Ausnahmebehandlung

Sebastian Stabinger, Thomas Hausberger

SS2021

Was versteht man unter Ausnahmebehandlung?

- Ausnahmebehandlung (engl. Exception Handeling) beschreibt die Methoden und Systeme die man verwendet um mit Fehlern zur Laufzeit umzugehen
- Wir wollen also nicht, dass unser Programm einfach abstürzt, wir wollen auf den Fehler sinnvoll reagieren

Beispiele

- Wir haben eine Division durch Null
- Wir wollen den Durchschnitt eines Vektors berechnen, aber der Vektor ist leer
- Wir öffnen eine Datei zum Lesen, aber die Datei existiert nicht
- Wir versuchen eine Verbindung zu einem Server aufzubauen, aber der Server meldet sich nicht
- ...

Ausnahmebehandlung in C

Ausnahmebehandlung in C

- In C gibt es kein spezielles System um mit Laufzeitfehlern umzugehen
- Man muss also existierende Sprachelemente nutzen

Möglichkeiten der Fehlerbehandlung:

- Über den Rückgabewert
- Über globale Variablen
- Über einen Zeiger der als Parameter übergeben wird

Beispiel mit Rückgabewert

```c
#include <stdio.h>
#include <stdlib.h>

int do_something() {
  if (rand() % 5 == 0)
    return -1; // Ein Fehler ist aufgetreten
  else
    return 0; // Alles OK
}

int main() {
  for (int i = 0; i < 10; i++) {
    int returnvalue = do_something();
    if (returnvalue == 0)
      printf("Alles OK\n");
    else if (returnvalue == -1)
      printf("In do_something ist ein Fehler aufgetreten.\n");
  }
}
```

Beispiel mit globaler Variable

```c
#include <stdio.h>

int sumerror = 0;
int sum(int start, int stop) {
  if (start > stop) {
    sumerror = -1; // Wir haben ein Problem!
    return 0;
  }
  int sum = 0;
  for (int i = start; i <= stop; i++)
    sum += i;
  sumerror = 0; // Alles war ok
  return sum;
}

int main() {
  int s = sum(10, 20);
  if (sumerror == 0) printf("Summe ist %d\n", s);
  else printf("Fehler beim Berechnen der Summer\n");
  s = sum(30, 10);
  if (sumerror == 0) printf("Summe ist %d\n", s);
  else printf("Fehler beim Berechnen der Summer\n");
}
```

Beispiel mit Zeiger

```c
#include <stdio.h>

int sum(int start, int stop, int *error) {
    if (start > stop) {
        *error = -1; // Wir haben ein Problem!
        return 0;
    }
    int sum = 0;
    for (int i = start; i <= stop; i++)
        sum += i;
    *error = 0; // Alles war ok
    return sum;
}

int main() {
    int sumerror = 0;
    int s = sum(10, 20, &sumerror);
    if (sumerror == 0) printf("Summe ist %d\n", s);
    else printf("Fehler beim Berechnen der Summer\n");
    s = sum(30, 10, &sumerror);
    if (sumerror == 0) printf("Summe ist %d\n", s);
    else printf("Fehler beim Berechnen der Summer\n");
}
```

Vor- und Nachteile der unterschiedlichen Varianten

- Mit Rückgabewert: Funktioniert gut, wenn wir den Rückgabewert der Funktion nicht für etwas anderes benötigen, oder wenn es Rückgabewerte gibt, die normalerweise nicht vorkommen können.
- Mit globaler Variable: Kann auch verwendet werden, wenn der Rückgabewert benötigt wird. Unübersichtlich und funktioniert nicht, wenn die gleiche Funktion parallel öfter aufgerufen wird (was Heute immer häufiger ist)
- Mit Zeiger: Die sauberste Variante, aber etwas aufwändig, da man eine Variable erzeugen muss etc.

Genereller Nachteil

Wir müssen jeden einzelnen Fehler immer manuell mit einer `if`-Bedingung abfangen

Ausnahmebehandlung in C++

Überblick

Ausnahme `exception`

Ein beliebiges Objekt. Üblicherweise aber eine Instanz einer Klasse die von der Klasse `exception` abgeleitet ist.

Werfen `throw`

- Entdecken wir einen Fehler, können wir eine Ausnahme werfen (engl. *to throw an exception*)

Fangen `catch`

- Wir können für einen bestimmten Teil unseres Codes angeben was passieren soll falls während der Ausführung eine Ausnahme auftritt. Man spricht davon, dass die Ausnahme gefangen wird (engl. *to catch an exception*)
- Wir können auch angeben welche Fehler wir fangen wollen und was bei unterschiedlichen Fehlern passieren soll

Das Werfen einer Ausnahme

- Das Werfen einer Ausnahme geschieht mit dem Schlüsselwort `throw`
- `throw` funktioniert ähnlich wie `return`, wir müssen uns aber nicht auf einen Rückgabetyp festlegen
- Üblicherweise wirft man aber Klassen die von `std::exception` abgeleitet sind

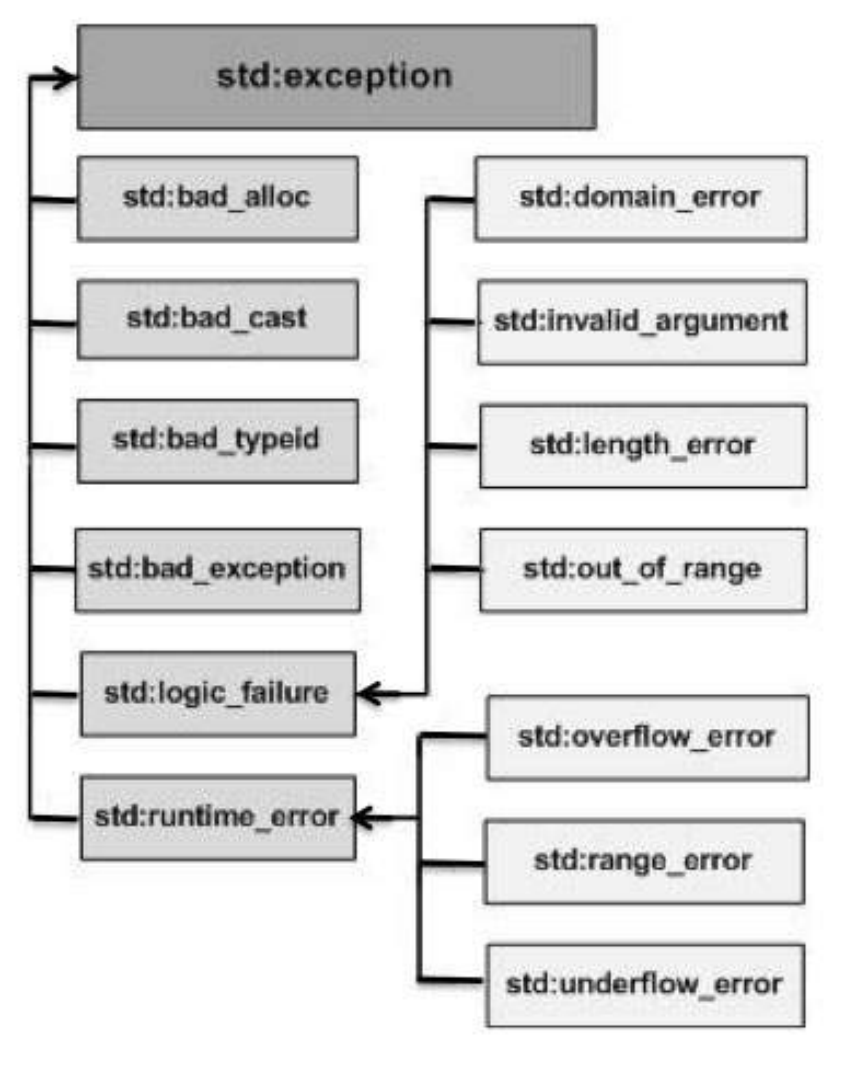
Beispiel

```
void ein_fehler() {
  // Wir haben einfach immer einen Fehler
  throw 23; // Wirft 23 als Ausnahme
}
```

Man sieht: Obwohl der Rückgabewert der Funktion `void` ist, können wir mit `throw` einen Integer als Ausnahme zurück geben.

Vordefinierte Exceptions

Verfügbar mit `#include <exception>`



Von https://www.tutorialspoint.com/cplusplus/cpp_exceptions_handling.htm

Das Werfen einer Ausnahme

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

void ein_fehler() {
  // Wir haben immer einen Fehler
  throw runtime_error("Oh nein, ein Fehler!!");
}

int main() {
    cout << "Alles OK" << endl;
    ein_fehler();
    cout << "Immer noch alles OK" << endl;
}
```

- Man kann einer Ausnahme üblicherweise einen String mitgeben der näher erklärt was genau passiert ist
- Wenn man eine Ausnahme nicht fängt, stürzt das Programm ab und man erfährt welche Ausnahme dafür verantwortlich war und welchen String sie enthalten hat

Das Fangen einer allgemeinen Ausnahme

- Das Stück Code in dem eine Ausnahme gefangen werden soll wird in ein `try { }` eingeschlossen
- Anschließend wird mit `catch(...) { }` angegeben welcher Code im Fall einer Ausnahme ausgeführt werden soll

Beispiel

```
try {
  cout << "Alles OK" << endl;
  eine_funktion();
  cout << "Immer noch alles OK" << endl;
} catch (...) {
  cout << "Ouch: Irgendwo oben gab es eine Ausnahme!" << endl;
}
```

Das Fangen einer speziellen Ausnahme

- `catch(...)` bedeutet, dass bei jeder Ausnahme reagiert werden soll
- Falls wir nur auf spezielle Ausnahmen reagieren wollen schreiben wir in die Klammern welche Ausnahme das sein soll und wie das Ausnahmeobjekt bezeichnet werden soll
- Wir können mehrere `catch` Anweisungen hintereinander hängen

Beispiel

```
try {
  cout << "Alles OK" << endl;
  eine_funktion();
  cout << "Immer noch alles OK" << endl;
} catch (overflow_error &e) {
  cout << "Ouch: Irgendwo oben gab es einen runtime_error!" << endl;
  cout << "Nachricht war: " << e.what() << endl;
} catch (range_error &e) {
    cout << "Ouch: Irgendwo oben gab es einen range_error!" << endl;
    cout << "Nachricht war: " << e.what() << endl;
}
```

Beispiel

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

void do_something() {
  if (rand() % 5 == 0)
    throw runtime_error("We got a 0, and we don't like it!");
}

int main() {
  for (int i = 0; i < 10; i++) {
    try {
      do_something();
      cout << "Alles OK!" << endl;
    } catch (exception &e) {
      cout << "Some error occured with the message: " << endl;
      cout << e.what() << endl;
    }
  }
}
```

Übung

- Implementieren Sie Ihre eigene Divisionsfunktion `double mydiv(double a, double b)` welche das Ergebnis von `a` geteilt durch `b` zurück gibt.
- Falls eine Division durch 0 droht, soll die Funktion eine Ausnahme (`domain_error`) zurück liefern.
- Testen Sie ihren Code an ein paar Beispielen und fangen Sie mögliche Ausnahmen mit `try` und `catch` ab