Exceptions: Motivation

- Kommunikation von Fehlerfall (und Informationen dbzgl.) an aufrufenden Code
- `errno` (vom Typ `int`) trägt zu wenig Information, erfordert *immer* manuelle Behandlung im aufrufenden Code

```
int divF(int a, int b) {
  if (b == 0) {
    // ???
  }
  return a / b;
}
```

Exceptions

- Ausdruck `throw` v für beliebigen Wert v *wirft* exception
- Rückabwicklung von Funktionsaufrufen
- Sprung in *nächsten passenden* `catch`-Block
- *Passend* anhand von Typ von Vereinbarung bei `catch`
- Falls kein passender `catch`-Block: `std::terminate` ⟶ Abbruch des Programms

divF

```
int divF(int a, int b) {
  if (b == 0) {
    throw domain_error("div(_, 0)");
  }
  return a / b;
}

int main() {
  try {
    cout << divF(2, 0) << endl;
  } catch (const domain_error& e) {
    cout << e.what() << endl;
  }
}
```

Exceptions in Standardlibrary

Methode `.what()` liefert C-Zeichenkette

what

```
domain_error e
  = domain_error("Beschreibung");
cout << e.what() << endl;
```

```
Beschreibung
```

Exceptions bei Speicherreservierung

- `new`-Operator kann exceptions vom Typ `std::bad_alloc` werfen
- `bad_alloc` stammt aus Header-Datei `<new>`

Beispiel: Exceptions bei Speicherreservierung

dyn_bad_alloc.cpp

```cpp
#include <iostream>
#include <new>
#include <cstddef>

using namespace std;

int main() {
  size_t n;
  cout << "n: "; cin >> n;

  double* a;
  try {
    a = new double[n];

    a[0] = 3.14;
    cout << a[0] << endl;
  } catch (const bad_alloc& e) {
    cerr << "Zu wenig Speicherplatz fuer a vorhanden: "
         << e.what() << endl;
    return 1;
  }

  return 0;
}
```

```
n: 100000000000
Zu wenig Speicherplatz fuer a vorhanden: std::bad_alloc
```

Bereichsüberprüfter Elementen-Zugriff

- Für v vom Typ `vector<`T`>` liefert v`.at(`n`)` Referenz auf Element von v mit Index n
- Wirft exceptions vom Typ `out_of_range` wenn Index unzulässig

Beispiel: Bereichsüberprüfter Elementen-Zugriff

vector_exc.cpp

```cpp
#include <iostream>
#include <new>
#include <vector>

using namespace std;

int main() {
  vector<double>::size_type n;
  cout << "n: "; cin >> n;

  cout << "Beginn" << endl;
  try {
    cout << "Vor Vereinbarung" << endl;
    vector<double> a(n);
    cout << "Nach Vereinbarung" << endl;
    a.at(n) = 1;
    cout << "Nach Zugriff" << endl;
  } catch (const bad_alloc& e) {
    cerr << "Zu wenig Speicherplatz fuer a vorhanden: " << endl
         << "  " << e.what() << endl;
  } catch (const out_of_range& e) {
    cerr << "Zugriff ausserhalb von Grenzen:" << endl
         << "  " << e.what() << endl;
  } catch (...) {
    cerr << "Andere exception" << endl;
    throw;
  }
  cout << "Ende" << endl;
  return 0;
}
```

```
n: 100000000000
Beginn
Vor Vereinbarung
Zu wenig Speicherplatz fuer a vorhanden:
  std::bad_alloc
Ende
```

```
n: 10
Beginn
Vor Vereinbarung
Nach Vereinbarung
Zugriff ausserhalb von Grenzen:
  vector::_M_range_check: __n (which is 10) >= this->size()
 ↪  (which is 10)
Ende
```

Exceptions bei Ein-/Ausgabe

- Für Strom s, Bitmaske m, Methode `s.exceptions(m)`
- Wenn bit in Stromzustand von s gesetzt wird, das auch in m gesetzt ist ⟶ exception
- exceptions vom Typ `ios::failure`

Beispiel: Exceptions bei Ein-/Ausgabe

iostream_exc.cpp

```cpp
#include <iostream>

using namespace std;

int main() {
  cin.exceptions(ios::failbit|ios::badbit);

  try {
    int n;
    cout << "n: "; cin >> n;
    cout << "n: " << n << endl;
  } catch (const ios::failure& e) {
    cerr << "I/O-exception: "
         << e.what() << endl;
  }

  return 0;
}
```

```
n: abc
I/O-exception: basic_ios::clear: iostream error
```