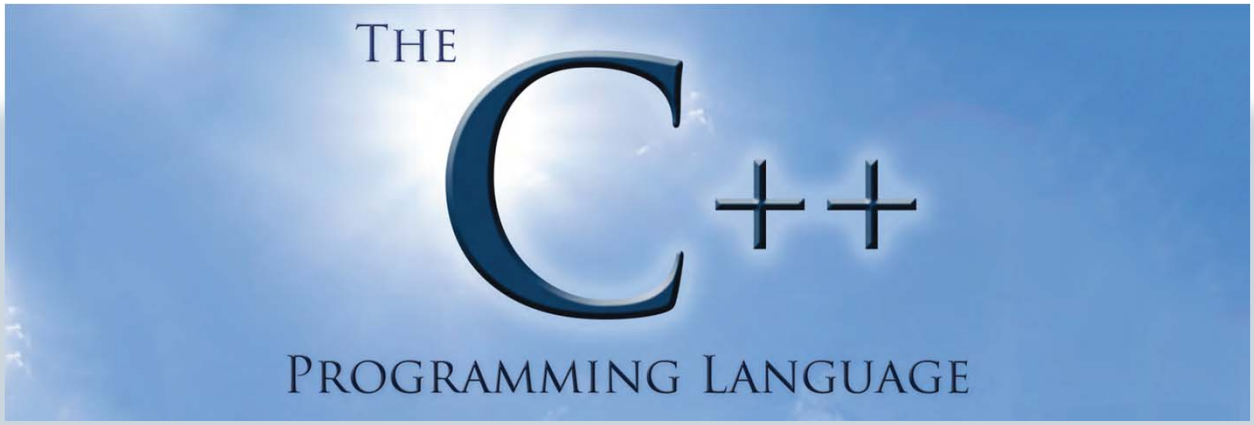

CPP-05 – C++ Exceptions

CPVR Vertiefungsmodul BTI-7281

Urs Künzler (urs.kuenzler@bfh.ch)

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

CPP-05 Table of Contents – C++ Exceptions

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

05.01 Fehlerbehandlung

- Das prinzipielle Problem bei der Fehlerbehandlung besteht darin, dass dort wo ein Fehler entdeckt wird meist nicht genügend Informationen vorhanden sind um den Fehler zu behandeln.
- Die Idee der Ausnahmebehandlung besteht darin, dass eine Funktion, die einen auftretenden Fehler nicht behandeln kann, eine Ausnahme (exception) wirft um einer übergeordneten Funktion die angemessene Fehlerbehandlung zu überlassen.
- Exceptions sind kein Mittel zur Kontrolle von Ausnahmesituationen die von aussen gesteuert werden (Interrupts, Signals, Messages ...).
- Jeder Fehler der bereits vom Compiler entdeckt werden kann, muss zur Laufzeit nicht mit Exceptions behandelt werden oder aufwendig mit dem Debugger gesucht werden!

05.02 C++ Exception Konzept

- Mit dem ANSI C++ Standard wurde ein allgemeiner Mechanismus zur getrennten Erkennung und Behandlung von Laufzeitfehlern in C++ integriert.
- Der Mechanismus beruht auf einem Ausdruck, der im Fehlerfall eine Ausnahme (exception) wirft (throw), die von einem entsprechenden Exception-Handler aufgefangen (catch) und behandelt wird.
- Diese Methode des Exception-Handlings erlaubt eine klare Trennung des eigentlichen Applikations-Codes und des Codes zur Behandlung von Fehlern.

05.03 C++ Exception Syntax

- **Zum Exception-Handling sind in C++ folgende Keywords definiert:**
  - `try` definiert den Gültigkeitsbereich, innerhalb dessen Fehler mit catch abgefangen und behandelt werden.
  - `throw` "wirft" ein entsprechende Ausnahme, wenn ein Fehler auftritt.
  - `catch` dient dazu eine Ausnahme "aufzufangen" und den Fehler zu behandeln.
- Beim Eintreten einer Applikations- oder Runtime-Exception (z.B. ein File- oder Memory Allozierungsfehler) wird ein Exception-Objekt erzeugt und anschliessend alle übergeordneten Blöcke geordnet verlassen (Stack Unwinding), bis die Ausnahme behandelt werden kann.
- Die Unterscheidung der auftretenden Fehler im catch Statement wird über den Typ des geworfenen Ausdrucks vorgenommen. Dabei können auch benutzer-definierte Typen (z.B. Fehlerklassen) verwendet werden.

C++ Exception Syntax II

Normaler Programmfluss:

```
try
{
    function1();
    function2();
    function3();
}
catch(errtype1 err){
  cout << "ErrorType 1";
}
catch(errtype2 err){
  cout << "ErrorType 2";
}
```

„Exception" Programmfluss:

```
try
{
    function1();
    function2();
    function3();
}
catch(errtype1 err){
    cout << "ErrorType 1";
}
catch(errtype2 err){
    cout << "ErrorType 2";
}
```

throw errtype2;

C++ Exception Syntax III

- **Einfaches Beispiel mit Integer-Typ als Exception:**

Deklaration:

```
void copy(const char* src, const char* dest)
{
     ifstream in(src);
     if (!in.good()) throw 0;

     ofstream out(dest);
     if (!out.good()) throw 1;

     char c;
     while (in.get(c)) out.put(c);
}
...
```

Main:

```
try {
     copy(file0, file1);
}
catch(int i) {
     cerr << "Fehler beim Oeffnen des "
          << (i == 0) ? "Quellfiles" : "Zielfiles";
}
```


Definition von Fehlerklassen

- Fehlerklassen sind normale Klassen und unterstehen deshalb den normalen Regeln von C++ für die Implementation von Klassen.
- Für verschiedene Fehlerkategorien können verschiedene hierarchisch gegliederte (vererbte) Fehlerklassen definiert werden.
- Beispiel - "Stack Range Error" für Stack Template-Klasse:

```
class CErrorStackRange
{
public:
     CErrorStackRange (int err) : err_code(err) {};
     int GetErrorCode(void) {return err_code;};

private:
     int err_code;
};
```

Auslösen einer Exception (throw)

- Nach dem Erkennen eines Applikationsfehlers wird eine Exception gezielt mit dem "**throw**" Statement ausgelöst.
- Beispiel - Pop() Funktion für Stack Template-Klasse:

```
template <class T>
T Stack<T>::Pop()
{
   // check for available stack element
   if ( numElems == 0 )
   {
       // "Stack Underflow Range Error", throw error code 1010
      throw CErrorStackRange(1010);
   }

   // pop element from stack
   return elements[ --numElems ];
}
```


Behandeln einer Exception (catch)

- Das Abfangen und die Behandlung einer Exception erfolgt durch eine Liste von "**catch**" Statements.
- Beispiel - Fehlerbehandlung für Stack Template-Klasse:

```
...
int max_elements = intStack.GetElementNum();
for (int i=0; i<max_elements; i++) {
   try {
      cout << "Stack Element: " << intStack.Pop() << endl;
   }
   catch ( CErrorStackRange& err ) {
      cout << "Stack Range Error (Error Code: "
           << err.GetErrorCode() << ")" << endl;
   }
   catch (...) { // watch order of exception handlers!
      cout << "Undefined Stack Error" << endl;
      throw; // re-throw exception for upper level handling
  }
}
```

Exception-Spezifikation für Funktionen

- Da die Exceptions einer Funktion einen wesentlichen Bestandteil der Interface-Definition darstellen, kann die Funktionsdeklaration mit einer Liste der geworfenen Exceptions ergänzt werden.
- Beispiele:

```
// Funktion kann beliebige Exceptions werfen
void f();

// Funktion wirft bestimmte Exceptions
void f() throw ( Exception1, Exception2 );

// Funktion wirft keine Exceptions
void f() throw ();
```

- **Hinweis:** In Konstruktoren sollten keine Exceptions verwendet werden, weil dann nicht gewährleistet ist, dass das Objekt auch tatsächlich vollständig alloziert worden ist (z.Bsp. bei einer Exception in einer Basisklasse).

Behandlung nicht aufgefangener Exceptions

- Nicht abgefangene Exceptions führen zu einem aussergewöhnlichen Programmabbruch. Zu diesem Zweck wird die Funktion **`terminate()`** aufgerufen, die wiederum die Funktion **`abort()`** aufruft. Dadurch wird das Programm sofort durch das C-Runtime System beendet, ohne z.B. die Destruktoren statisch erzeugter Objekte aufzurufen.
- Um allfällige Fehler einer Software zu vermeiden (nicht richtig geschlossene Files, DB's etc.) kann die **`terminate()`** Funktion mit **`set_terminate()`** durch eine eigene Terminate-Funktion ersetzt werden.
- Alternativ und besser kann dazu kann in der Funktion **`main()`**auch ein Top-Level **`try{} catch(...)`** Handler implementiert werden.
- Hinweis: Ein "finally" keyword wie in Java gibt es in C++ nicht.

05.04 Standard Library Exceptions

- Exceptions können durch das C++ Runtime System oder durch Funktionen der Standardbibliothek ausgelöst werden.
- Aus Kompatibilitätsgründen und durch die Heterogenität der Standard Library, werden Exceptions uneinheitlich verwendet.
- Verwendung der Basisklasse exception für alle vom System erzeugten Exceptions (Header File **<exception>**).
- Von dieser Basisklasse werden für verschiedene Fehler-Kategorien weitere Standard-Exceptions abgeleitet (Header File **<stdexcept>**).
- Vordefinierte Exceptions können durch eigene Exception-Klassen erweitert werden.

Standard Library Exceptiones II

- **Klassenhierarchie der Standard Library Exceptions:**