Teil 8
Fehlerbehandlung

- Exceptions behandeln
- Exception-Hierarchie
- Exceptions werfen
- eigene Exceptionklassen

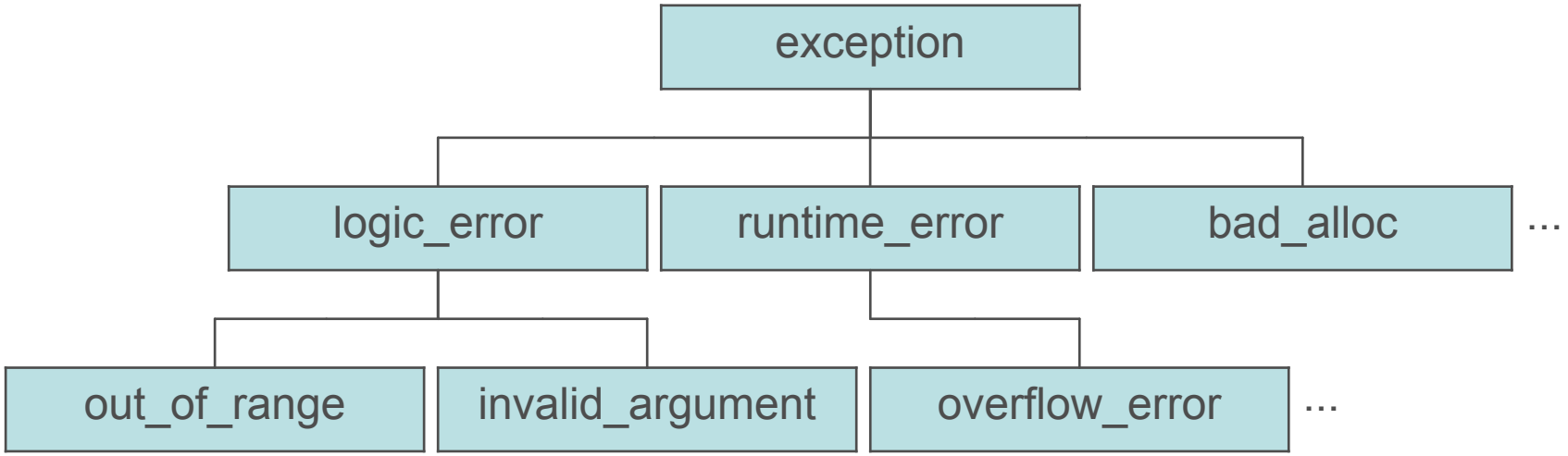
Exceptions behandeln

```
#include <stdexcept>

try
{
  std::vector<int> list = {1, 2, 3};
  int element = list.at(3);
}
catch (const std::out_of_range& e)
{
  std::cerr << e.what() << endl;
}
```

- im Try-Block der potentiell problematische Code
- Exceptions immer als Referenz „fangen“
- Exceptions, die nicht gefangen werden, propagieren in den übergeordneten Catch-Block

Exception-Hierarchie



- Standard-Exceptions sind letztlich alle von `std::exception` abgeleitet
- dadurch lassen sich Exceptions speziell oder allgemein behandeln

Exceptions behandeln

```
#include <stdexcept>

try
{
  std::vector<int> list = {1, 2, 3};
  return list.at(3);
}
catch (std::out_of_range& e)
{
  std::cerr << e.what() << endl;
  return -1;
}
catch (std::exception& e)
{
  std::cerr << e.what() << endl;
  throw;
}
```

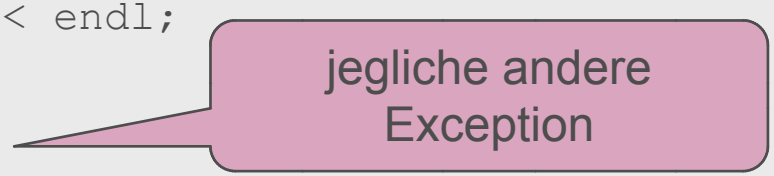
spezielle Exception

allgemeinere Exception

weitergeben an den nächsten Handler

Exceptions behandeln

```
while (true)
{
  try
  {
    proceed();
  }
  catch (std::exception& e)
  {
    std::cerr << e.what() << endl;
  }
  catch (...)
  {
    std::cerr << "unbekannter Fehler" << endl;
  }
}
```



Exceptions werfen

```
#include <stdexcept>

bool isDividable(int a, int b)
{
  if (b == 0)
  {
    throw std::invalid_argument("Divisor darf nicht null sein");
  }
  int r = a % b;
  return r == 0;
}
```

- Exceptions als Werte „werfen“ (kein new)
- semantisch richtigen Exception-Typ wählen

eigene Exceptionklassen

```
#include <stdexcept>

class InvalidDivisor : public std::invalid_argument
{
public:
  InvalidDivisor() :
    std::invalid_argument("Divisor darf nicht null sein")
  {
  }
}

bool isDividable(int a, int b)
{
  if (b == 0) throw InvalidDivisor();
  int r = a % b;
  return r == 0;
}
```

nicht unbedingt notwendig

Aufgaben

Aufgabe 22A

- Fehlerbehandlung in die vorausgegangenen Programme einbauen, z.B. beim Berechnen der Fakultät negativer Zahlen eine Exception vom Typ std::invalid_argument werfen

Aufgabe 22B

- Fehlerbehandlung wie in Aufgabe A einbauen, dabei eine eigene Exception-Klasse verwenden (sinnvoll abgeleitet)