1 Fehler in einem Programm

Folgende Arten von Fehlern können bei der Programmierung auftreten:

- Syntaxfehler:
  Die Syntax legt die grammatikalischen Regeln der Programmiersprache fest. Viele Fehler in der Syntax, wie zum Beispiel das Vergessen eines Kommas als Abschlussmarkierung einer Zeile werden angezeigt, wenn das Programm kompiliert wird. Einige Entwicklungsumgebungen zeigen Fehler in der Syntax durch farbliche Hervorhebungen im Editor selber an.
- Kompilierungsfehler:
  Ein Kompilierungsfehler wird angezeigt, wenn eine Prozedur nicht aufgerufen werden kann. Diese Fehler werden durch den Compiler eines C++-Programms abgefangen.
- Laufzeitfehler
  Laufzeitfehler sind Bugs, die nach dem Start eines Programms auftreten können. Laufzeitfehler betreffen immer die Programmlogik. Laufzeitfehler entstehen zum Beispiel wenn eine Datei auf eine Diskette kopiert werden soll, aber keine Diskette im Laufwerk vorhanden ist. „Division durch Null“ ist auch ein häufig auftretender Laufzeitfehler. Diese Art von Fehlern können Programme zu einem unerwünschten Verhalten oder einen Programmabsturz zwingen. In C++ können diese Fehler mit Hilfe von Exceptions abgefangen werden.
- Logische Fehler
  Logische Fehler entstehen beim Design eines Programms oder bei der Definition von Anforderungen an das Programm. Logische Fehler können durch fehlendes Fachwissen oder Missverständnissen zwischen Nutzern und Entwicklern entstehen. Logische Fehler können nicht mit Hilfe von Programmstrukturen abgefangen werden.

1.1 Expection Handling in einer Prozedur

Fehler, die in einer Prozedur aufgefangen werden sollen, können in einer Enumeration abgelegt werden.

```
enum Fehler {ISZERO, RECT};
```

Das Exception Handling selber wird mit einem `try`-Block eingeleitet.

```
int main(){
      const int zahl = 0;
      const float wert = 4.5;
      double rueckgabe;

      try {
        rueckgabe = rechteck(zahl, wert, 'c');
      }
}
```

Innerhalb dieses `try`-Blocks versucht die Prozedur eine bestimmte Aufgabe auszuführen. In diesem Beispiel wird die Fläche eines Rechtecks berechnet.

Falls die Ausführung Fehler erzeugt, werden diese mit Hilfe eines `catch`-Blocks abgefangen.

```
int main(){
      const int zahl = 0;
      const float wert = 4.5;
      double rueckgabe;
      try {
        rueckgabe = rechteck(zahl, wert, 'c');
      }
      catch(const Fehler& e){
            cout << "Main: Länge oder Breite ist null.";
      }
}
```

Die Bearbeitung der Fehlermeldung wird mit Hilfe einer `throw`-Anweisung ausgelöst.

```
double cmToMili(double wert){
     double rueckgabe;

     if (wert <= 0) {
           throw ISZERO;
     }
     else {
            rueckgabe = wert / 10;
     }

     return (rueckgabe);
}

double rechteck(double breite, double laenge, char whichMass){
       double la;
       double br;

       la = laenge;
       br = breite;

       switch (whichMass){
                  case 'c':
                         la = cmToMili(laenge);
                         br = cmToMili(breite);
                         break;
       }

      return (br * la );
}
```

Die Behandlung der Execption kann in der Prozedur erfolgen, die den Fehler auslöst oder aber auch in einer übergeordneten Prozedur.
Das Programm wird mit Hilfe von Exception Handling in einem definierten Zustand beendet. Der Stack, der die Rücksprungadressen, eventuelle Funktionsparameter und die lokale Daten der Prozedur enthält, wird vom Laufzeitsystem aufgeräumt.

Der Ablauf eines Exception Handlings wird hier nochmals grafisch dargestellt.

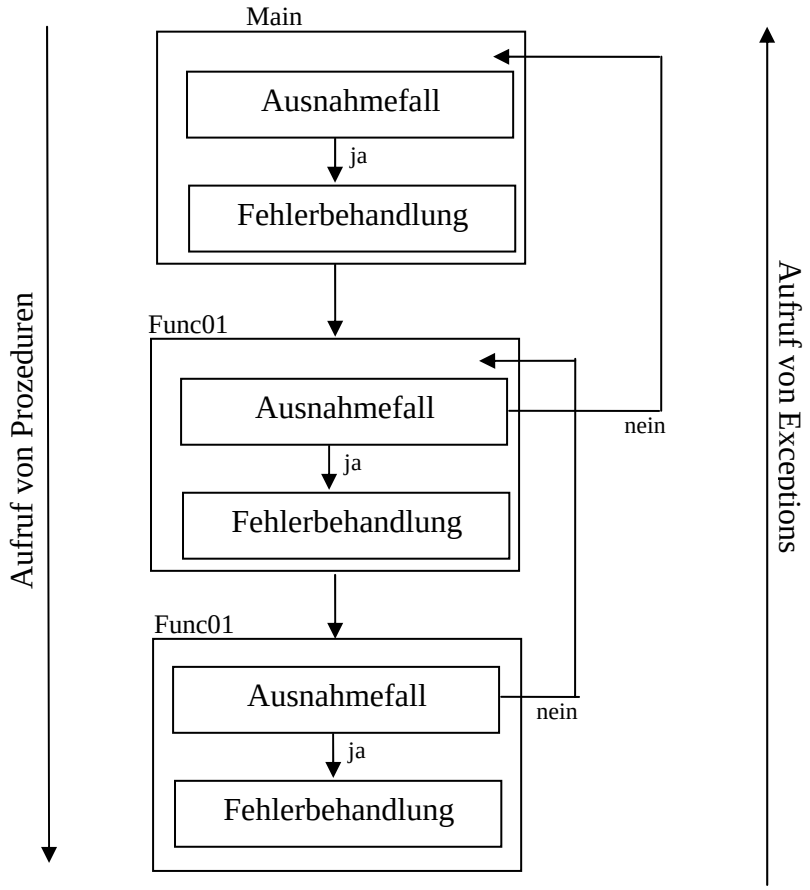

Abbildung 1: Ablauf eines Exception Handlings

Eine Prozedur oder Methode versucht eine bestimmte Aufgabe zu lösen (engl. try). Beim Lösen der Aufgabe kann ein Problem auftreten. Um dieses Problem abzufangen, wirft die Prozedur einen Laufzeitfehler aus (engl. throw). Eine Fehlerbehandlung versucht den aufgetretenen Fehler abzufangen (engl. catch). Falls der Fehler nicht in der Funktion behandelt werden kann, in der dieser aufgetreten ist, wird der Fehler an den Aufrufer der Funktion durchgeleitet. Die Durchleitung eines Fehlers kann bis zur Main-Funktion erfolgen. Falls der Fehler auch dort nicht abgefangen wird, bricht das Programm mit einer Fehlermeldung ab.

1.2 Exception Handling in einer Klasse

In dem folgenden Beispiel wird eine Klasse für ein Fenster entworfen. Das Fenster wird über einen Konstruktor initialisiert. Die Klasse besitzt Methoden zum Verändern der Position oder der Größe des Fensters. Falls die Fenstergröße einen bestimmten Wert unter- oder überschreitet, wird ein Exception Handling gestartet.

```
#include <iostream>
#include <exception>

class Fenster{
     private:
            int xPos;
            int yPos;
            int hoehe;
            int breite;

     public:
         Fenster(short width, short height){
                  xPos = yPos = 0;
                  if ((height < 1) || (width < 1)){
                  throw ISSMALL;
               }
               else {
                  hoehe = height;
                      breite = width;
               }
            }

            ~Fenster(){
                  cout << "Objekt zerstoert.";
            }

            void Move(int x, int y);
            void WinSize(int width, int height);
};
```

Der Fehler wird in der main-Funktion aufgefangen.

```
int main(){
   try {
      Fenster myWin(620, 480);
   }
   catch(const Fehler& e){
            cout << "Das Fenster ist zu groß oder zu klein.";
   }
   myWin.WinSize(10, 10);
   return 0;
}
```

Innerhalb des `try`-Blocks wird ein Objekt vom Typ Fenster erzeugt. Die Zeile `myWin.WinSize(10, 10)` erzeugt einen Kompilierungsfehler, weil das Objekt `myWin` nur in dem `try`-Block definiert ist.

Wenn das Objekt außerhalb des Blocks genutzt werden soll, muss ein Objektzeiger definiert werden.

```
int main(){
   Fenster *objPtr = 0;

   try {
        objPtr = new Fenster(0, 480);
   }
   catch(const exception& e){
             cout << "Fehler\n";
             delete(objPtr);
   }

   objPtr->WinSize(10, 10);
   delete(objPtr);
   objPtr = 0;
   return 0;
}
```

In diesem Beispiel verweist der Zeiger auf ein mit `new()` erzeugtes Objekt. Mit Hilfe der Funktion `delete()` wird Speicher freigegeben, der mit `new()` erzeugt wurde. Bei einem Zeiger auf einem Objekt wird das Objekt zerstört und der Verweis auf dieses Objekt. Der Zeiger enthält anschließend einen undefinierten Wert.

1.3 Nutzung von Smart Pointern am Beispiel „auto_ptr“

Smart-Pointer werden für die Erzeugung von dynamischen Objekten verwandt. Ein Smart-Pointer kümmert sich selber um die Speicherverwaltung. Sobald ein referenziertes Objekt zerstört wird, gibt der Smart-Pointer den dazugehörigen Speicher frei und setzt sich selber auf NULL.
`auto_ptr` ist ein Smart-Pointer, der als Template in der Header-Dateid <memory> definiert ist. Der Zeiger wird für die Verwaltung von mit `new()`-erzeugten Objekten eingesetzt. `auto_ptr` wird immer lokal auf dem Stack abgelegt.
Der folgende Code-Ausschnitt zeigt die Verwendung eines `auto_ptr`.

```
#include <iostream>
#include <exception>
#include <memory>

int main(){
   auto_ptr<Fenster> objPtr;
   try {
        auto_ptr<Fenster> objPtr (new Fenster(0,1));
   }
   catch(const exception& e){
             cout << "Fehler\n";

   }

   objPtr->WinSize(10, 10);

   return 0;
}
```

Durch die Verwendung des Smart-Pointer wird die Funktion `delete()` nicht benötigt. Der Smart-Pointer gibt den Speicher, auf den er verweist, immer automatisch frei. Anschließend wird der `auto_ptr` mit NULL initialisiert, um ein Verweis auf eine nicht bekannte Speicherstelle zu vermeiden.

Nachteile:

- Der Zeiger `auto_ptr` besitzt immer ein Objekt.
- Mehrere Smart-Pointer können nicht auf dasselbe Objekt verweisen.
- Beim Kopieren wird der Besitz eines Objekts gelöscht.
- `auto_ptr` darf nicht für Arrays sowie nicht als Element von STL-Containern genutzt werden.

1.4 Standard-Exception

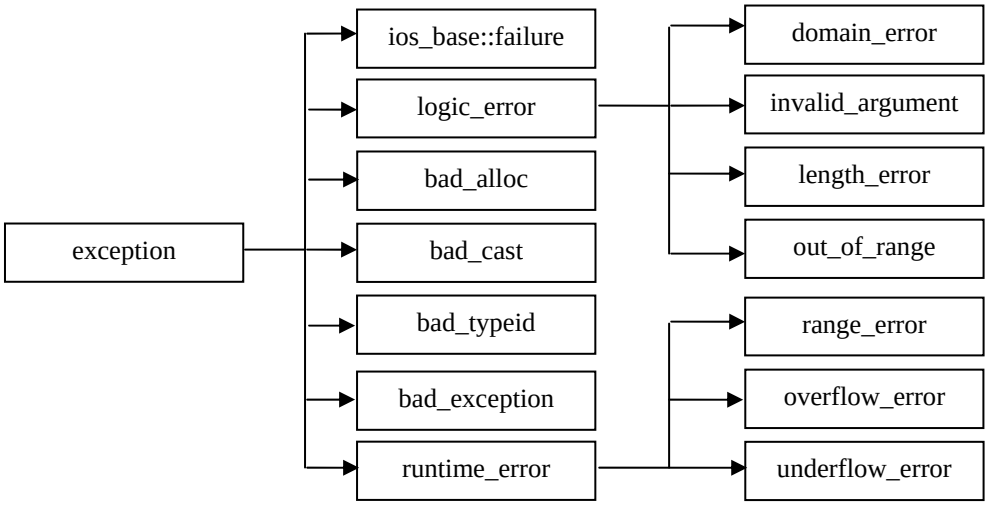

Abbildung 2: Standard-Exception-Klasse

Alle Standard-Expections enthalten eine public-Methode `what()`, um eine Fehlermeldung als String herauszugeben. Der Text der Fehlermeldung ist implementierungsabhängig.
Die Standard-Exceptions werden in

- System-Exceptions,
- Exceptions, die logische Fehler wie zum Beispiel das Überschreiten der Bereichsgrenzen und
- Exceptions, die aufgrund von Berechnungen auftreten

unterteilt.

Die folgenden Ausnahmen sind als System-Exceptions definiert:

- `bad_alloc` ist in der Header-Datei `new` definiert und wird ausgelöst, wenn der Operator `new` den angeforderten Speicher nicht reservieren kann. Veraltete Compiler lösen häufig dieses Ereignis nicht aus.
- `bad_cast` wird ausgelöst, wenn die Typkonvertierung einer Referenz mit Hilfe des `dynamic_cast`-Operators fehlschlägt. Eine fehlerhafte Konvertierung eines Zeigers liefert einen NULL-Zeiger und keine Exception. `bad_cast` ist in der Header-Datei <typeinfo> definiert.
- `bad_typeid` wird ausgelöst, wenn die Typinformation eines dereferenzierten NULL-Zeigers ermittelt wird. Die Definition der Exception befindet sich in der Header-Datei <typeinfo>.
- `bad_exception` ist in der Header-Datei <exception> definiert und wird ausgelöst, wenn eine undefinierte Exception au einer Funktion oder Methode heraus geworfen wird.

Ein Beispiel für bad_alloc:

```
int main(){
   Fenster *objPtr = 0;

   try {
            objPtr = new Fenster(0,200);
   }
   catch(int fehler){
            cout << "Das Fenster ist zu klein gewählt.";
            delete(objPtr);
   }
   catch(bad_alloc &ex){
            cout << ex.what() << endl;
   }

   delete(objPtr);
   objPtr = 0;

   return 0;

}
```

Exceptions, die zum Bereich `logic_error` gehören, liegen im Namensraum `std` und sind in der Header-Datei <stdexcept> definiert. Die Fehlermeldungen können durch die Standard-Bibliothek oder das eigene Programm ausgelöst werden.

- `out_of_range` wird ausgelöst, wenn ein Wert eine Bereichsgrenze überschreitet. Es findet ein Zugriff auf einen unerlaubten Index statt.
- `invalid_argument` signalisiert ein falsches Argument in einer Funktion.
- `length_error` wird ausgelöst, wenn ein Objekt größer als die maximale Größe erzeugt werden soll.

`runtime_error`-Exception sind Fehler, die aufgrund falscher Berechnungen geworfen werden. Die Exception liegen im Namensraum `std` und sind in der Header-Datei <stdexcept> definiert.

- `range_error` wird bei einem fehlerbehafteten Feldzugriff ausgelöst.
- `overflow_error` signalisiert einen arithmetrischen Überlauf und `underflow_error` einen arithmetrischen Unterlauf.