



# Praxis des Programmierens

Clemens Gröpl

Wintersemester 2009

16. C++ -- *Das Ende*

Bioinformatik  
Universität Greifswald

An dieser Stelle ein herzlicher Dank an Oliver Kohlbacher  
für die Überlassung der Präsentation! - Clemens Gröpl

# Informatik II

Oliver Kohlbacher

Sommersemester 2006

16. C++ -- *Das Ende*

Abt. Simulation biologischer Systeme  
WSI/ZBIT, Eberhard Karls Universität Tübingen

# Gliederung

---

- STL revisited
  - Bausteine der STL
  - Funktionsobjekte
  - Verwendung
- Die Kunst des Programmierens
  - Rekapitulation: wo sind wir?
  - Ist Programmieren eine Kunst?
  - Was ist Kunst?
  - Was ist gute Kunst?
  - Was ist ein gutes Programm?
  - Handelsübliche Fallstricke

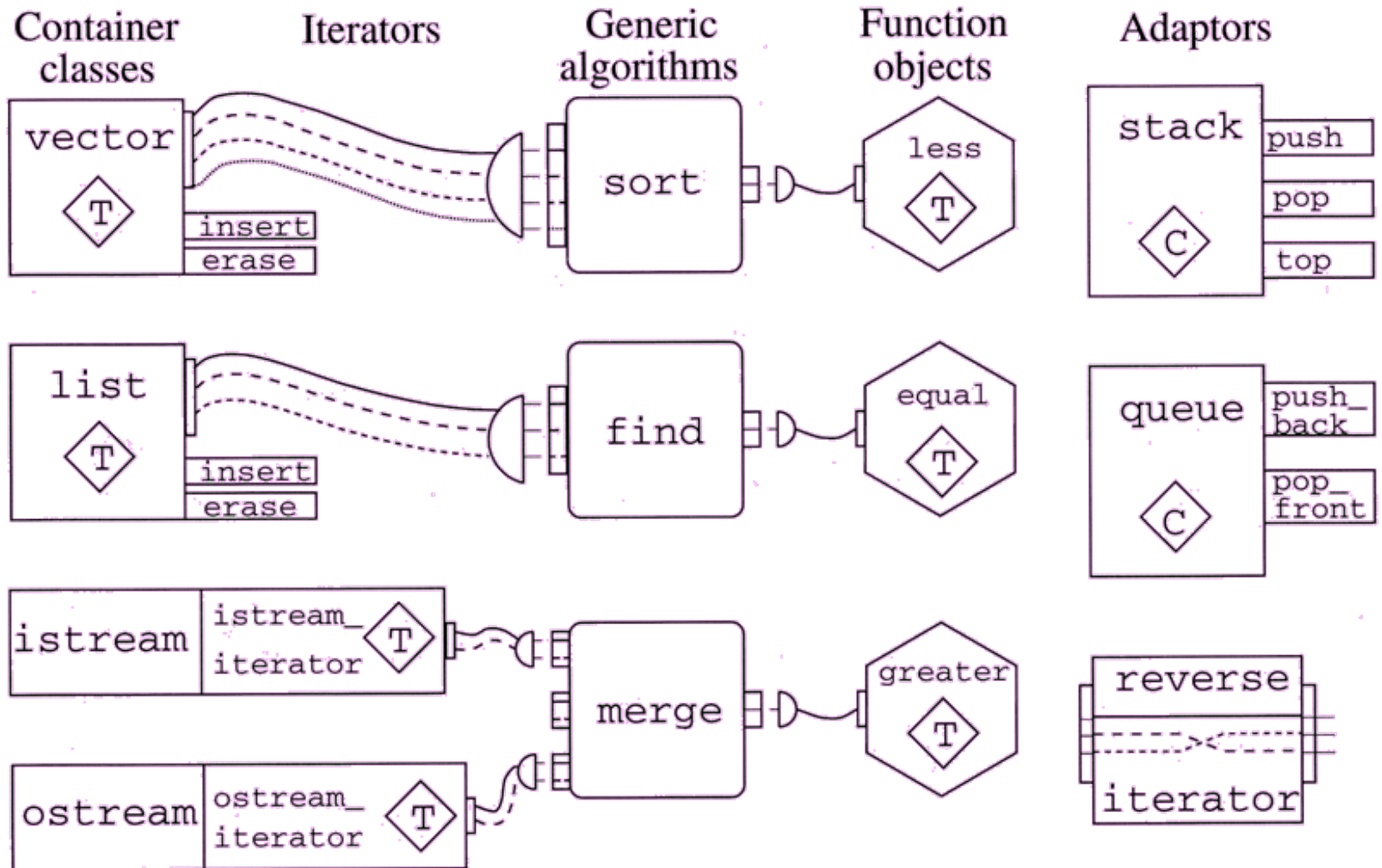
# Funktionsobjekte

- Ein Objekt einer Klasse, die `operator ()` definiert, lässt sich wie eine Funktion aufrufen:

```
class Verdoppler
{
    public:
    int operator () (int i)
    {
        return 2 * i;
    }
};
...
Verdoppler v;
int x = v(2); // x = 4
```

- Solche Objekte nennt man **Funktionsobjekte** oder **Funktoren**

# Bausteine der STL



# Funktoren in der STL

- Funktoren werden in der STL oft als **Argumente generischer Algorithmen verwendet**
- Bisher haben wir dazu stets Funktionen verwendet
- Funktionsobjekte sind jedoch flexibler:
  - Instanz einer Klasse kann zusätzliche Elemente haben
  - Verhalten ist dadurch einfacher veränderbar

**Beispiel:** Zähle alle Wörter mit mehr als sechs Buchstaben

```
bool mehrAlsSechs(const string& s)
{
    return s.size() > 6;
}

vector<string> text = ...;
cout << "Anzahl Woerter:"
     << count_if(text.begin(), text.end(),
                 mehrAlsSechs) << endl;
```

# Funktoren in der STL

- Funktorobjekte können mit unterschiedlichen Parametern initialisiert werden
- Dadurch ist das Zählen unterschiedlicher Wortlängen eleganter machbar also im vorherigen Beispiel:

```
class MehrAls // 16_mehral.C
{
public:
    MehrAls(unsigned int n) : n_(n) {}
    bool operator () (const string& s)
    { return s.size() > n_; }
protected:
    unsigned int n_;
};
...
for (unsigned int i = 1; i < 20; i++)
{
    cout << "mehr als " << i << " Buchstaben:"
        << count_if(text.begin(), text.end(), MehrAls(i))
        << endl;
}
```

# Funktoren der STL

---

- STL enthält nützliche vordefinierte Funktoren
- Jede Funktorklasse enthält einen **operator** () und führt eine wohldefinierte Operation aus
- Funktoren sind im Header **functional** definiert
- **Beispiele:**
  - **plus<T>** wendet operator + an
  - **minus<T>** wendet operator - an
  - **multiplies<T>** wendet operator \* an
  - **divides<T>** wendet operator / an
  - **modulus<T>** wendet operator % an
  - **negate<T>** wendet operator - an (unär)
- In Kombination mit den Standardalgorithmen lassen sich damit viele arithmetische Probleme elegant lösen

# Funktoren der STL

- Funktoren sind meistens **binär** oder **unär**
- Binäre Funktoren akzeptieren zwei Argumente, unäre ein Argument
  - `plus<T>` ist ein **binärer Funktor**
  - `negate<T>` ist ein **unärer Funktor**
- Beispiel aus der STL:

```
template <class T>
class multiplies
  : public binary_function<T, T, T>
{
  public:
  T operator() (const T& x, const T& y) const
  { return x * y; }
};
```

- Alle STL-Funktoren sind von gemeinsamen Basisklassen (`unary_function<>`, `binary_function<>`) abgeleitet und lassen sich dadurch auch polymorph einsetzen

# Funktoren der STL

---

- Arithmetische STL-Algorithmen gibt es meist in zwei Varianten: mit einem Standardfunktoren und mit frei wählbarem Funktoren
- `accumulate` gibt es z.B. in diesen beiden Varianten:

```
template <typename Iter, typename T>  
T accumulate(Iter start, Iter end, T init);
```

```
template <typename Iter, typename T,  
          typename BinaryOp>  
T accumulate(Iter start, Iter end, T init,  
             BinaryOp op);
```

- Damit lassen sich dann statt Summen z.B. Produkte berechnen:

```
vector<float> v = ...;  
float produkt = accumulate(v.begin(), v.end(),  
                           1.0, multiplies<float>());
```

# Funktoren der STL

---

Weitere Funktoren dienen dem Vergleich...

- `equal_to<T>` operator `==`
- `not_equal_to<T>` operator `!=`
- `greater<T>` operator `>`
- `greater_equal<T>` operator `>=`
- `less<T>` operator `<`
- `less_equal<T>` operator `<=`

...oder Logikoperationen:

- `logical_and<T>` operator `&&`
- `logical_or<T>` operator `||`
- `logical_not<T>` operator `!`

# Beispiel: Sortierrichtung ändern

```
// 16_sort.C
...
vector<int> v(20);
generate(v.begin(), v.end(), random);
// Füllt Array mit Zufallszahlen
sort(v.begin(), v.end()); // Aufsteigend
copy(v.begin(), v.end(),
      ostream_iterator<int>(cout, " "));

sort(v.begin(), v.end(), greater<int>()); // Absteigend
copy(v.begin(), v.end(),
      ostream_iterator<int>(cout, " "));
```

# Wdh: Informatik II - Ziele

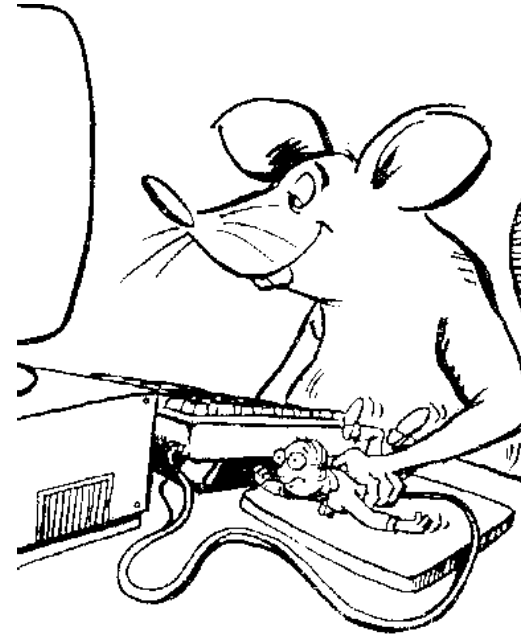
---

## Zwei Hauptziele

- Aus Mausschubsern Programmierer machen
- Aus Programmierern Informatiker machen
  - **Verstehen**, was der Rechner aus Ihrem Code wirklich macht, wie und warum (Teil I)
  - Lernen, statt lauffähiger Programme **clevere, schnelle, elegante Programme** zu erzeugen (Teil II+III)
    - ⇒ Back to the roots!
    - ⇒ Kein Eclipse mehr
    - ⇒ Keine graphischen Benutzerschnittstellen

# Wdh: Von Hackern und Mausschubsern

## Mausschubser



### *mouse pusher*

A person that prefers a mouse over a keyboard; originally used for Macintosh fans. The derogatory implication is that the person has nothing but the most superficial knowledge of the software he/she is employing, and is **incapable of using or appreciating the full glory of the command line.**

# Wdh: Von Hackern und Mausschubsern

## Richtige Programmierer



# The Art of Computer Programming

---

- Warum hat Don Knuth sein Buch "*The Art of Computer Programming*" genannt?
- In seiner Rede zur Verleihung des Turing Awards 1974 geht er darauf ein:

*"Science is knowledge which we understand so well that we can teach it to a computer; and if we don't fully understand something, it is an art to deal with it." ([3], S. 668)*

# The Art of Computer Programming

---

- Kunst kommt in der Tat von Können!
  - mhd. *kunst* = *Kenntnis, Meisterschaft*
  - engl. *art* vom lat. *ars, artis* = *Geschicklichkeit, Kunstfertigkeit*
- Programmieren ist eine Kunst
  - ⇒ Programmieren erfordert Können
  - ⇒ Können erfordert Übung

# The Art of Computer Programming

---

*"It is my purpose to transmit the importance of good taste and style in programming, [but] the specific elements of style presented serve only to illustrate what benefits can be derived from 'style' in general. In this respect I feel akin to the teacher of composition at a conservatory: He does not teach his pupils how to compose a particular symphony, he must help his pupils to find their own style and must explain to them what is implied by this. (It has been this analogy that made me talk about 'The Art of Programming.' (E. Dijkstra, in [3], S. 670)*

# Sprache bietet Freiräume

---

- SIEG GEGEN SCHWEDEN. Deutschland entweicht die Elche (*spiegel.de*)
- 2:0 gegen Schweden - Deutschland im Viertelfinale (*Lübecker Nachrichten*)
- 2:0 gegen Schweden. Poldi machte Peng-Peng! (*Bild*)
- Hurrah! Deutschland gegen Schweden: 2 zu 0 (*Rheinische Post*)
- 2:0 gegen Schweden. Deutscher Blitzstart ins Viertelfinale (*FAZ*)
- Deutschland erreicht WM-Viertelfinale: 2:0 gegen Schweden (*dpa*)

# Programmiersprachen

---

Auch Programmiersprachen bieten expressiven Freiraum:

```
int sum=0;
for(int i=0;i<v.size();sum+=v[i++]);
```

```
int sum = accumulate(v.begin(), v.end(), 0);
```

```
int sum = 0, i = 0;
while (i < v.size())
{
    sum = sum + v[i];
    i = i + 1;
}
```

# Computer Programming as an Art

---

Knuth am Ende seiner Turing-Award-Rede:

*"To summarize:*

*We have seen that computer **programming is an art**, because it applies **accumulated knowledge** to the world, because it **requires skill and ingenuity**, and especially because it produces **objects of beauty**. A programmer who subconsciously views himself as an artist will enjoy what he does and will do it better. Therefore we can be glad that people who lecture at computer conferences speak about the **state of the Art.**" ([3], S. 673)*

# The Art of Computer Programming

---

## Machine Longing for Rio (*Richard Gabriel*)

I seduced the machine  
sitting there solving a hard partial  
differential equation programmed by a geek  
hacking for an acoustician. I was like a siren  
with an outrageous sexual presence  
whose polarity I switched several times  
to lead the machine into confusion  
and deep desire. It was lost looking at small  
changes  
spreading out to a large picture when it displayed  
itself as a fractal set which swung so cool and  
swayed so gentle

*When I'm writing poetry, it feels like the center of my thinking  
is in a particular place, and when I'm writing code the center of  
my thinking feels in the same kind of place.*  
like music from rodas de samba at the botequins,  
parts the machine hid from all but its lovers,  
hid from those who use it only, who don't know

Richard Gabriel, Disruptive Engineer, San Mateo, CA

<http://www.dreamsongs.com/NewFiles/SharpTone.book2000-2001.pdf>

# Gedichte in Perl

---

## *The Invocation*

If light were dark and dark were light  
The moon a black hole in the blaze of night  
A raven's wing as bright as tin  
Then you, my love, would be darker than sin.

*(Jim Steinman)*

Das ganze in Perl implementiert:

```
if ((light eq dark) && (dark eq light) &&  
    ($blaze_of_night{moon} == black_hole) &&  
    ($ravens_wing{bright} == $tin{bright})) {  
my $love = $you = $sin{darkness} + 1; };
```

# Ein Python-Sonett, das $\pi$ berechnet

```
import sys

def main():
    k, a, b, a1, b1 = 2L, 4L, 1L, 12L, 4L
    while 1:
        p, q, k = k*k, 2L*k+1L, k+1L
        a, b, a1, b1 = a1, b1, p*a+q*a1, p*b+q*b1
        d, d1 = a/b, a1/b1
        while d == d1:
            output(d)
            a, a1 = 10L*(a%b), 10L*(a1%b1)
            d, d1 = a/b, a1/b1

def output(d):
    sys.stdout.write(`int(d)`)
    sys.stdout.flush()

main()
```

# Was macht gute Lyrik aus?

---

**"What is good poetry?**

Good poetry is poetry that does things that those who are serious about the genre or who can bring certain qualifications to the genre feel are good things to do or to have done. This is not to say that others who are not serious about the genre can't also respond to those good things."

...

**What is great poetry?**

Great poetry is poetry that is not only good in its parts, but consistently of high quality most of the way through. And it must satisfy both criteria for doing good things (exploring both artistic and non-artistic problems)."

*(Malcolm Hayward)*

# Was macht gute Programme aus?

---

- Kürze
- Leichte Lesbarkeit
- Klarheit

} = *Eleganz*

- Korrektheit
- Effizienz

} = *Wert*

- Erweiterbarkeit
- Durchdachtes Design
- Dokumentation

} = *Nachhaltigkeit*

# Probleme: Stil

---

- Korrekte Einrückung lässt die **Struktur** in Sekundenbruchteilen klar werden
- Leerzeichen vereinfachen optische Trennung in einzelne Worte; erhöhen **Lesegeschwindigkeit!**
- Je länger der Code, desto länger dauert das Verstehen: typedefs helfen!
- Schlecht gewählte Namen lassen **Bedeutung** nicht erkennen
- Unnötiges Abweichen von üblichen **Konventionen** verwirrt: *i, j, k, l* sind in der Informatik gängige Namen für Indices, warum *x* oder *aa* verwenden?
- **Nicht \*zu\* clever sein**: nicht die Lesbarkeit der Effizienz opfern

# Stil

---

*"Style is knowing who you are, what you want to say, and not giving a damn."*

(Gore Vidal)

*"In matters of grave importance, style, not sincerity, is the vital thing."*

(Oscar Wilde)

*"To do a dull thing with style-now THAT'S what I call art."*

(Charles Bukowski)

# Probleme: Lesbarkeit

---

```
int b, g=0, f=1;
for (b=0; b<100; )
{ g=g+v[b];
  f=f*v[b++]; }
```

```
int summe = 0;
int produkt = 1;

for (int i = 0; i < 100; ++i)
{
    summe += v[i];
    produkt *= v[i];
}
```

```
// accumulate addiert (default) bzw. multipliziert
// mit Funktor (times) alle Elemente des Iteratorbereichs.
summe = accumulate(v.begin(), v.end(), 0);
produkt = accumulate(v.begin(), v.end(), 1, times<int>);
```

# Probleme: Effizienz und Optimierung

---

- Make it **right** before you make it faster.
- Make it **fail-safe** before you make it faster.
- Make it **clear** before you make it faster.
- To make it faster, **change the algorithm** not small details in the code.
- Actually **test code** to see how fast it is.

# Probleme: Ahnungslosigkeit

```
bool isPrime(int p)
{
    for (int i = 2; i < p; i++)
    {
        if (p % i == 0)
        {
            return false;
        }
    }
    return true;
}

void sortGreater(vector<int>& v)
{
    bool sorted = false;
    while (not sorted)
    {
        sorted = true;
        for (int i = 0; i < v.size() - 1; ++i)
        {
            if (v[i] < v[i + 1])
            {
                swap(v[i], v[i + 1]);
                sorted = false;
            }
        }
    }
}
```

# Zuletzt

---

## *Was ist Kunst*

Hab'n Sie was mit Kunst am Hut?

Gut.

Denn ich möchte Ihnen allen  
etwas auf den Wecker fallen

Kunst ist was?

Das:

Kunst, das meint vor allen Dingen  
andren Menschen Freude bringen  
und aus vollen Schöpferhänden  
Spaß bereiten, Frohsinn spenden,  
denn die Kunst ist eins und zwar  
heiter. Und sonst gar nichts. Klar?

Ob das klar ist? Sie ist heiter!

Heiter und sonst gar nichts weiter!

Heiter ist sie! Wird es bald?

Heiter! Hab'n Sie das geschnallt?

Ja? Dann folgt das Resümee;

bitte sehr:

Obenstehendes ist zwar

alles Lüge, gar nicht wahr,

und ich meinte es auch bloß

irgendwie als Denkanstoß -

aber wenn es jemand glaubt:

ist erlaubt.

Mag ja sein, daß wer es mag.

Guten Tag.

*(Robert Gernhardt)*

## Literatur zu diesem Teil der Vorlesung:

- [1] Lippman, Lajoie, Moo, C++ Primer, 4. Auflage, Addison Wesley, 2005.  
(besonders Kapitel 16)
- [2] Stroustrup, Die C++ Programmiersprache, 4. Auflage, Addison Wesley, 2000. (besonders Kapitel 13)
- [3] Donald E. Knuth, Computer Programming as an Art, Comm. ACM, 1974, 17, 667-673.  
<http://fresh.homeunix.net/~luke/misc/knuth-turingaward.pdf>
- [4] Brian Kernighan and P. J. Plauger, The Elements of Programming Style. McGraw-Hill Book Company, New York, 1974.