



FH MÜNSTER
University of Applied Sciences

Programmieren in C++

Teil 9 – Fehlerbehandlung

Prof. Dr. Kathrin Ungru
Fachbereich Elektrotechnik und Informatik

kathrin.ungru@fh-muenster.de

Fehlerbehandlung

Inhalt

- Strategien zur Fehlerbehandlung
- Fehler abfangen
- Exception-Klassen der C++ Standardbibliothek



Fehlerbehandlung

Schlüsselwörter in diesem Kapitel

| | | | | | |
|--------------|------------|--------------|------------------|---------------|----------|
| alignas | const | dynamic_cast | long | short | typedef |
| alignof | constexpr | else | mutable | signed | typeid |
| asm | constexpr | enum | namespace | sizeof | typename |
| auto | constinit | explicit | new | static | union |
| bool | const_cast | export | noexcept | static_assert | unsigned |
| break | continue | extern | nullptr | static_cast | using |
| case | co_await | false | operator | struct | virtual |
| catch | co_return | float | private | switch | void |
| char | co_yield | for | protected | template | volatile |
| char8_t | decltype | friend | public | this | wchar_t |
| char16_t | default | goto | register | thread_local | while |
| char32_t | delete | if | reinterpret_cast | throw | |
| class | do | inline | requires | true | |
| concept | double | int | return | try | |

Fehlerbehandlung

Beispiel

```
double div(double a, double b)
{
    return a / b;
}
```

Division durch 0?

```
double zeichne(Form* form)
{
    form->zeichnen()
}
```

Form initialisiert?

Fehlerbehandlung

Voraussetzungen und Anwendung

- Bei der Fehlerbehandlung geht es im allgemeinen um erwartete Fehler.
- Ein Fehler, der in einer Funktion auftritt, kann nicht in dieser Funktion behoben werden.
- Nutzende der Funktion sollen Fehlermeldung erhalten, um:
 - den Fehler abzufangen
 - weiterzuleiten
- Art der Fehlerbehandlung auch abhängig davon, wie sicherheitskritisch der Einsatz der Software ist.

Fehlerbehandlung

Voraussetzungen und Anwendung


- **Erkennung von Fehlern**, z.B.:
 - Division durch Null
 - Bereichsüberschreitung eines Arrays
 - Syntaxfehler bei Eingaben
 - Zugriff auf eine nichtgeöffnete Datei
 - Fehlschlag der Speicherbeschaffung
 - Nichteinhaltung der Vorbedingung einer Funktion
- **Behandlung von Fehlern**
 - Schwieriger als die Erkennung
 - Sehr abhängig vom Fehlerfall

Fehlerbehandlung

Bisherige Möglichkeiten

- **Programmabbruch:**
 - wenig benutzerfreundlich
 - u.U. keine Rückmeldung über Art des Fehlers

```
double div(double a, double b) {  
    if (0. == b) {  
        exit(EXIT_FAILURE);  
    }  
    return a / b;  
}
```




Fehlerbehandlung

Bisherige Möglichkeiten

- **Kopf-In-den-Sand-Methode:** Trotz Fehler wird ein Wert zurückgegeben. Besonders problematisch, da
 - eine Fehleranalyse schwer werden kann.
 - es zu falschen Ergebnissen kommen kann.

```
double div(double a, double b) {  
    if (0. == b) {  
        return 0.;  
    }  
    return a / b;  
}
```



Fehlerbehandlung

Bisherige Möglichkeiten

- **Fehlerbehandlung über Rückgabewert einer Funktion:**
 - **Vorteil** differenziertere Betrachtung möglich, Programm läuft weiter
 - **Nachteil**
 - wenig übersichtlich, schlechte Lesbarkeit
 - Benutzende der Funktion könnten sich das Abfangen des Fehlers aus Bequemlichkeit sparen
- **Ähnliche Methode:**
 - Funktion wird aufgerufen, die die Fehlerbehandlung durchführt

```
int div(double a, double b, double &res) {  
    if (0. == b) {  
        return -1;  
    }  
    res = a / b;  
    return 0;  
}
```

Fehlerbehandlung

C++ Ausnahmebehandlung

- Von den vorgestellten Methoden ist die Methode den Fehler über einen Parameter der Funktion zurückzugeben am besten. (Bspl. `printf()` in C)
- **Aber noch besser:**
 - Ausnahmebehandlung (exception handling) von C++ nutzen!**
 - Saubere Trennung von "normalem" Programmcode und Fehlerbehandlung.
 - Abfrage von Fehlerparametern an vielen Stellen nicht mehr notwendig.

C++ Ausnahmebehandlung

Ablauf

1. Funktion **versucht** eine Aufgabe zu erledigen.
2. Falls die Funktion einen Fehler feststellt, der nicht behoben werden kann, **wirft** sie eine Ausnahme.
3. Die Ausnahme wird **aufgefangen**, die den Fehler bearbeitet oder an die höhere Ebene weiterreicht.
4. Wenn die oberste Funktion `main()` den Fehler nicht bearbeiten kann wird das Programm abgebrochen.

`try`

`throw`

`catch`



C++ Ausnahmebehandlung

Ablauf

```
try{
    foo(); // throw wird in einer Funktion aufgerufen
}
catch(Datentyp e) { // Syntax für einfach Datentypen z.B. int, const char*
    // Fehlerbehandlung
}
catch(const Objekttyp &e) { // Syntax für die Übergabe von Objekten
    // Fehlerbehandlung
}
// weitere catch-Blöcke könnten hier folgen
// Nach Fehlerbehandlung wird Programm hier fortgesetzt
```

```
void foo() {
    if(einFehlerIstAufgetreten)
        throw Exception(); // throw wird bei Fehler in einer Funktion aufgerufen
}
```

C++ Ausnahmebehandlung

Nichtspezifizierte Fehler

- ... = beliebige Anzahl beliebiger Datentypen

```
catch(...) { // ... bedeutet hier nicht spezifizierter Fehler
    cerr << "nicht behebbarer Fehler!" << endl;
    throw; // Weitergabe an nächsthöhere Instanz
}
```

- Muss am Ende stehen, sonst werden spezifizierte Fehler evtl. abgefangen.

C++ Ausnahmebehandlung

Was passiert beim Aufruf von **throw** ?

- Wird in einem Block eine Ausnahme geworfen werden automatisch alle **Destruktoren** der Objekte des Blocks aufgerufen. Gilt nur für **Objekte im Stack** und somit nicht für dynamisch erzeugte Objekte im Heap.
- **Destruktoren dürfen daher keine Ausnahmen werfen.**

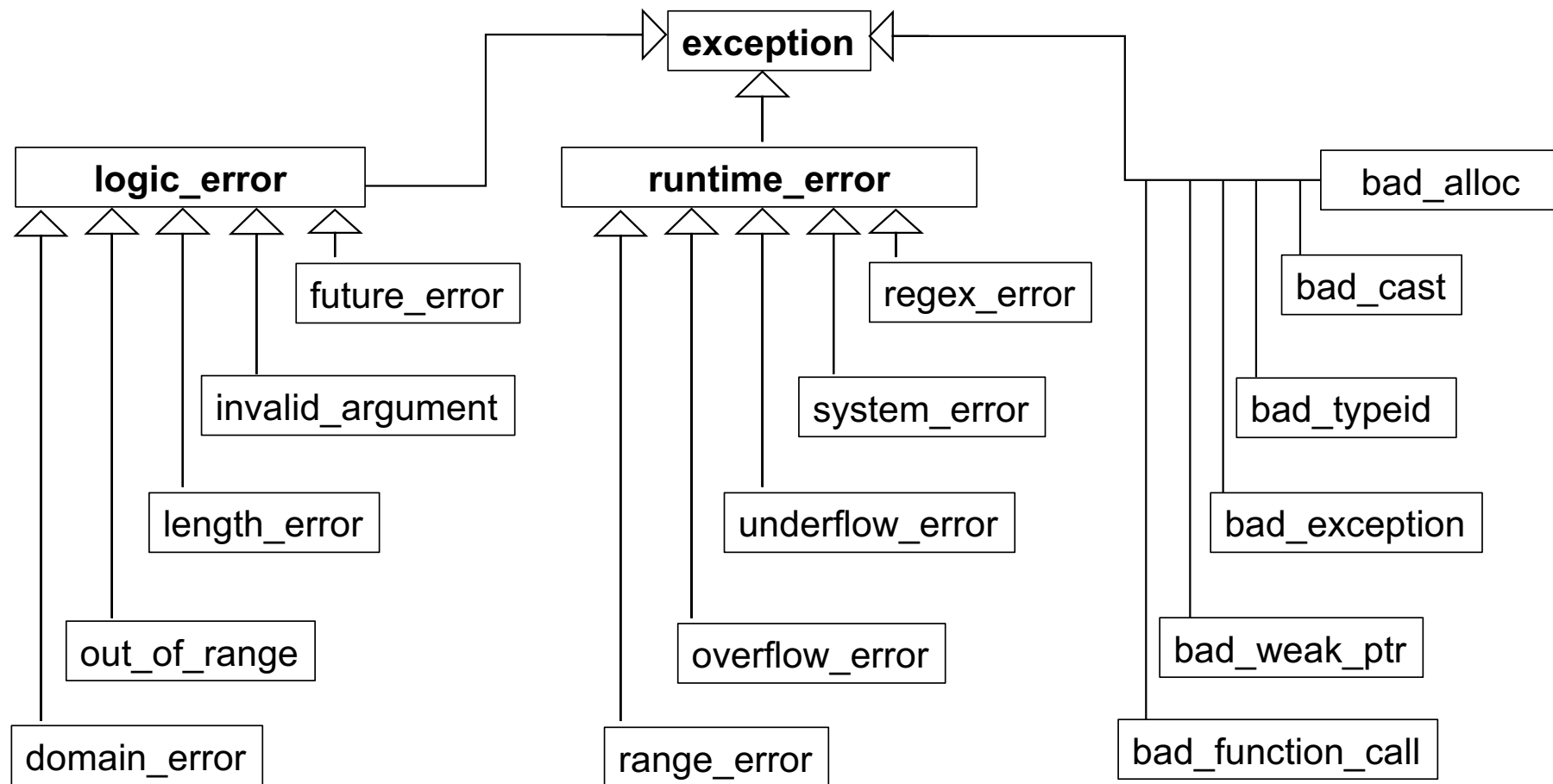
C++ Ausnahmebehandlung

Das Schlüsselwort `noexcept`

- `noexcept` dokumentiert, ob eine Funktion Ausnahmen werfen (`throw`) kann oder nicht:
- Funktion kann beliebige Ausnahmen werfen:
 - `void f();`
 - `void f() noexcept (false);`
- Funktion verspricht keine Ausnahme zu werfen:
 - `void f() noexcept;`
 - `void f() noexcept (true);`
 - Wirft `f()` trotzdem eine Ausnahme, wird Programm ausgeführt beim Aufruf von `throw` in `f()` abgebrochen.
 - Sollte daher nur in besonderen Fällen genutzt werden, denn wenn einmal die Funktionsdeklaration in der Welt ist werden spätere Code-Anpassungen (die evtl. doch Ausnahmen werfen sollen) schwierig!

Standard-Exceptions

Exception-Klassen der C++ Standardbibliothek



| Klasse | Bedeutung | Header |
|----------------------|--|--------------------------|
| exception | Basisklasse | <exception> |
| logic_error | theoretisch vermeidbare Fehler, z.B. Verletzung von logischen Vorbedingungen | <stdexcept> |
| invalid_argument | ungültiges Argument bei Funktionen | <stdexcept> |
| length_error | Fehler in Funktionen der C++ Standardbibliothek, wenn ein Objekt erzeugt werden soll, das die maximal erlaubte Größe für dieses Objekt überschreitet | <stdexcept> |
| out_of_range | Bereichsüberschreitungsfehler | <stdexcept> |
| domain_error | anderer Fehler des Anwendungsbereichs | <stdexcept> |
| future_error | für asynchrone Systemaufrufe | <future> |
| runtime_error | nicht vorhersehbare Fehler, z.B. Datenabhängige Fehler | <stdexcept> |
| regex_error | Fehler bei regulären Ausdrücken | <regex> |
| system_error | Fehlermeldung des Betriebssystems | <system_error> |
| range_error | Bereichsüberschreitung | <stdexcept> |
| overflow_error | arithmetischer Überlauf | <stdexcept> |
| underflow_error | arithmetischer Unterlauf | <stdexcept> |
| bad_alloc | Speicherzuweisungsfehler | <new> |
| bad_typeid | falscher Objekttyp | <typeinfo> |
| bad_cast | Typumwandlungsfehler | <typeinfo> |
| bad_weak_ptr | kann vom shared_ptr-Konstruktor geworfen werden | <memory> |
| bad_function_call | wird ggf. beim Aufruf eines Funktionshandlers geworfen | <functional> |

Exception Sicherheit

- Ein Programm ist Exception-sicher, wenn Laufzeitfehler keine nachteiligen Auswirkungen haben. Es können 4 Stufen definiert werden:
- **Stufe 1:** Es gibt keinerlei Zusicherung, ob und wie sich Fehler auswirken.
- **Stufe 2:** Ein Programm kann zwar falsche Daten erzeugen, es soll aber weder abstürzen noch Speicherlecks hinterlassen.
- **Stufe 3:** **Starke Exception-Sicherheit** liegt vor, wenn ein Objekt durch einen auftretenden Fehler im selben Zustand ist wie vor der fehlerhaften Operation.
 - Lässt sich meistens erreichen. Evtl. nicht immer wünschenswert.
- **Stufe 4:** **Die sicherste Stufe liegt vor**, wenn garantiert wird, dass keine Fehler auftreten oder alle auftretenden Fehler so abgefangen werden, dass diese keine nachteiligen Auswirkungen haben.
 - Schwer zu realisieren.

Exception-Sicherheit

- Prinzip: "Design bei Contract": eine Funktion gewährleistet die Nachbedingung, wenn der Aufrufer die Vorbedingung einhält.
 - z.B. sollte die Funktion zur Division zweier Zahlen überhaupt aufgerufen werden, wenn der Nenner Null ist?
- Aufwand Abschätzung:
 - Sollte eine Funktion jede verletzte Vorbedingung abfragen oder auf eine korrekte Eingabe vertrauen?
 - z.B. abzufragen, ob Null im Nenner steht geht schnell.
 - z.B. zu überprüfen, ob ein Array ungültige Werte enthält ist aufwändig.
 - Abschätzung erfolgt im Einzelfall!



FH MÜNSTER
University of Applied Sciences

Programmieren in C++

Prof. Dr. Kathrin Ungru

Fachbereich Elektrotechnik und Informatik

kathrin.ungru@fh-muenster.de