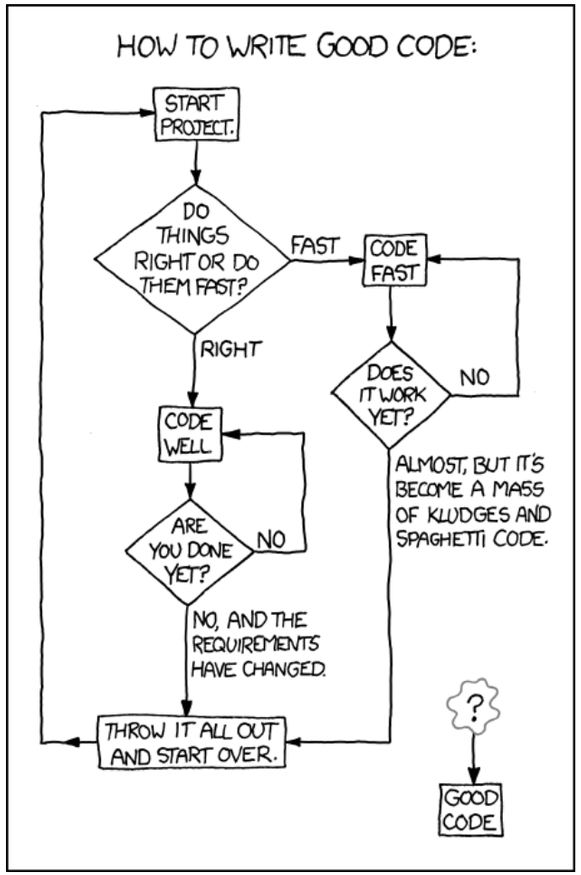
Universität Trier

APW



xkcd.com/844

Algorithmen für Programmierwettbewerbe

Vorlesung 1: Organisatorisches und Einführung

Philipp Kindermann

Sommersemester 2025

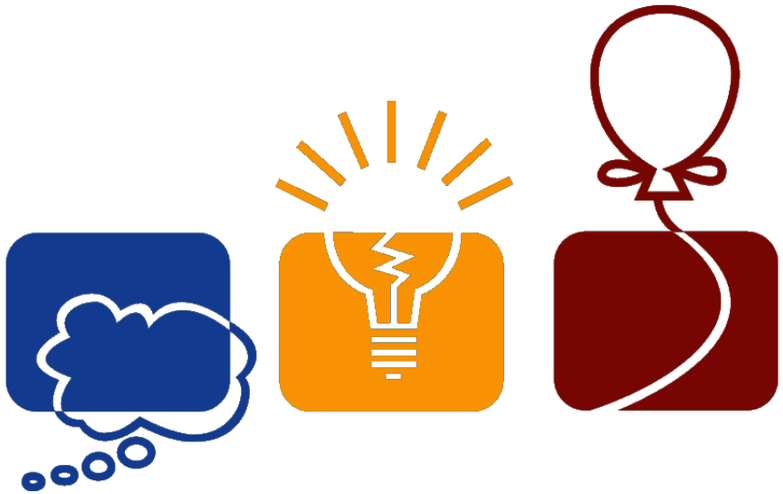
Was sind Programmierwettbewerbe?

International Collegiate Programming Contest

- gößter und bekanntester universitärer Programmierwettbewerb
- seit den 1970er Jahren
- 3er Teams, 5 Stunden, 10+ Aufgaben

Mehrstufig:

- World Finals
- Regionals: NWERC im November
  North West European Regional Contest
- Subregionals: GCPC im Juni
  German Collegiate Programming Contest



Warum Programmierwettbewerbe?

Spaß!

Trainiert algorithmische Denkweise

Übt Formalisieren!

Erfahrung mit C++

Trainiert systematisches Problemlösen

Wiederholt/festigt Paradigmen wie . . .

. . . Greedy-Algorithmen

. . . Dynamische Programmierung

. . . Divide & Conquer

Bereitet auf Coding Interviews wie . . . bei Google, Microsoft & Co. vor.

Organisatorisches

Vorlesung:

- 1x wöchentlich, Dienstag 10:15–11:45
- neuer Stoff

Übung:

- 1x wöchentlich, Donnerstag 12:15–13:45
- Aufgaben zum Stoff der Vorlesung
- Aufgaben sind bis kommenden Dienstag offen
- Lösungsideen zu den letztwöchentlichen Aufgaben
- Bearbeiten der Aufgaben während der Übung mit meiner Unterstützung

salesch.uni-trier.de

discord.gg/qAttRVVqPF

Anforderungen

Zulassung:

- Im Schnitt 2 gelöste Aufgaben pro Woche.

Regeln:

- Arbeiten in 2er-/3er-Gruppen
- Ich unterstütze euch während der Übung; außerhalb stellt ruhig Fragen oder tauscht euch im Discord aus.
- nicht: Lösungen / vorgefertigtes aus dem Internet (Doku online lesen ist ok)

Vor- und Nachbereitung:

- Konzepte aus Vorlesung für Übung verstehen und ggf. Test-Implementieren
- Aufgaben können anschließend zu Übungszwecken noch gelöst werden

Klausur:

- Mini-Contest (2 Stunden) am Ende des Semesters

Themen

- STL-Datenstrukturen
- Algorithmische Entwurfsparadigmen
- Dynamische Programmierung
- Strings
- Graphenalgorithmen (DFS, BFS, …)
- Dijkstra, Spannbäume, Union-Find, Bellmann Ford
- Flüsse, Matchings
- große Zahlen, Kombinatorik, Spiele
- ggT, kgV, Primzahlen, chinesischer Restsatz
- Strecken und Geraden, konvexe Hülle, Sweepline-Verfahren
- …?

How To ICPC?

Alle Aufgaben lesen!

- Schwierigkeit einschätzen
- Lösungsansatz? (Technik/Bausteine)
- Größenordnungen abschätzen
  Was kann ich mir leisten?
  Brute force oder was cleveres?

- Nicht zu sehr an einer schwierigen Aufgabe festbeißen

Lösungsprozess

1. Problem abstrakt formulieren

- Was ist gesucht?

2. Lösungskonzept entwickeln

- Passt das wirklich zur Aufgabenstellung
- Kann das überhaupt stimmen?
- Passt die Laufzeit?
- Reichen die Variablengrößen?

3. Implementieren

- Funktionalität aufteilen
- Optimierung nicht hier sondern in Schritt 2

Überblick

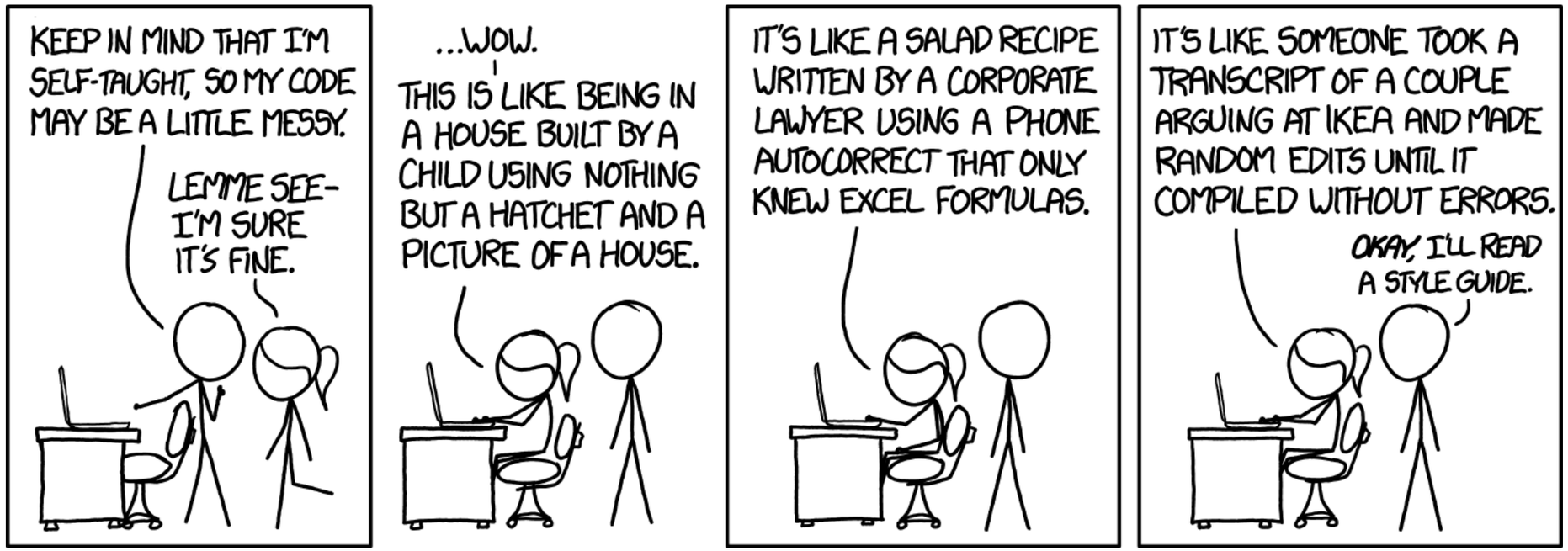
How to…

Programmieren

Komplexität

Aufgaben

xkcd.com/1513



Format

Aufgaben

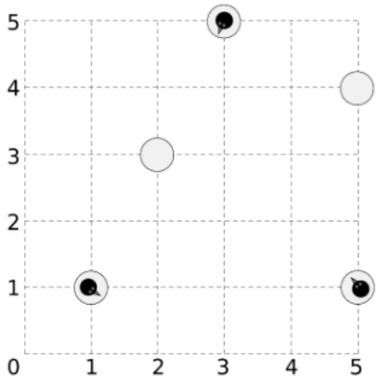
- mathematisch/algorithmisch
- meist als kleine Geschichte formuliert
- stark variierende Schwierigkeit, nicht sortiert (unbedingt alle Aufgaben lesen!)

Interaktion

- genau spezifizierte Eingabe (keine Fehlerbehandlung notwendig)
- Programm liest von `stdin` und schreibt nach `stdout`
- *Quelltext* hochladen, Programm wird dort gegen geheime Testdaten getestet.

Problem A
March of the Penguins
Problem ID: penguins
Time limit: 10 seconds

Somewhere near the south pole, a number of penguins are standing on a number of ice floes. Being social animals, the penguins would like to get together, all on the same floe. The penguins do not want to get wet, so they have use their limited jump distance to get together by jumping from piece to piece. However, temperatures have been high lately, and the floes are showing cracks, and they get damaged further by the force needed to jump to another floe. Fortunately the penguins are real experts on cracking ice floes, and know exactly how many times a penguin can jump off each floe before it disintegrates and disappears. Landing on an ice floe does not damage it. You have to help the penguins find all floes where they can meet.



A sample layout of ice floes with 3 penguins on them.

Input

On the first line one positive number: the number of testcases, at most 100. After that per testcase:

- One line with the integer N ($1 \leq N \leq 100$) and a floating-point number D ($0 \leq D \leq 100000$), denoting the number of ice pieces and the maximum distance a penguin can jump.
- N lines, each line containing x_i, y_i, n_i and m_i, denoting for each ice piece its X and Y coordinate, the number of penguins on it and the maximum number of times a penguin can jump off this piece before it disappears ($-10000 \leq x_i, y_i \leq 10000, 0 \leq n_i \leq 10, 1 \leq m_i \leq 200$).

Output

Per testcase:

- One line containing a space-separated list of 0-based indices of the pieces on which all penguins can meet. If no such piece exists, output a line with the single number -1.

Sample Input 1	Sample Output 1
2 5 3.5 1 1 1 1 2 3 0 1 3 5 1 1 5 1 1 1 5 4 0 1 3 1.1 -1 0 5 10 0 0 3 9 2 0 1 1	1 2 4 -1

Eine erste Aufgabe

APW

0 ICPC-Hello-World

0.1 Problem

The simple ICPC Hello World!

0.2 Input

The input file contains one line containing the string s, consisting only of alphanumeric characters without whitespace.

0.3 Output

The output should contain the literal string „Hello” without quotes, followed by a space, the input string s and then a single exclamation mark character and a newline character. Please refer to the example for clarification.

0.4 Sample Data

Input	Output
`World`	`Hello World!`

Lösung in Java

Main.java

```
import java.util.Scanner;

public class Main {
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
                System.out.println("Hello␣" + sc.next() + "!");
        }
}
```

Java ⇒ C++

- Dateiname egal!
- 1 Datei ≠ 1 Klasse
- main-Methode
- kein return!?

main.cpp

```
#include <iostream>
#include <string>

using namespace std;

int main() {
std::string name;
std::cin >> name;
std::cout << "Hello␣"
          << name << "!" << std::endl;
}
```

macht das Template...

Imports!

Namespaces
statt Packages

```
namespace std
    ...
}
```

```
"\n" + flush()
```

Ein- und Ausgabeoperatoren

Kompilieren und Ausführen

main.cpp

```cpp
#include <iostream>
#include <string>

using namespace std;

int main() {
  string name;
  cin >> name;
  cout << "Hello␣"
       << name << "!" << endl;
}
```

Kompilieren auf der Konsole

```
g++ -std=gnu++17 -Wall main.cpp
./a.out < sample.in
```

Angabe + Testdaten

In Moodle / Salesch

Testen mit Diff

```
./a.out < sample.in
  | diff - sample.out
```

Workflow

compile, run, diff → Salesch

`https://salesch.uni-trier.de`

Unter Windows:

```
a.exe < sample.in > ans.txt
FC sample.out ans.txt
```

Cheatsheet: `https://algo.uni-trier.de/lectures/apw/icpc.html`

Eingabe

main.cpp

```cpp
#include <iostream>
using namespace std;

int main() {
  int a,b;
  cin >> a >> b;
  cout << (a+b) << endl;
}
```

Beachte

In Competitive-Programming-Aufgaben nie Fehlerbehandlung nötig!

Summe von n Zahlen

main.cpp

```cpp
#include <iostream>
using namespace std;

int main() {
  int sum = 0, x;
  while(cin >> x) {
    sum += x;
  }
  cout << sum << endl;
}
```

Eingabe

```
␣␣␣␣␣␣␣␣␣1␣␣␣2␣3
␣␣␣␣␣␣␣␣␣4␣␣5␣7␣␣␣8
␣␣␣␣␣␣␣␣␣␣4␣␣2␣␣␣␣6
```

Whitespace

`cin` überspringt:

- Leerzeichen
- Tabs
- Newlines

Manuelles Eingabe-Ende

CTRL + D $\rightarrow$ End of File

Ausgabe

```
42
```

Endlosschleife

main.cpp

```
#include <iostream>
using namespace std;

int main() {
  for(;;) {
    cout << "loop" << endl;
  }
}
```

Ausführen

Abschießen mit CTRL + C
(Erinnerung: CTRL + D $\rightarrow$ EOF)

Überblick

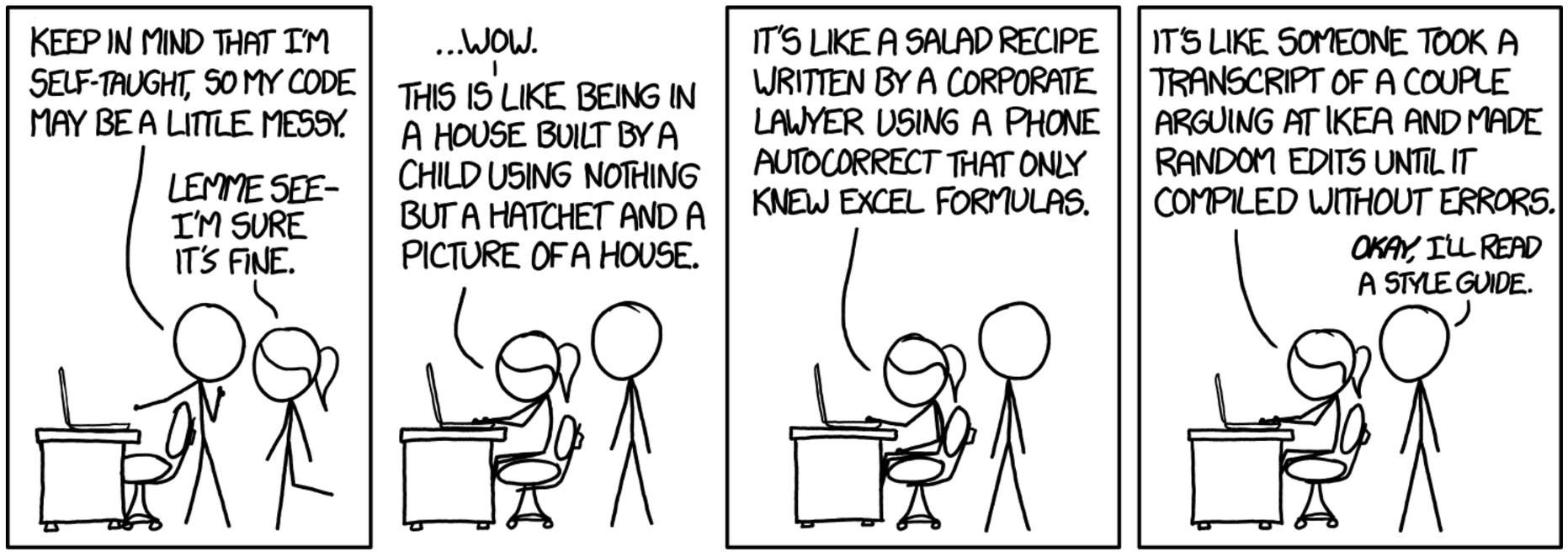
How to…

Programmieren

Komplexität

Aufgaben

xkcd.com/1513



Sauberer vs. dreckiger Code

- Niemand wird euren Code noch in x Wochen lesen.
  ➡ Wartbarkeit ist unwichtig.
- Globale Variablen sind nicht böse.
- Ihr müsst euren Code nur während des Schreibens verstehen.
- Aber: Wenn ihr Hilfe wollt, dann muss ich ihn auch nachvollziehen können.

Brauchen wir nicht

- Fehlerbehandlung
- manuelle Speicherverwaltung: `new` / `delete`
- Pointer
- Objektorientung (`class`)

Programmier-Paradigmen

Objektorientiertes Programmieren

- bringt Daten und Operationen in *Objekten* zusammen
- komplexe Probleme in Klassen zerlegen
- für ICPC fast nie hilfreich

Datenorientiertes Programmieren

- Was für Daten brauche ich?
- Wie (und wo) im Speicher repräsentieren?
- Wie Input verarbeiten um gewünschten Output zu erhalten?
- Funktionen für Teilprobleme

Datentypen

- Ganze Zahlen: `int` (32 bit), `unsigned int` (32 bit), ...
- Fake-Reele Zahlen: `float` (32 bit), `double` (64 bit)
- Booleans: `bool` (8 bit)
- Zeichenketten: `string` (?)
- ...

Integer-Größen

Größte Zahl, die in `int` passt, ist etwa 2 Milliarden.
Größte Zahl, die in `long long` passt, ist etwa 10^{18}.

Typ	**Größe**	**min**	**max**
`int`	32 bit	-2^{31}	$2^{31}-1$
`unsigned int`	32 bit	0	$2^{32}-1$
`long long`	64 bit	-2^{63}	$2^{63}-1$
`unsigned long long`	64 bit	0	$2^{64}-1$

Integer-Größen – besser

Besser:

Verwende explizite Typen aus `<cstdint>`: `int64_t`, `uint64_t`

Größte Zahl, die in `int32` passt, ist etwa 2 Milliarden.

Größte Zahl, die in `int64` passt, ist etwa 10^{18}.

Typ	**Größe**	**min**	**max**
`int32_t`	32 bit	-2^{31}	$2^{31}-1$
`uint32_t`	32 bit	0	$2^{32}-1$
`int64_t`	64 bit	-2^{63}	$2^{63}-1$
`uint64_t`	64 bit	0	$2^{64}-1$
`__int128`	128 bit	-2^{127}	$2^{127}-1$
`unsigned __int128`	128 bit	0	$2^{128}-1$

Eigene Typen

```
struct MyStruct {
  int a,b;
  string s;
};  /* Semikolon nicht vergessen! */
```

Verwendung

```
MyStruct ms; // muss nicht mit new erzeugt werden!
ms.a = 1;
ms.b = 2;
ms.s = "Hello";
// Alternative:
ms = MyStruct {1, 2, "Hello"};
```

Verhalten beim Kopieren

```
MyStruct x;
x.a = 42;
MyStruct y = x;
y.a = 23;

/* x.a == ? */
```

- `x.a == 42`
- Wir sind nicht in Java.
- `a` und `b` sind Werte, nicht Referenzen!
- `y` ist Kopie von `x`

Referenzen

```
MyStruct x;
x.a = 42;
MyStruct& y = x;
y.a = 23;

/* x.a == ? */
```

- `x.a == 23`
- Referenzen müssen explizit erzeugt werden.
- Referenz ist ein eigener Typ: `MyStruct&`
- Referenzen kann man nicht “umhängen”
  - ➡ Pointer

- In Java werden Objekte irgendwo im Speicher angelegt und ihr bekommt ausschließlich Referenzen
- In C++ könnt ihr direkt mit Structs und Referenzen darauf arbeiten.

Arrays?

C-Arrays sind einfach, schnell und oft ausreichend…

```
void foo() {
  int a[256];           // KONSTANTE Größe.

  for (int i = 0; i < 256; ++i)
    cout << a[i] << endl;
  for (int x: a)
    cout << x << endl;
}
```

…aber manchmal total bescheuert zu benutzen.

```
void bar() {
  int a[256];       // konstante Größe
  a[5];             // uninitialisiert -> nicht genullt!
  a.size();         // gibt's nicht
  a.resize(512);    // gibt's nicht
  a.push_back(42); // gibt's nicht
  int bla[1000000];// Stack evtl. zu klein -> overflow
}
```

Arrays?

C-Arrays sind einfach, schnell und oft ausreichend…

```
void foo() {
  int a[256];           // KONSTANTE Größe.

  for (int i = 0; i < 256; ++i)
    cout << a[i] << endl;
  for (int x: a)
    cout << x << endl;
}
```

etwas besser: `std::array`

```
void bar() {
  std::array<int, 256> a; // konstante Größe
  a.fill(0);          // initialisierung
  a.size();           // gibt's!
  a.resize(512);      // gibt's nicht
  a.push_back(42); // gibt's nicht
  std::array<int, 1000000> bla; // Stack evtl. zu klein
}
```

Ausgabe von n Zahlen in umgekehrter Reihenfolge

main.cpp

```
std::array<int, 1000> a;
// (grosse) Arrays immer global!
// a[0] ... a[999] moegliche Slots
// Groesse laesst sich nicht veraendern.
// -> zur compile-time festgelegt!

int main() {
  int n = 0;
  while (cin >> a[n])
    ++n;
  cout << "Es␣gibt␣" << n << "␣Zahlen\n";
  for (int i = 0; i < n; ++i)
    cout << a[n-i-1] << "␣";
  cout << endl;
}
```

Vektoren!

```cpp
#include <vector>
#include <iostream>
using namespace std;

int main() {
    vector<int> a(256);          // variable Größe
    cout << a[5] << endl;        // mit 0 initialisiert
    cout << a.size() << endl; // 256
    a.resize(512);               // Größe ändern
    a.clear();                   // vector leeren
    a.push_back(42);             // Wert ans Ende anfügen

    vector<int> bla(1000000); // kein Problem

    for(int i = 0; i < (int)a.size(); ++i) // cast
        cout << a[i] << endl;
    for(auto& x: a)
        cout << x << endl;
}
```

Vektoren von anderen Dingen

Genauso:

```
// Vektor von Strings
vector<string> s;                 // leerer Vektor
vector<string> s(10);             // 10 mal ""
vector<string> s(10,"Hello");     // 10 mal "Hello"
vector<string> s();               // ERROR! Funktionsdeklaration

// Vektor von Vektoren von Zeug
vector<vector<Zeug>> zv1;         // 0x0 Vektor
vector<vector<Zeug>> zv2(100);    // 100x0 Vektor
vector<vector<Zeug>> zv3(100, vector<Zeug>(10));
                                  // 100 x 10 Vektor

Zeug z;
zv2[4].push_back(z);              // zv2[4] hat danach Länge 1
zv3[42][3] = z;
```

Und wo liegt das Zeug jetzt im Speicher?

main.cpp

```
int a[1000];  // statisch!

int main() {
  int b[256]; // auf dem stack
  vector<int> c(128); // auf dem heap
                      // + 3*8 byte auf dem Stack
}
```

- Vektoren: Dynamischer Speicher/Heap-Allokation ohne Stress
- Der Vektor räumt auf, wenn sein Scope endet.
- Aber Vorsicht mit Kopien (teuer)
- Lebensdauer beachten

```
int &foo() {
  vector<int> x(42);
  return x[5];  // Referenz auf lokale Variable
}
```
→ Faustregel: Referenzen nur an Unterfunktionen “ausleihen”

Call by value

Bauen wir uns mal eine Funktion, die einen Vektor umdreht

```
void revert(vector<int>  v) {
  int n = (int)v.size();
  for (int i = 0; i < (n/2); ++i) {
    swap(v[i], v[n-i-1]);
  }
}

int main() {
  vector<int> z = {1, 2, 3, 4, 5}; // Initializer list
  revert(z);
  // z verändert?
}
```

Call by value ➡ `revert` arbeitet auf Kopie!

Sortieren

```
#include <algorithm>

vector<int> v;
sort(v.begin(), v.end()); // Sortiert v aufsteigend
```

- `begin()` zeigt an den Anfang
- `end()` zeigt *hinter* das Ende

Und wenn es nicht um `int` geht?

Oder nicht um *aufsteigendes* Sortieren?

Komparator / Operatorüberladung / Lambdas

Eigener Komparator:

```
bool kleiner(const Type& m1, const Type& m2) {
  return [mysterious formula]; // true or false
}
vector<Type> v;
sort(v.begin(),v.end(),kleiner);
```

Operator überladen:

```
bool operator<(const Type& m1, const Type& m2) {
  return [mysterious formula]; // true or false
}
vector<Type> v;
sort(v.begin(),v.end());
```

Lambda (anonyme Funktion)

```
vector<Type> v;
sort(v.begin(),v.end(), [](const Type& m1, const Type& m2){
  return [mysterious formula]; // true or false
});
```

Überblick

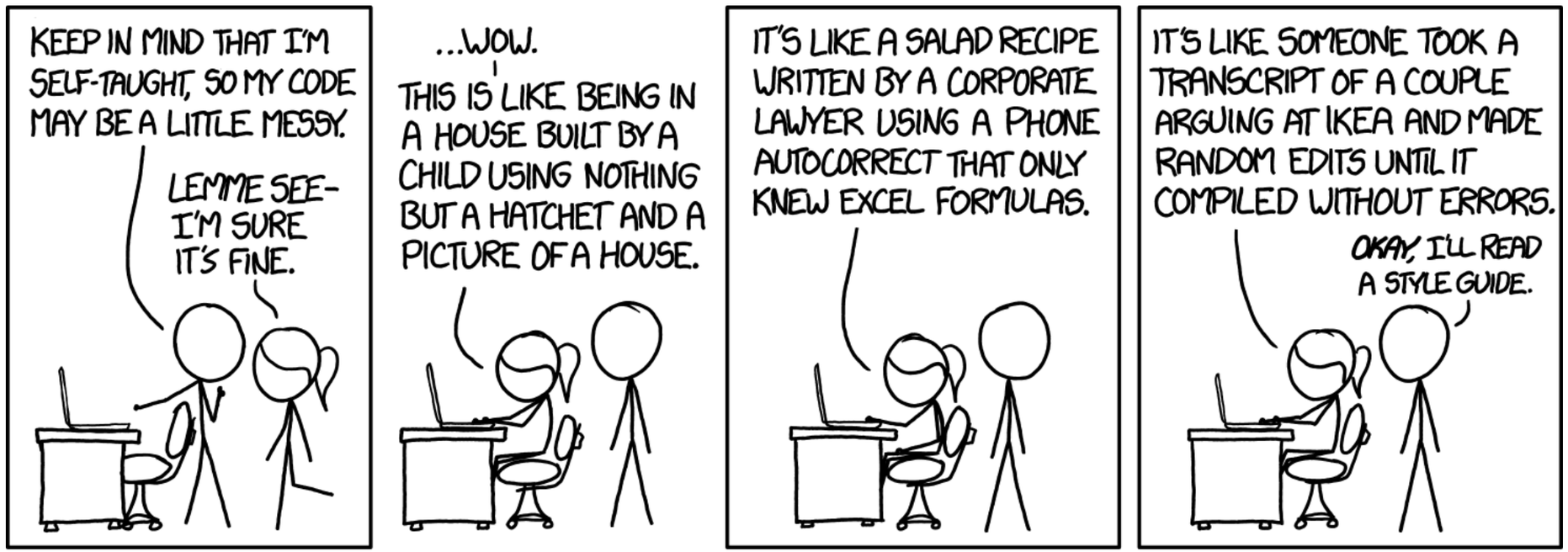
How to…

Programmieren

Komplexität

Aufgaben

xkcd.com/1513



Berechne $\sum_{i=1}^{n} i$

a.cpp

```
int n;
cin >> n;
int64_t s = 0;
for (int i = 1; i <= n; ++i)
  a += i;
cout << s << endl;
```

b.cpp

```
int64_t n;
cin >> n;
cout << (n*(n+1)/2) << endl;
```

Kompilieren mit Optimierung

```
g++ a.cpp -o a1
g++ a.cpp -o a2 -O2
g++ b.cpp -o b  -O2
```

Zeitmessung ($n = 2^{30} = 1{,}073{,}741{,}824$)

```
time ./a1 < sample.in
  real    0m2.505s

time ./a2 < sample.in
  real  0m0.386s

time ./b < sample.in
  real 0m0.003s
```

Fibonacci-Zahlen

Definition

- $f(0) = 0$
- $f(1) = 1$
- $f(n) = f(n-1) + f(n-2)$

Aufgabe

Eingabe: Liste von Zahlen $n_1, \ldots, n_k \in \{0, \ldots, 80\}$
Ausgabe: Liste von Zahlen $f(n_1), \ldots, f(n_k)$

Lösung zu Fibonacci?

```
long long fib(int n) {
  if (n == 0 || n == 1)
    return n;
  else
    return fib(n-1) + fib(n-2);
}

int main() {
  int n;
  while (cin >> n)
    cout << fib(n) << endl;
}
```

Immer noch Probleme:
Richtig, aber zu langsam (exponentiell)

Lösung mit linearer Laufzeit

```
long long fib(int n) {
  long long f0 = 0, f1 = 1, f2;
  for(int i = 0; i < n; ++i) {
    f2 = f0 + f1;
    f0 = f1;
    f1 = f2;
  }
  return f0;
}

int main() {
  int n;
  while (cin >> n)
    cout << fib(n) << endl;
}
```

Weitere Verbesserungsmöglichkeit:
Nicht jedes mal von vorne rechnen wenn `fib(80)` gefordert.

Vorberechnung

```
const int MAX_N = 80;  // aus Aufgabenstellung
long long fib[MAX_N+1];

void precalc() {
  fib[0] = 0;
  fib[1] = 1;
  for(int i=2; i <= MAX_N; ++i)
    fib[i] = fib[i-1] + fib[i-2];
}

int main() {
  precalc();
  int n;
  while (cin >> n)
    cout << fib[n] << endl;
}
```

Größenordnungen von Laufzeiten

Rechner können $\sim$1 Milliarde Operationen pro Sekunde ausführen.
Wenn wir $\sim$1 Sekunde Zeit haben:

- 1 Million ist wenig.
- (1 Million)^2 ist zu viel.
- 2^{1000} ist noch viel mehr.

Operation auf Array	**Laufzeit**	**Max.** n
Maximum finden	$\mathcal{O}(n)$	100 Millionen
Mergesort	$\mathcal{O}(n \log n)$	1 Million
Bubblesort	$\mathcal{O}(n^2)$	10000
Alle Permutationen aufzählen	$\mathcal{O}(n!)$	11 ($11! \approx 40$ Millionen)

Überblick

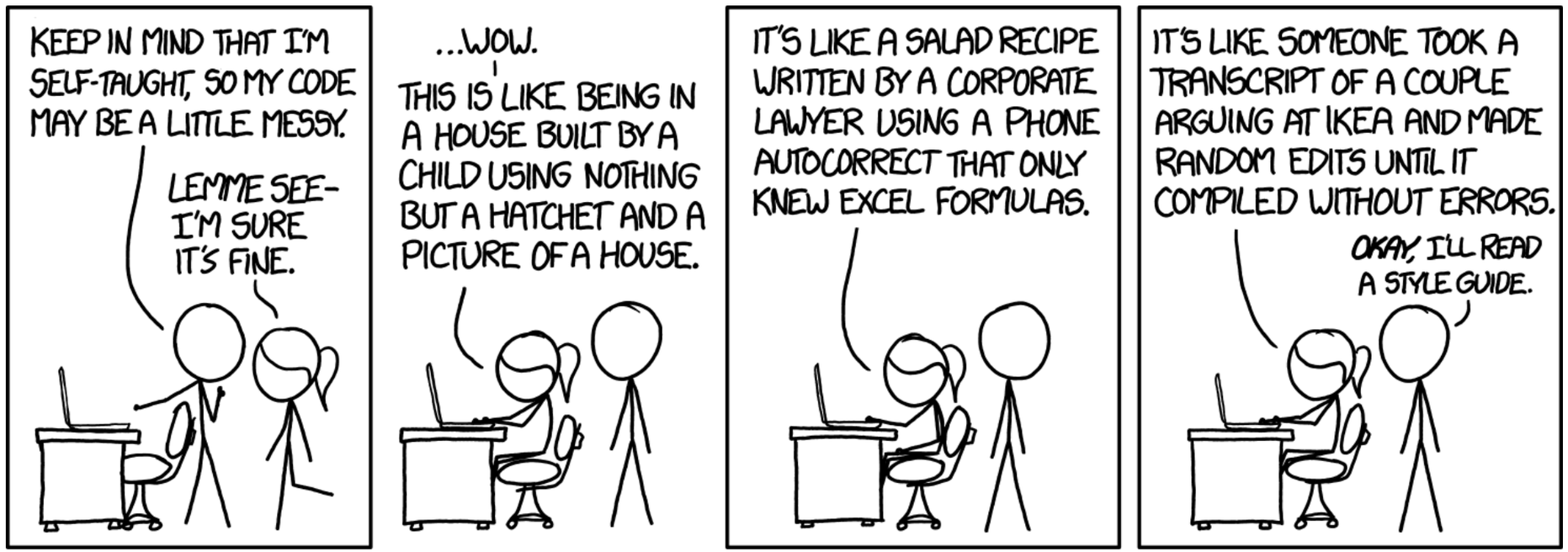
How to…

Programmieren

Komplexität

Aufgaben

xkcd.com/1513



Aufgaben

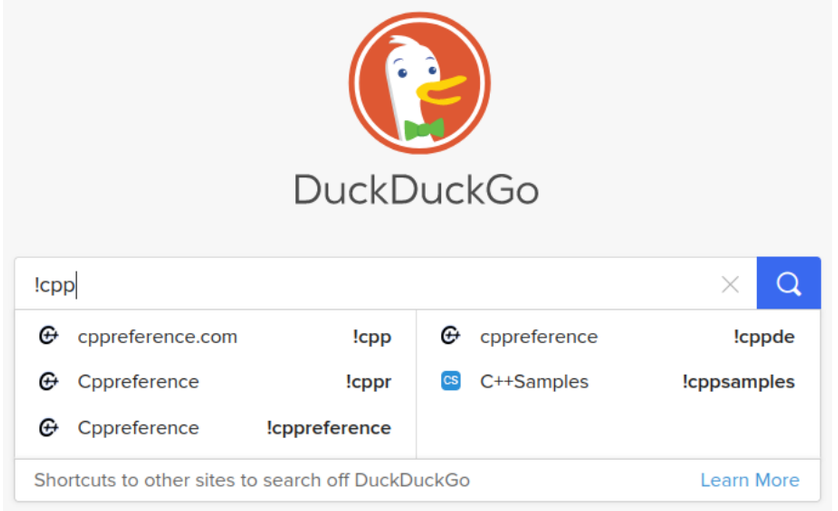
Folgende Aufgaben können Sie schon lösen:

- Hello World
- Palindromic Bracket Sequence
- Event Planning
- Spiralized Matrix
- Photographer

Aufgabenstellungen und diese Präsentation sind auf Moodle

Zum Nachschlagen:

- `http://www.cppreference.com`
- `http://www.cplusplus.com`



Cheatsheet

```cpp
// einmal mit allem bitte
#include <bits/stdc++.h>
// wer will schon immer std:: schreiben
using namespace std;
// schreibfaul aber wissen was drin ist
typedef int32_t i32;
typedef int64_t i64;
typedef uint32_t u32;
typedef uint64_t u64;
// besser ist das
#define float fliesskommazahlensindboese
#define double einfachnichtbenutzen
// und los
int main() { ... }
```

```
g++ -std=gnu++17 -Og -g -Wall -Wextra -Wconversion\
-fsanitize=address -fsanitize=undefined task.cpp \
&& cat sample.in | ./a.out | diff sample.out -
```

...oder via CLion!

Cheatsheet: `https://algo.uni-trier.de/lectures/apw/icpc.html`