# Ausnahmebehandlung

# (exceptionhandling)

# in C++

Diese Folien basieren auf *Kapitel 19* in *Herbert Schildt, C++ ENT-PACKT, mitp*.
Weiterführende Informationen zu diesem Themengebiet sind insbesondere dort zu finden.

# Ausnahmebehandlung

- basiert auf den Schlüsselwörtern `try`, `catch` und `throw`
- zu überwachender Code muss innerhalb eines `try`-Blocks ausgeführt werden
- auch Funktionen die von einem `try`-Block aufgerufen werden, können Ausnahmen erzeugen
- Ausnahmen, die von überwachtem Code ausgelöst werden, werden von einer catch-Anweisung abgefangen, die direkt der `try`-Anweisung folgt.

Allgemeine Form von `try` und `catch`

```
try
{
   // try-Block
}
catch(type1 arg)
{
   // catch-Block
}
catch(type2 arg)
{
   // catch-Block
}
...
```

Allgemeine Form von `throw`:

```
throw exception;
```

# Ausnahmebehandlung, einfaches Beispiel

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Start" << endl;
    try{ // beginne try-Block
        cout << "In try-Block" << endl;
        throw 100;
        cout << "Das wird nicht ausgeführt";
    }
    catch(int i){ // fange Ausnahme ab
        cout << "Ausnahme abgefangen, Wert = " << i << endl;
    }
    cout << "Ende";
    return 0;
}
```

Programmausgabe:

```
Start
In try-Block
Ausnahme abgefangen, Wert = 100
Ende
```

# Ausnahmebehandlung, einfaches Beispiel, falscher Typ

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Start" << endl;
    try{ // beginne try-Block
        cout << "In try-Block" << endl;
        throw 100;
        cout << "Das wird nicht ausgeführt";
    }
    catch(double i){ // fängt Ausnahme nicht ab
        cout << "Ausnahme abgefangen, Wert = " << i << endl;
    }
    cout << "Ende";
    return 0;
}
```

Programmausgabe:

```
Start
In try-Block
Abnormal program termination
```

# Ausnahmebehandlung

Eine Ausnahme kann außerhalb eines `try`-Blocks ausgelöst werden, solange sie von einer Funktion ausgelöst wird, die innerhalb eines `try`-Blocks aufgerufen wird.

```
...
void Xtest(int test)
{
    cout << "In Xtest, test = " << test << endl;
    if (test) throw test;
}
int main()
{
    cout << Start << endl;
    try{ // beginne try-Block
        cout << "In try-Block" << endl;
        Xtest(0);
        Xtest(1);
        Xtest(2);
    }
    catch(int i){ //fange Fehler ab
        cout << "Ausnahme abgefangen, Wert = ";
        cout << i << endl;
    }
    cout << "Ende";
    return 0;
}
```

Programmausgabe:

```
Start
In try-Block
In Xtest, test = 0
In Xtest, test = 1
Ausnahme abgefangen, Wert = 1
Ende
```

# Ausnahmebehandlung

Ein `try`-Block kann lokal in einer Funktion sein. Die Ausnahmebehandlung dieser Funktion wird dann bei jedem Aufruf zurückgesetzt.

```
...
void Xhandler(int test)
{
    try{
        if(test) throw test;
    }
    catch(int i){
        cout << "Ausnahme #: " << i << endl;
    }
}
int main()
{
    cout << "Start" << endl;
    Xhandler(1);
    Xhandler(2);
    Xhandler(0);
    Xhandler(3);
    cout << "Ende";
    return 0;
}
```

Programmausgabe:

```
Start
Ausnahme #: 1
Ausnahme #: 2
Ausnahme #: 3
Ende
```

# Ausnahmebehandlung

Eine `catch`-Anweisung wird nur ausgeführt, wenn eine Ausnahme abgefangen wird. Sonst wird der gesamte `catch`-Block übergangen

```
...
int main()
{
    cout << "Start" << endl;
    try{
        cout << "In try-Block" << endl;
        cout << "In try-Block 2" << endl;
    }
    catch(int i){
        cout << "Ausnahme " << i << endl;
    }
    cout << "Ende";
    return 0;
}
```

Programmausgabe:

```
Start
In try-Block
In try-Block 2
Ende
```

# Ausnahmebehandlung, Klassentypen

Eine Ausnahme kann von einem beliebigen Typ sein, also auch selbstgeschriebene Klassen.

```
...
class MyException{
public:
    char str_what[80];
    int what;

    MyException(){
        *str_what = 0; what = 0;}
    MyException(char* s, int e){
        strcpy(str_what, s);
        what = e;
    }
};
int main()
{
    int i;
    try{
        cout << "Positive Zahl: ";
        cin >> i;
        if (i<0) throw MyException("Nicht positiv",i);
    }
    catch(MyException e){
        cout << e.str_what << ": " << e.what << endl;
    }
    return 0;
}
```

Programmausgabe (Beispiel)

```
Positive Zahl: -4
Nicht positiv: -4
```

# Ausnahmebehandlung, mehrere `catch`-Anweisungen

Jedes `catch` muss einen unterschiedlichen Typ von Ausnahme besitzen.

```
...
void Xhandler(int test)
{
    try{
        if (test) throw test;
        else throw "Wert ist null";
    }
    catch(int i){
        cout << "Ausnahme #: " << i << endl;
    }
    catch(const char* str){
        cout << str << endl;
    }
}
int main()
{
    cout << "Start" << endl;
    Xhandler(1);
    Xhandler(2);
    Xhandler(0);
    Xhandler(3);
    cout << "Ende";
    return 0;
}
```

Programmausgabe

```
Start
Ausnahme #: 1
Ausnahme #: 2
Wert ist null
Ausnahme #: 3
Ende
```

# Ausnahmebehandlung, abgeleitete Klassen

Zum Abfangen einer Basisklasse und einer abgeleiteten Klasse, muss die `catch`-Anweisung der abgeleiteten Klasse vor der `catch`-Anweisung der Basisklasse stehen. Sonst fängt die `catch`-Anweisung der Basisklasse auch alle abgeleitete Klassen ab.

```
...
class B{
};

class D: public B{
};

int main()
{
    D derived;

    try{
        throw derived;
    }
    catch(B b){
        cout << "Basisklasse abgefangen" << endl;
    }
    catch(D d){
        cout << "Abgeleitete Klasse abgefangen" << endl;
    }
    return 0;
}
```

Programmausgabe

```
Basisklasse abgefangen
```

# Ausnahmebehandlung, abgeleitete Klassen

Zum Abfangen einer Basisklasse und einer abgeleiteten Klasse, muss die `catch`-Anweisung der abgeleiteten Klasse vor der `catch`-Anweisung der Basisklasse stehen. Sonst fängt die `catch`-Anweisung der Basisklasse auch alle abgeleitete Klassen ab.

```
...
class B{
};

class D: public B{
};

int main()
{
    D derived;

    try{
        throw derived;
    }
    catch(D d){
        cout << "Abgeleitete Klasse abgefangen" << endl;
    }
    catch(B b){
        cout << "Basisklasse abgefangen" << endl;
    }
    return 0;
}
```

Programmausgabe

```
Abgeleitete Klasse abgefangen
```

# Ausnahmebehandlung, alle Ausnahmen abfangen

Form:

```
    catch(...){
        // behandele alle Ausnahmen
    }
```

```
...
void Xhandler(int test)
{
    try{
        if (test==0) throw test; //löse int aus
        if (test==1) throw 'a'; //löse char aus
        if (test==2) throw 123.23; //löse double aus
    }
    catch(...){ //fange alle Ausnahmen ab
        cout << "Ausnahme abgefangen" << endl;
    }
}
int main()
{
    cout << "Start" << endl;
    Xhandler(0);
    Xhandler(1);
    Xhandler(2);
    cout << "Ende" << endl;
    return 0;
}
```

Programmausgabe

```
Start
Ausnahme abgefangen
Ausnahme abgefangen
Ausnahme abgefangen
Ende
```

# Ausnahmebehandlung, alle Ausnahmen abfangen

`catch(...)` ist z.B. gut als letztes `catch` in der Reihe als default-`catch`, das alle abfängt

```
...
void Xhandler(int test)
{
    try{
        if (test==0) throw test; //löse int aus
        if (test==1) throw 'a'; //löse char aus
        if (test==2) throw 123.23; //löse double aus
    }
    catch(int i){ // fange int-Ausnahme ab
        cout << "Integer abgefangen" << endl;
    }
    catch(...){ //fange alle anderen Ausnahmen ab
        cout << "Ausnahme abgefangen" << endl;
    }
}
int main()
{
    cout << "Start" << endl;
    Xhandler(0);
    Xhandler(1);
    Xhandler(2);
    cout << "Ende" << endl;
    return 0;
}
```

Programmausgabe

```
Start
Integer abgefangen
Ausnahme abgefangen
Ausnahme abgefangen
Ende
```

# Ausnahmebehandlung, Ausnahmen beschränken

Der Typ der Ausnahme, die eine Funktion nach außen hin auslösen kann, kann beschränkt werden. Zur Funktionsdefinition muss dazu eine `throw()`-Klausel hinzugefügt werden.

```
...
// diese Funktion kann nur int, char und double
// als Ausnahme auslösen
void Xhandler(int test) throw(int, char, double)
{
    if (test==0) throw test; //löse int aus
    if (test==1) throw 'a'; //löse char aus
    if (test==2) throw 123.23; //löse double aus
}
int main()
{
    try{
        Xhandler(0);
    }
    catch(int i){
        cout << "int abgefangen" << endl;
    }
    catch(char c){
        cout << "char abgefangen" << endl;
    }
    catch(double d){
        cout << "double abgefangen" << endl;
    }
    ...
}
```

Programmausgabe

```
int abgefangen
```

# Ausnahmebehandlung, Ausnahmen beschränken

Der Typ der Ausnahme, die eine Funktion nach außen hin auslösen kann, kann beschränkt werden. Zur Funktionsdefinition muss dazu eine `throw()`-Klausel hinzugefügt werden.

```
...
// diese Funktion kann nur noch char und double
// als Ausnahme auslösen
void Xhandler(int test) throw(char, double)
{
    if (test==0) throw test; //löse int aus
    if (test==1) throw 'a'; //löse char aus
    if (test==2) throw 123.23; //löse double aus
}
int main()
{
    try{
        Xhandler(0);
    }
    catch(int i){
        cout << "int abgefangen" << endl;
    }
    catch(char c){
        cout << "char abgefangen" << endl;
    }
    catch(double d){
        cout << "double abgefangen" << endl;
    }
    ...
}
```

Programmausgabe

```
Abnormal program termination
```

# Ausnahmebehandlung, Ausnahmen beschränken

Wichtig: Eine Funktion kann nur einschränken, welche Ausnahmen an den `try`-Block übergeben werden, von wo aus die Funktion aufgerufen wird. Ein `try`-Block *innerhalb* einer Funktion kann jeden Typ von Ausnahme auslösen, solange die Ausnahme *innerhalb* der Funktion abgefangen wird. Diese Einschränkung gilt nur dann, wenn eine ausgelöste Ausnahme die Funktion verlässt.

```
// kann keine Ausnahmen mehr auslösen,
// Programm wird fehlerhaft beendet
void Xhandler(int test) throw()
{
    if (test==0) throw test;
    if (test==1) throw 'a';
    if (test==2) throw 123.23;
}
```

```
// ist korrekt
void Xhandler(int test) throw()
{
    try{
        if (test==0) throw test;
        if (test==1) throw 'a';
        if (test==2) throw 123.23;
    }
    catch(int i){
        ...
    }
    catch(char c){
        ...
    }
    catch(double d){
        ...
    }
}
```

# Ausnahmebehandlung, Ausnahmen erneut auslösen

Wenn man von einer Ausnahmebehandlung aus eine Ausnahme erneut auslösen möchte, wird einfach nur `throw` aufgerufen ohne Angabe einer Ausnahme.

```
...
void Xhandler()
{
    try{
        throw "hello"; // löse char* aus
    }
    catch(const char*){ // fange char* ab
        cout << "char* in Xhandler abgefangen" << endl;
        throw; // löse char* außerhalb von Funktion aus
    }
}
int main()
{
    cout << "Start" << endl;
    try{
        Xhandler();
    }
    catch(const char*){
        cout << "char* in main abgefangen" << endl;
    }
    cout << "Ende";
    return 0;
}
```

Programmausgabe

```
Start
char* in Xhandler abgefangen
char* in main abgefangen
Ende
```

# Ausnahmebehandlung, `terminate()` und `unexpected()`

- Tritt während der Ausnahmebehandlung ein Fehler auf, werden `terminate()` und `unexpected()` aufgerufen.
- Werden von der Standardbibliothek von C++ bereitgestellt.
- `terminate()` und `unexpected()` benötigen den Header `<exception>`.

`terminate()` ist ein Handler, der alle Ausnahmen entgegennimmt, die von keinem anderen Handler behandelt wurden, also auch in der Ausnahmebehandlung selbst. Z.B. wird `terminate()` aufgerufen, wenn keine passende `catch`-Anweisung für die ausgelöste Ausnahme gefunden wird. Defaultmäßig ruft `terminate()`die Funktion `abort()` auf.

`unexpected()` wird aufgerufen, wenn eine Funktion versucht, eine Ausnahme auszulösen, deren Typ durch die `throw`-Liste nicht erlaubt ist. Defaultmäßig ruft die Funktion `terminate()` auf.

# Ausnahmebehandlung, Setzen der Handler

Über die Funktionen `set_terminate` und `set_unexpected` in `<exception>` können an Stelle von `terminate` und `unexpected` eigene Handler eingesetzt werden. Dadurch kann ein Programm die vollständige Kontrolle über die Behandlung von Ausnahmen übernehmen.

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

void my_Thandler(){
    cout << "In neuem terminate-Handler << endl;
    abort();
}

int main()
{
    // setze neuen terminate-Handler
    set_terminate(my_Thandler);
    try{
        cout << "In try-Block" << endl;
        throw 100; // erzeuge eine Ausnahme
    }
    catch(double i){ //fängt keine int-Ausnahme ab
        // ...
    }
    return 0;
}
```

## Programmausgabe

```
In try-Block
In neuem terminate-Handler
Abnormal program termination
```

# Ausnahmebehandlung, Ausnahmebehandlung anwenden

- Programm kann "nicht normale" Ereignisse auf eine strukturierte Art behandeln
- Handler der Fehlerbehandlung sollte etwas Sinnvolles tun
- Besonders nützlich für den Rücksprung aus tief verschachtelten Routinen, wenn ein katastrophaler Fehler aufgetreten ist
- **Nachteil**: **macht den Code *spürbar* langsamer**

## Beispiel einer Fehlerbehandlung

```
...
void divide(double a, double b)
{
    try{
        if (!b) throw b; //überprüfen auf null
        cout << "Ergebnis: " << a/b << endl;
    }
    catch(double b){
        cout << "Kann nicht durch null teilen." << endl;
    }
}
```