

# Funktionale Programmierung mit modernem C++

Rainer Grimm

Schulungen, Coaching und  
Technologieberatung

# Funktionale Programmierung in modernem C++

Funktional in C++

Warum Funktional?

Definition

Charakteristiken

Was fehlt (noch) in C++?

# Funktionale Programmierung in modernem C++

Funktional in C++

Warum funktional?

Definition

Charakteristiken

Was fehlt (noch) in C++?

# Funktional in C++

- Automatische Typableitung

```
std::vector<int> myVec;  
for (auto v: myVec) std::cout << v << " " ;
```

- Lambda-Funktionen

```
int a= 2000, b= 11;  
auto sum= std::async( [=]{return a+b;} );
```

- Partielle Funktionsanwendung

- `std::function` und `std::bind`
- Lambda-Funktionen und `auto`



Haskell Curry

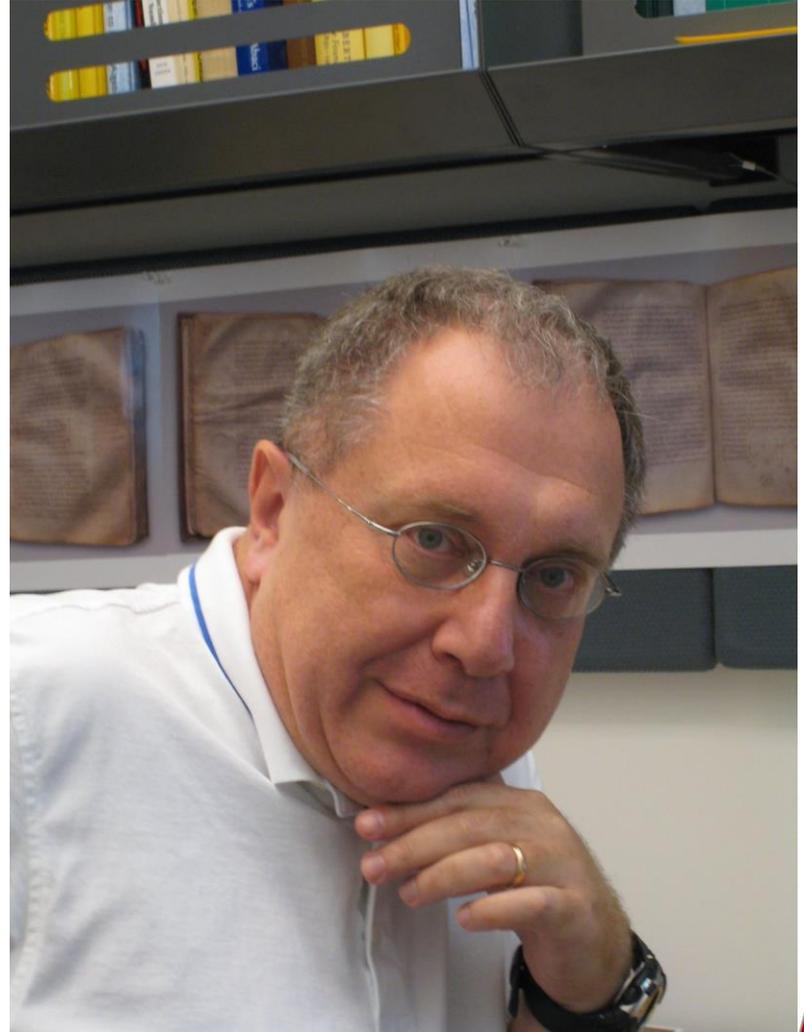


Moses Schönfinkel

# Funktional in C++

## Generische Programmierung

- ML führt generische Programmierung ein.
- Alexander Stepanov (Vater der STL) wurde von Lisp beeinflusst.
- Template Metaprogrammierung ist eine rein funktionale Subsprache in der imperativen Sprache C++.



# Funktionale Programmierung in modernem C++

Funktional in C++

Warum Funktional?

Definition

Charakteristiken

Was fehlt (noch) in C++?

# Warum Funktional?

- Effizientere Nutzung der Standard Template Library

```
std::accumulate(v.begin(), v.end(), 1,  
                [](int a, int b){ return a*b; });
```

- Funktionale Muster erkennen

```
template <int N>  
struct Fac{ static int const val= N * Fac<N-1>::val; };
```

```
template <>  
struct Fac<0>{ static int const val= 1; };
```

- Besserer Programmierstil
  - Bewusster Umgang mit Seiteneffekten (globalen Variablen)
  - Kurz und prägnante Programmierung
  - Verständlichere Programmierung

# Funktionale Programmierung in modernem C++

Funktional in C++

Warum Funktional?

**Definition**

Charakteristiken

Was fehlt (noch) in C++?

# Definition

Funktionale Programmierung ist Programmierung mit **mathematischen** Funktionen.

- Mathematische Funktionen sind Funktionen, die jedes Mal den selben Wert zurückgeben, falls sie die gleichen Argumente erhalten (Referenzielle Transparenz).
- Mathematische Funktionen werden auch reine Funktionen genannt.

# Funktionale Programmierung in modernem C++

Funktional in C++

Warum Funktional?

Definition

**Charakteristiken**

Was fehlt (noch) in C++?

# Charakteristiken



# Charakteristiken



# Funktionen erster Ordnung

Funktionen sind Funktionen erster Ordnung (first class objects). ➡ Sie verhalten sich wie Daten.

- Funktionen können
  - als Argumente an Funktionen übergeben werden.
  - von Funktionen zurückgegeben werden.
  - an Variablen zugewiesen oder gespeichert werden.

# Funktionen erster Ordnung

```
map<const char, function<double(double, double)>> tab;  
tab.insert(make_pair('+', [](double a, double b){ return a + b; } ));  
tab.insert(make_pair('-', [](double a, double b){ return a - b; } ));  
tab.insert(make_pair('*', [](double a, double b){ return a * b; } ));  
tab.insert(make_pair('/', [](double a, double b){ return a / b; } ));
```

```
cout << "3.5+4.5= " << tab['+'](3.5, 4.5) << endl;  8
```

```
cout << "3.5*4.5= " << tab['*'](3.5, 4.5) << endl;  15.75
```

```
tab.insert(make_pair('^',  
                [](double a, double b){ return pow(a, b); } ));
```

```
cout << "3.5^4.5= " << tab['^'](3.5, 4.5) << endl;  280.741
```

# Charakteristiken



# Funktionen höherer Ordnung

Funktionen, die Funktionen als Argumente annehmen oder als Ergebnis zurückgeben können

map, filter, and reduce  
explained with emoji 😂

```
map([🐮, 🍷, 🐔, 🌽], cook)  
=> [🍔, 🍟, 🍗, 🍿]
```

```
filter([🍔, 🍟, 🍗, 🍿], isVegetarian)  
=> [🍟, 🍿]
```

```
reduce([🍔, 🍟, 🍗, 🍿], eat)  
=> 💩
```

# Funktionen höherer Ordnung

Jede Programmiersprache, die Programmierung in funktionaler Art unterstützt, bietet die Funktionen **map**, **filter** und **fold** an.

Haskell	Python	C++
<code>map</code>	<code>map</code>	<code>std::transform</code>
<code>filter</code>	<code>filter</code>	<code>std::remove_if</code>
<code>fold*</code>	<code>reduce</code>	<code>std::accumulate</code>

- **map**, **filter** und **fold** sind drei mächtige Funktionen, die in vielen Kontexten verwendet werden  
➔ `map + reduce = MapReduce`

# Funktionen höherer Ordnung

## Listen und Vektoren

- Haskell

```
vec= [1 .. 9]
```

```
str= ["Programming", "in", "a", "functional", "style."]
```

- C++

```
vector<int> vec{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
vector<string> str{"Programming", "in", "a", "functional",  
"style."}
```



**Die Ergebnisse werden in Haskell Notation dargestellt.**

# Funktionen höherer Ordnung: map

- Haskell

```
map(\a → a^2) vec  
map(\a -> length a) str
```

- C++

```
transform(vec.begin(), vec.end(), vec.begin(),  
          [](int i){ return i*i; });  
transform(str.begin(), str.end(), back_inserter(vec2),  
          [](string s){ return s.length(); });
```



**[1,4,9,16,25,36,49,64,81]**  
**[11,2,1,10,6]**

# Funktionen höherer Ordnung: filter

- Haskell

```
filter(\x-> x<3 || x>8) vec  
filter(\x -> isUpper(head x)) str
```

- C++

```
auto it= remove_if(vec.begin(), vec.end(),  
                  [](int i){ return !((i < 3) or (i > 8)) });  
auto it2= remove_if(str.begin(), str.end(),  
                   [](string s){ return !( isupper(s[0])); });
```

 **[1,2,9]**  
**["Programming"]**

# Funktionen höherer Ordnung: reduce

- Haskell

```
foldl (\a b → a * b) 1 vec
foldl (\a b → a ++ ":" ++ b) "" str
```

- C++

```
accumulate(vec.begin(), vec.end(), 1,
            [](int a, int b){ return a*b; });
accumulate(str.begin(), str.end(), string(""),
            [](string a, string b){ return a+":"+b; });
```



**362800**

**":Programming:in:a:functional:style."**

# Charakteristiken



# Unveränderliche Daten

Daten sind in rein funktionalen Programmiersprachen unveränderlich.

- Konsequenzen
    - Es gibt keine
      - Zuweisungen: `x = x + 1`, `++x`
      - Schleifen: `for`, `while`, `until`
    - Beim Daten modifizieren werden
      - veränderte Kopien der Daten erzeugt.
      - die ursprünglichen unveränderlichen Daten geteilt.
- ➔ Unveränderliche Daten sind threadsicher.

# Unveränderliche Daten

## ■ Haskell

```
qsort [] = []  
qsort (x:xs) = qsort [y | y <- xs, y < x] ++ [x] ++ qsort [y | y <- xs, y >= x]
```

## ■ C

```
void quickSort(int arr[], int left, int right) {  
    int i = left, j = right;  
    int tmp;  
    int pivot = arr[abs((left + right) / 2)];  
    while (i <= j) {  
        while (arr[i] < pivot) i++;  
        while (arr[j] > pivot) j--;  
        if (i <= j) {  
            tmp = arr[i];  
            arr[i] = arr[j];  
            arr[j] = tmp;  
            i++; j--;  
        }  
    }  
    if (left < j) quickSort(arr, left, j);  
    if (i < right) quickSort(arr, i, right);  
}
```

# Charakteristiken



# Reine Funktionen

Reine Funktionen	Unreine Funktionen
Erzeugen immer dasselbe Ergebnis, wenn sie die gleichen Argumente erhalten.	Können verschiedene Ergebnisse für dieselben Argumente erzeugen.
Besitzen keine Seiteneffekte.	Können Seiteneffekte besitzen.
Verändern nie den Zustand.	Können den globalen Zustand des Programms ändern.

## Vorteile

- Korrektheitsbeweise sind einfacher durchzuführen.
- Refactoring und Test ist einfacher möglich.
- Ergebnisse von Funktionsaufrufe können gespeichert werden.
- Die Ausführungsreihenfolge der Funktion kann umgeordnet oder parallelisiert werden.

# Reine Funktionen

Monaden sind Haskell's Antwort auf die unreine Welt.

- Eine Monade
  - kapselt die unreine Welt.
  - ist ein imperatives Subsystem.
  - repräsentieren eine Rechenstruktur.
  - definieren die Komposition von Berechnungen



➔ Funktionale Design Pattern für generische Typen

# Reine Funktionen

Reader Monade

List Monade

**I/O Monade**

**STM Monade**

Error Monade

State Monade

Maybe Monade

# Charakteristiken



# Rekursion

Ist die Kontrollstruktur in der funktionalen Programmierung.

- Schleifen
  - benötigen eine Laufvariable: `(for int i=0; i <= 0; ++i)`  
➔ Variablen existieren nicht in rein funktionalen Sprachen.
- Rekursion in Kombination mit dem Verarbeiten von Listen ist ein mächtiges Pattern in funktionalen Sprachen.

# Rekursion

- Haskell

```
fac 0= 1  
fac n= n * fac (n-1)
```

- C++

```
template<int N>  
struct Fac{  
    static int const value= N * Fac<N-1>::value;  
};
```

```
template <>  
struct Fac<0>{  
    static int const value = 1;  
};
```

**➔ fac(5) == Fac<5>::value == 120**

# Charakteristiken



# Verarbeitung von Listen

**List Processing** (Lisp) ist typisch für funktionale Sprachen.

- Transformiere die Liste in ein andere Liste.
  - Reduziere die Liste auf einen Wert.
- 
- Funktionale Pattern für die Listenverarbeitung
    1. Prozessiere den Kopf der Liste.
    2. Prozessiere rekursive den Rest der Liste .

```
mySum [] = 0
```

```
mySum (x:xs) = x + mySum xs
```

```
mySum [1, 2, 3, 4, 5]  15
```

# Verarbeitung von Listen

```
template<int ...> struct mySum;
```

```
template<> struct  
mySum<>{  
    static const int value= 0;  
};
```

```
template<int i, int ... tail> struct  
mySum<i,tail...>{  
    static const int value= i + mySum<tail...>::value;  
};
```

```
int sum= mySum<1, 2, 3, 4, 5>::value;
```

 **sum == 15**

# Charakteristiken



# Bedarfsauswertung

Werte nur aus, was benötigt wird.

- Haskell ist faul (lazy evaluation)

```
length [2+1, 3*2, 1/0, 5-4]
```

- C++ ist gierig (greedy evaluation)

```
int onlyFirst(int a, int){ return a;}  
onlyFirst(1, 1/0); 
```

- Vorteile

- Sparen von Zeit und Speicher
- Verarbeitung von unendlichen Datenstrukturen

# Bedarfsauswertung

- Haskell

```
successor i= [i..]
```

```
take 5 (successor 10)
```

➔ **[10, 11, 12, 13, 14]**

```
[1..]= [1, 2, 3, 4, 5, 6, 7, 8, 9, ... Control-C
```

```
odds= takeWhile (< 1000) . filter odd . map (^2)
```

```
odds [1..]
```

➔ **[1, 9, 25, ... , 841, 961]**

- Spezialfall in C++: Kurzschlussauswertung

```
if (true or (1/0)) cout << "short circuit evaluation";
```

# Funktionale Programmierung in modernem C++

Funktional in C++

Warum Funktional?

Definition

Charakteristiken

Was fehlt (noch) in C++?

# Was fehlt (noch) in C++?

**List comprehension:** Syntactic Sugar für `map` und `filter`

- An mathematische Notation angelehnt

$\{ v*v \mid v \in \mathbb{N}, v \bmod 2 = 0 \}$  : Mathematik

`[n*n | n <- [1..], n `mod` 2 == 0 ]` : Haskell

- Haskell

`[n | n <- [1..6] ]`

**➔ [1, 2, 3, 4, 5, 6]**

`[n*n | n <- [1..6] ]`

**➔ [1, 4, 9, 16, 25, 36]**

`[n*n | n <- [1..6], n `mod` 2 == 0]`

**➔ [4, 16, 36]**

# Was fehlt (noch) in C++?

Funktionskomposition: fluent interface

- Haskell

```
(reverse . sort) [10, 2, 8, 1, 9, 5, 3, 6, 4, 7]
```

➔ **[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]**

```
isTit (x:xs) = isUpper x && all isLower xs
```

```
sortTitLen = sortBy(comparing length) . filter isTit . words
```

```
sortTitLen "A Sentence full of Titles ."
```

➔ **["A", "Titles", "Sentence"]**

# Was fehlt (noch) in C++?

Mit C++20 erhält C++ die Ranges-Library von Eric Niebler

- Algorithmen arbeiten auf dem ganzen Container

```
std::vector<int> v{1,2,3,4,5,6,7,8,9};  
  
std::sort(v.begin(), v.end());           // classic  
  
std::sort(v);                             // new
```

- Ranges unterstützen Bedarfsauswertung,  
Funktionskomposition und Range Comprehension

```
int total= std::accumulate(view::ints(1) |  
    view::transform([](int x){ return x*x; }) |  
    view::take(10), 0);
```

 total= sum \$ take 10 \$ map (\x -> x\*x) [1..]

# Funktionale Programmierung in modernem C++

Funktional in C++

Warum Funktional?

Definition

Charakteristiken

Was fehlt (noch) in C++?

# Modernes C++

The image shows a business card for Rainer Grimm. At the top left, a red circle contains the text 'Alle meine Leistungen auch in Englisch.' To the right of this is the logo 'MC++' in large, bold letters. Below the logo is a photograph of a laptop displaying code and a white mug. In the bottom right corner of the card is a portrait of Rainer Grimm with the name 'Rainer Grimm' written below it. On the left side, the text 'C++ und Python' is followed by three red arrows pointing to 'Schulungen', 'Coaching', and 'Technologieberatung'. At the bottom left, there is an email icon followed by 'schulung@grimm-jaud.de' and a phone icon followed by '0 74 72 / 91 74 41'. At the bottom center, the website 'www.ModernesCpp.de' is printed.

Alle meine Leistungen auch in Englisch.

**MC++**

**C++ und Python**

- ➔ Schulungen
- ➔ Coaching
- ➔ Technologieberatung

Rainer Grimm

schulung@grimm-jaud.de  
0 74 72 / 91 74 41

www.ModernesCpp.de

- Schulungen, Coaching und Technologieberatung durch Rainer Grimm
  - [www.ModernesCpp.de](http://www.ModernesCpp.de)
- Blogs zu modernem C++
  - [www.grimm-jaud.de](http://www.grimm-jaud.de) (Deutsch)
  - [www.ModernesCpp.com](http://www.ModernesCpp.com) (Englisch)
- Kontakt
  - [@rainer\\_grimm](https://twitter.com/rainer_grimm) (Twitter)
  - [schulungen@grimm-jaud.de](mailto:schulungen@grimm-jaud.de)