `std::string`.

- Die mehr als 100 Algorithmen der STL werden in C++20 als `constexpr` deklariert.

Wenn möglich, sollen benutzerdefinierte Datentypen oder Funktionen als `constexpr` erklärt werden.

constexprVector.cpp

Varianten von Konstantheit

Varianten

- `const`
- `const_cast`
- `constexpr`
- `consteval`
- `constinit`
- `is_constant_evaluated`

Unterschiede

- Funktion
- Variablen

consteval

`consteval` erzeugt eine *immediate* Funktion.

- Jeder Aufruf einer immediate Funktion erzeugt einen konstanten Ausdruck, der zur Compilezeit ausgeführt wird.

`consteval`

- besitzt die gleichen Anforderungen wie eine `constexpr`-Funktion.

```
consteval int sqr(int n) {
    return n * n;
}
constexpr int r = sqr(100); // OK

int x = 100;
int r2 = sqr(x);            // Error
```

Varianten von Konstantheit

Varianten

- `const`
- `const_cast`
- `constexpr`
- `consteval`
- `constinit`
- `is_constant_evaluated`

Unterschiede

- Funktion
- Variablen

`constinit`

`constinit` garantiert, dass eine Variable mit statischer Speicherdauer zur Compilezeit initialisiert wird.

Diese Variable ist weiterhin veränderbar.

- Statische Speicherdauer
  - Globale Objekte oder Objekte, die mit `static` oder `extern` deklariert sind, haben eine statische Speicherdauer.
  - Objekte mit statischer Speicherdauer werden beim Programmstart allokiert und am Programmende deallokiert.

constinit

Static Initialization Order Fiasco: Die Reihenfolge der Initialisierung von statischen Variablen zwischen Übersetzungseinheiten ist nicht festgelegt.

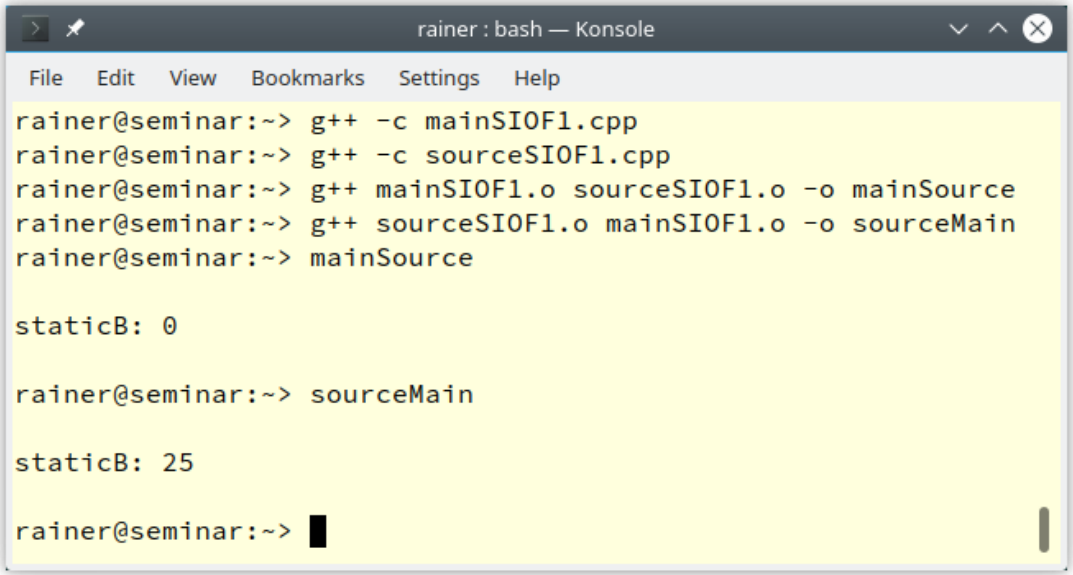
- Die Initialisierung nicht-konstanter statischer Variablen erfolgt in zwei Schritten.
  - Compilezeit: Die Variablen werden Null-initialisiert.
  - Laufzeit: Die Null-initialisierten Variablen werden dynamisch initialisiert.

➡ `constinit` löst das Static Initialization Order Fiasco.

constinit

```
// sourceSIOF1.cpp
int square(int n) {
  return n * n;
}
auto staticA = square(5);
```

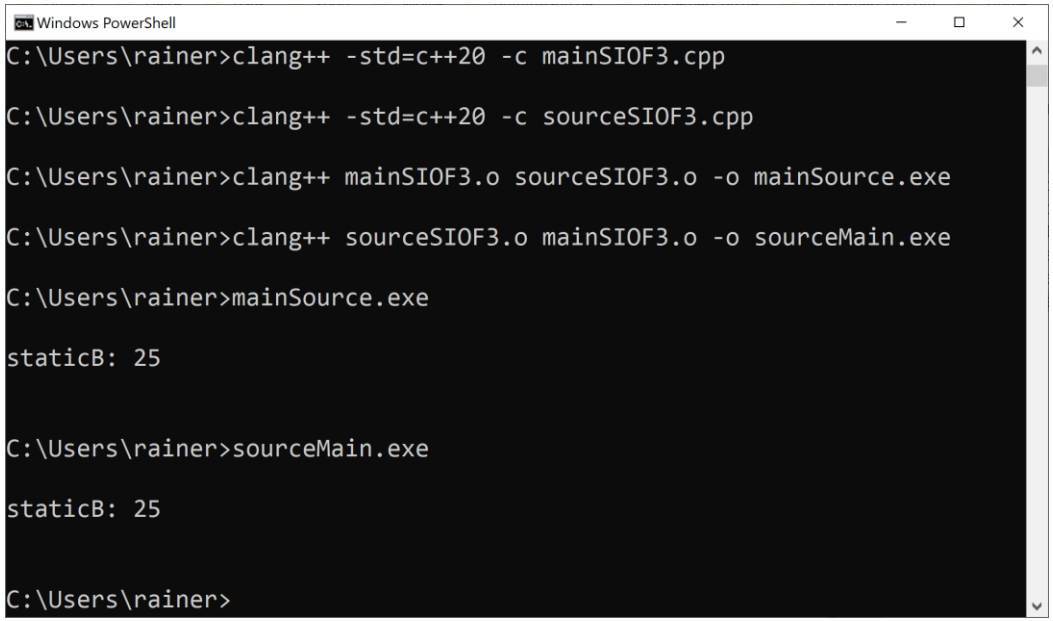
```
// mainSOIF1.cpp
#include <iostream>

extern int staticA;
auto staticB = staticA;

int main() {
  std::cout << "staticB: " << staticB;
}
```



constinit

```
// sourceSIOF3.cpp
constexpr int square(int n) {
  return n * n;
}

constinit auto staticA = square(5);
```

```
// mainSOIF3.cpp
#include <iostream>

extern constinit int staticA;
auto staticB = staticA;

int main() {
    std::cout << "staticB: " << staticB;
}
```



Varianten von Konstantheit

Varianten

- `const`
- `const_cast`
- `constexpr`
- `consteval`
- `constinit`
- `is_constant_evaluated`

Unterschiede

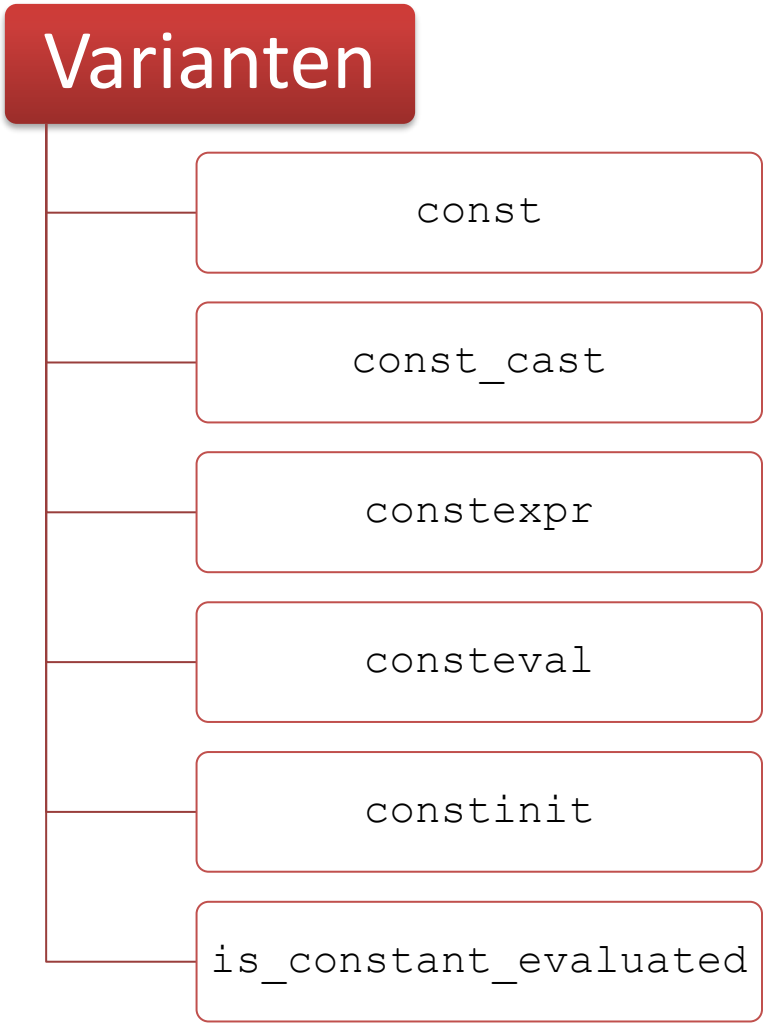
- Funktion
- Variablen

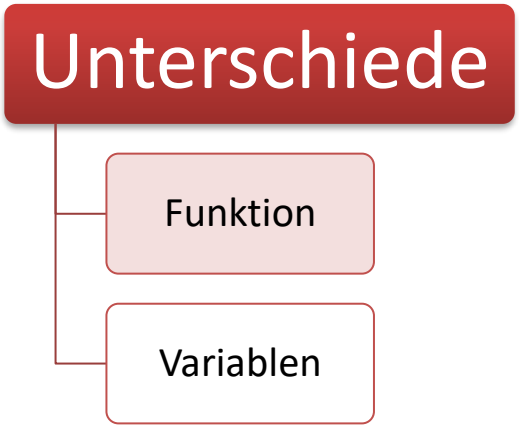
std::is_constant_evaluated

`std::is_constant_evaluated` bestimmt, ob die Funktion zur Compilezeit oder Laufzeit ausgeführt wird.

```
constexpr double power(double b, int x) {
    if (std::is_constant_evaluated() && !(b == 0.0 && x < 0)) {
        if (x == 0) return 1.0;
        double r = 1,0, p = x > 0 ? b : 1,0 / b;
        auto u = unsigned(x > 0 ? x : -x);
        while (u != 0) {
            if (u & 1) r *= p;
            u /= 2;
            p *= p;
        }
        return r;
    }
    else return std::pow(b, double(x)); // not declared constexpr
}   // https://en.cppreference.com/w/cpp/types/is_constant_evaluated
```

Varianten von Konstantheit



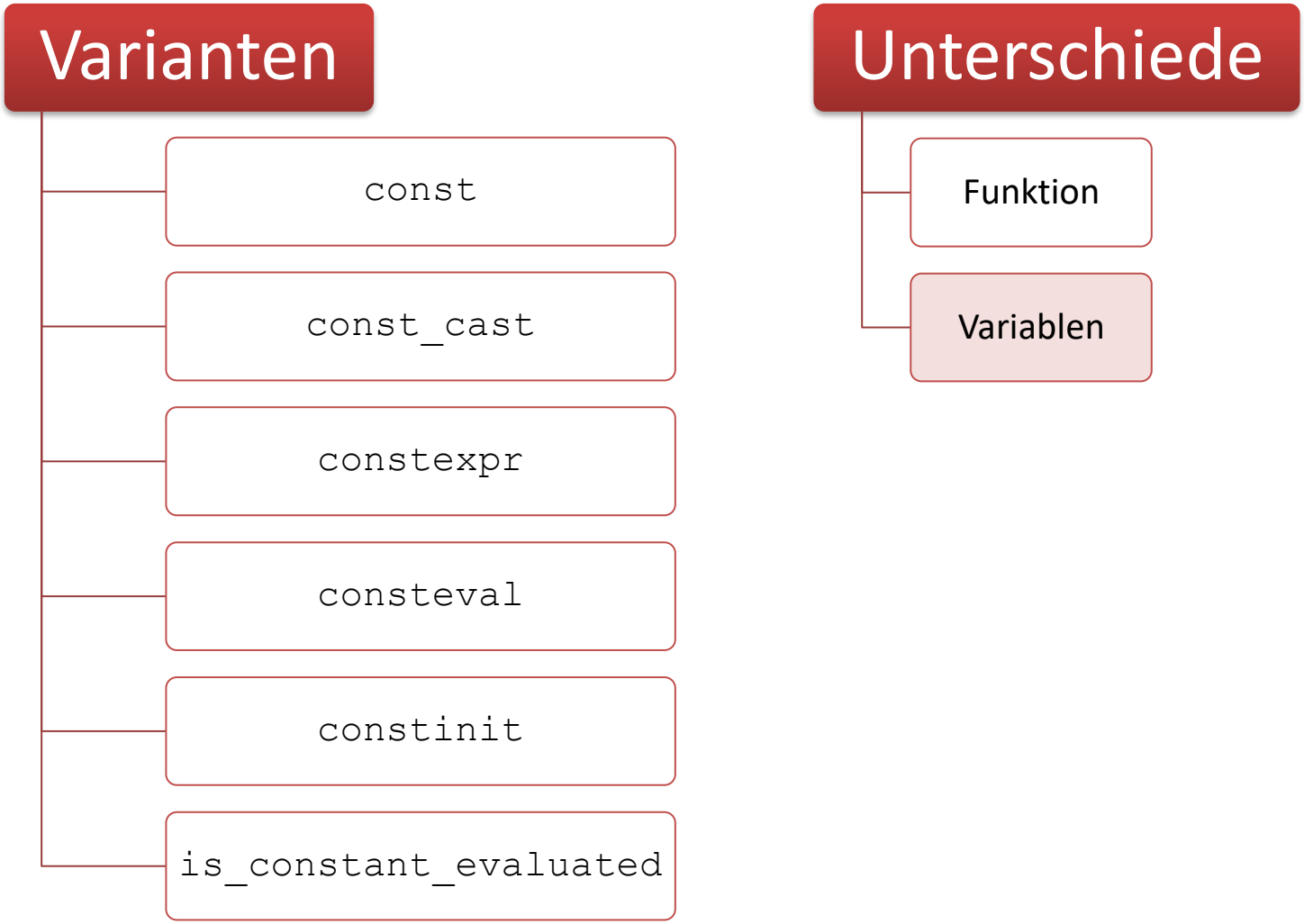


Funktion

```
#include <iostream>

int sqrRunTime(int n) { return n * n; }
consteval int sqrCompileTime(int n) { return n * n; }
constexpr int sqrRunOrCompileTime(int n) { return n * n; }

int main() {
    constexpr int prod1 = sqrRunTime(100); // ERROR
    constexpr int prod2 = sqrCompileTime(100);
    constexpr int prod3 = sqrRunOrCompileTime(100);

    int x = 100;
    int prod4 = sqrRunTime(x);
    int prod5 = sqrCompileTime(x); // ERROR
    int prod6 = sqrRunOrCompileTime(x);
}
```

consteval.cpp

Varianten von Konstantheit



Variablen

```
#include <iostream>

constexpr int constexprVal = 1000;
constinit int constinitVal = 1000;

int main() {
    auto val = 1000;
    const auto res = ++val;

    std::cout << "res: " << ++res << '\n'; // ERROR
    std::cout << "++constexprVal: " << ++constexprVal << '\n'; // ERROR
    std::cout << "++constinitVal: " << ++constinitVal << '\n';

    constexpr auto localConstexpr = 1000;
    constinit auto localConstinit = 1000; // ERROR
}
```

constexprConstinit.cpp

Variablen

Initialisierung einer lokalen nicht-konstanten Variablen zur Compilezeit

```
consteval auto doubleMe(auto val) {
    return 2 * val;
}

int main() {

    auto res = doubleMe(1010);
    ++res;        // 2021
}
```

compileTimeInitializationLocal.cpp

Varianten von Konstantheit

Varianten

- `const`
- `const_cast`
- `constexpr`
- `consteval`
- `constinit`
- `is_constant_evaluated`

Unterschiede

- Funktion
- Variablen


Blog: ModernesCpp.com
Schulung: ModernesCpp.de
Mentoring: ModernesCpp.org
Rainer Grimm
Schulung und Mentoring