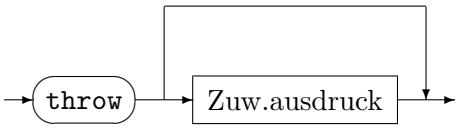
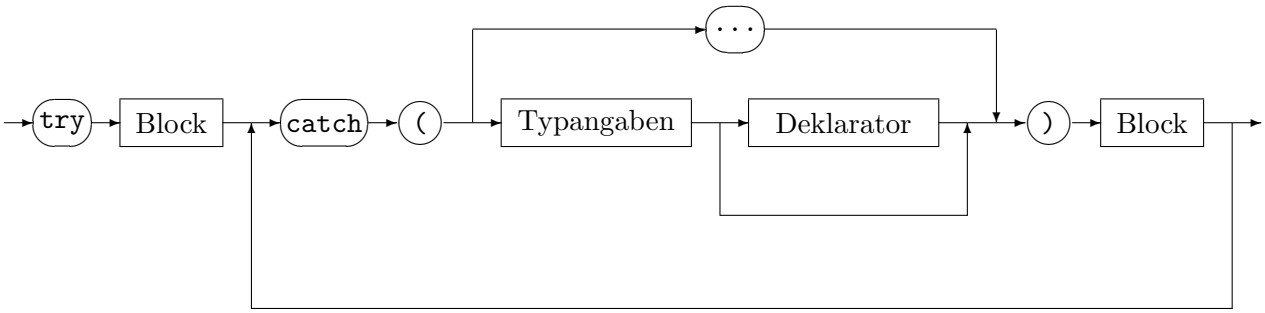
Ausnahmebehandlung (`<exception>`, `<stdexcept>`)

Eine Ausnahme (Exception) kann mit einem `throw`-Ausdruck erzeugt werden. Ihr Auftreten führt zum Aufruf der Funktion `terminate()` (Voreinstellung: Programmabbruch), wenn sie nicht mit einem `try`/`catch`-Block abgefangen wird.

`throw`-Ausdruck :



Ausnahmeanweisung :



Beim Auftreten einer Ausnahme im `try`-Block verzweigt der Programmablauf in den ersten `catch`-Block mit einer passenden Vereinbarung. Falls nicht vorhanden, wird die Suche in den `catch`-Blöcken des kleinsten dynamisch umfassenden `try`-Blocks fortgesetzt.
"`...`" in der `catch`-Vereinbarung steht für alle Ausnahmen und darf nur am Ende der Ausnahmeanweisung auftreten.

Bsp.: Dynamische Speicherreservierung (s.o.) mit Ausnahmebehandlung

```
#include <iostream>
#include <new>

using namespace std;

int main()
{
  long n; cout << "n: "; cin >> n;
  double *a;

  try {
    a = new double[n];
  }
  catch(bad_alloc) {
    cerr << "Zu wenig Speicherplatz fuer a vorhanden" << endl;
    return 1;
  }

  return 0;
}
```

Ausgabe:

```
n: 10000000000
Zu wenig Speicherplatz fuer a vorhanden
```

Wird in der `catch`-Klammer eine Variable vom Typ einer Exception-Klasse der Standardbibliothek vereinbart, so liefert die Komponentenfunktion `what()` eine Fehlermeldung.

Bsp.:

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

int main()
{
  try {
    long n;
    cout << "-- Beginn --" << endl;
    cout << "n: "; cin >> n;
    vector<double> a(n);
    cout << "-- Position 1 --" << endl;
    a.at(n) = 1;     // Fehler
    cout << "-- Position 2 --" << endl;
  }
  catch(out_of_range ausnahme) {
    cerr << "Ausnahme out_of_range entdeckt" << endl;
    cerr << "Meldung: " << ausnahme.what() << endl;
  }
  catch(bad_alloc ausnahme) {
    cerr << "Ausnahme bad_alloc entdeckt" << endl;
    cerr << "Meldung: " << ausnahme.what() << endl;
  }
  catch(...) {
    cerr << "Andere Ausnahme entdeckt" << endl;
  }
  cout << "-- Ende --" << endl;

  return 0;
}
```

Ausgabe: g++-9.4 unter Ubuntu Linux 20.04 amd64

```
-- Beginn --
n: 10000000000
Ausnahme bad_alloc entdeckt
Meldung: std::bad_alloc
-- Ende --

-- Beginn --
n: 1000
-- Position 1 --
Ausnahme out_of_range entdeckt
Meldung: vector::_M_range_check: __n (which is 1000) >= this->size() (which is 1000)
-- Ende --
```

Beispiel: Rationale Arithmetik mit Ausnahmebehandlung

```
#include <iostream>
#include <iomanip>
#include <stdexcept>                                          // Standardexceptions
#include <sstream>
#include <algorithm>

using namespace std;

class rational {

  private:
    long p,q;

    void normalisieren() {
      // Kuerzen, Nenner positiv, 0/1 als Darst. von 0
      long teiler=ggt(abs(p),abs(q));
      if (teiler>1) { p/=teiler; q/=teiler; }
      if (q<0) { p=-p; q=-q; }
      if (p==0) q=1;
    }

    static long ggt(long a,long b) {
      long r;
      if (a==0 && b==0) throw domain_error("ggt(0,0)");       // throw-Anweisung
      if (b==0) swap(a,b);
      do {
        r=a%b; a=b; b=r;
      } while (r!=0);
      return a;
    }

  public:
    rational(long p_=0, long q_=1) : p(p_), q(q_) {
      if (q==0) {
        ostringstream os;
        os << "Unerlaubter Aufruf rational(" << p << "," << q << ")";
        throw domain_error(os.str());                         // throw-Anweisung
      }
      normalisieren();
    }

  friend ostream& operator<<(ostream& stream, const rational& r) {
    if (r.q==1)
      stream << r.p;
    else
      stream << r.p << '/' << r.q;
    return stream;
  }
```

```
  friend istream& operator>>(istream& stream, rational& r) {
    char c;
    stream >> r.p >> c >> r.q;
    if (c!='/' || r.q==0)
      { stream.setstate(ios::failbit); return stream; }
    r.normalisieren();
    return stream;
  }

  friend rational operator/(rational s, rational t) {
    if (t.p==0) {
      ostringstream os;
      os << "Unerlaubte Operation: " << s << " / " << t;
      throw domain_error(os.str());                          // throw-Anweisung
    }
    return rational(s.p*t.q,s.q*t.p);
  }
};

int main() try                                               // Funktionstryblock
{
  cin.exceptions(ios::failbit|ios::badbit);                  // Exceptions fuer
                                                             // Eingabe aktivieren
  rational r,s;
  cout << "p1/q1 p2/q2: ";
  cin  >> r >> s;
  cout << "r/s = " << r/s << endl;
  return 0;
}
catch (domain_error ausnahme)                                // Catch-Bloecke
{
  cerr << "--- Ausnahme entdeckt: " << ausnahme.what() << endl;
}
catch (ios::failure ausnahme)
{
  cerr << "--- Ausnahme entdeckt: Eingabefehler" << endl;
  cerr << ausnahme.what() << endl;
}
catch (...)
{
  cerr << "--- Sonstige Ausnahme entdeckt!" << endl;
}
```

Ausgabe:

```
p1/q1 p2/q2: 1/2 3/4
r/s = 2/3

p1/q1 p2/q2: 1/0 3/4
--- Ausnahme entdeckt: Eingabefehler
basic_ios::clear: iostream error

p1/q1 p2/q2: 1/2 0/4
--- Ausnahme entdeckt: Unerlaubte Operation: 1/2 / 0
```