

Was sonst noch dazu gehört

- Zahlendarstellung - Konvertierung
 - Dezimal – Dual/Binär – Oktal – Hexadezimal
 - Einer-/Zweierkomplement
- Bits und Bytes
 - Einer- und Zweierkomplement
- Ausnahmebehandlung
 - Exceptions – try – throw – catch

Zahlendarstellung

- Zahlen, insbesondere die natürlichen Zahlen (0, 1, 2, 3, ...) und die ganzen Zahlen (... , -3, -2, -1, 0, 1, 2, 3, ...) sind von der Schule bekannt. Es geht im Folgenden nicht um die Eigenschaft, sondern um die Darstellung der Zahlen.
- Das im täglichen Umgang vertraute Dezimalsystem findet seinen Begründung darin, dass der Mensch 10 Finger zum Zählen hat.
- Mathematische aussagen sind aber unabhängig davon, wie viele Finger derjenige hat, der sie formuliert.
- Eine intelligente Spezies mit 8 Fingern würde sicherlich die gleiche Mathematik entwickeln wie wir, nur die Darstellung ist eine andere.
- Die Gesetze des Zählens und Rechnens sind unabhängig von der Darstellung immer die gleichen.
- In der Datenverarbeitung ist die Darstellung im Dezimalsystem denkbar ungeeignet.

Zahlendarstellung

- Es bieten sich Dual-, Oktal- oder Hexadezimalsystem zur Darstellung an.
- Weitere Darstellungsmethoden sind z.B. Kettenbrüche oder die römische Zahlendarstellung).
- Die vermeintliche Überlegenheit des Dezimalsystems gegenüber den anderen diskutierten Systemen beruht ausschließlich auf der langjährigen Vertrautheit mit dieser speziellen Darstellung.
- Würde man als Kind nur die Hexadezimaldarstellung lernen, würde man dies als das natürlichste und einfachste aller Zahlensysteme empfinden.

Dirk Seeber , Informatik , Teil 6

3

Zahlendarstellung

- Die folgenden Betrachtungen basieren auf dem bekannten Prinzip der "Division mit Rest".
- Dies Prinzip lautet:
 - Gegeben sind zwei ganze Zahlen a und b mit $a \geq 0$ und $b > 0$.
 - Dann gibt es genau zwei ganze Zahlen m (Multiplikator) und r (Rest) mit:
 - $m \geq 0$
 - $0 \leq r < b$
 - $a = m * b + r$
- Wählen wir $a = 5349$ und $b = 8$ so erhalten wir bei der Division von 5349 durch 8:
 - $5349 = 668 * 8 + 5$
 - Also $m = 668$ (Multiplikator) und $r = 5$ (Rest)
- Wir bearbeiten den Multiplikator nach der gleichen Methode weiter
 - $668 = 83 * 8 + 4$
 - $83 = 10 * 8 + 3$
 - $10 = 1 * 8 + 2$
 - $1 = 0 * 8 + 1$bis er ganz verschwunden ist.

Dirk Seeber , Informatik , Teil 6

4

Zahlendarstellung

- Zusammensetzen der einzelnen Puzzlesteine

$$\begin{aligned}5349 &= 668 * 8 + 5 \\ &= (83 * 8 + 4) * 8 + 5 \\ &= ((10 * 8 + 3) * 8 + 4) * 8 + 5 \\ &= (((1 * 8 + 2) * 8 + 3) * 8 + 4) * 8 + 5 \\ &= ((1 * 8^2 + 2 * 8 + 3) * 8 + 4) * 8 + 5 \\ &= (1 * 8^3 + 2 * 8^2 + 3 * 8 + 4) * 8 + 5 \\ &= 1 * 8^4 + 2 * 8^3 + 3 * 8^2 + 4 * 8^1 + 5 * 8^0\end{aligned}$$

- Die Ausgangszahl lässt sich mit geeigneten Koeffizienten als Summe von Potenzen der Basiszahl 8 darstellen.
- Wesentlich sind dabei die vor den Potenzen auftretenden Koeffizienten, die nach dem Prinzip der Division mit Rest eindeutig bestimmt sind und im Bereich von 0 bis 7 liegen können. Da die Potenzen durch ihre Stellung in der Formel eindeutig festgelegt sind, kann man vereinfacht schreiben

$$5349 = (12345)_8$$

Dirk Seeber , Informatik , Teil 6

5

Oktaldarstellung - Beispiel

$$\begin{aligned}5349 &= 668 * 8 + 5 \\ 668 &= 83 * 8 + 4 \\ 83 &= 10 * 8 + 3 \\ 10 &= 1 * 8 + 2 \\ 1 &= 0 * 8 + 1\end{aligned}$$

Ergebnis: 12345₈

$$\begin{aligned}5349 / 8 &= 668 \text{ Rest } 5 \\ 668 / 8 &= 83 \text{ Rest } 4 \\ 83 / 8 &= 10 \text{ Rest } 3 \\ 10 / 8 &= 1 \text{ Rest } 2 \\ 1 / 8 &= 0 \text{ Rest } 1\end{aligned}$$

Ergebnis: 12345₈

Dirk Seeber , Informatik , Teil 6

6

Zahldarstellung

- Den Zusatz der Basiszahl kann man weglassen, wenn aus dem Zusammenhang klar ist, welche Basiszahl gemeint ist.
- Da beliebige Zahlen a und b gewählt wurden, gilt, dass eine beliebige Ausgangszahl a ($a \geq 0$) in dieser Weise bezüglich einer beliebigen Basis b ($b > 0$) zerlegt werden kann.
- Beschränkt man sich auf n Stellen, können bzgl. der Basis b auf diese Weise die Zahlen von 0 bis $b^n - 1$ dargestellt werden.

Dezimaldarstellung

- Das Dezimalsystem ordnet sich in die zuvor dargestellte Zusammenhänge ein.
- Für $b = 10$ und mit den Ziffernsymbolen $0, 1, 2, \dots, 9$ erhält man die vertraute Dezimaldarstellung
- Zum Beispiel:

$$56789 = 5 * 10^4 + 6 * 10^3 + 7 * 10^2 + 8 * 10^1 + 9 * 10^0$$

Dualdarstellung

- Es wird das zuvor angesprochene Verfahren verwendet.
- Ausgangszahl wird sukzessive durch 2 dividiert.
- Die gesuchte Dualzahl erhält man über die Divisionsreste.
- Beispielzahl: 56789

Dualdarstellung - Beispiel

```
56789 / 2 = 28394 Rest      1
28394 / 2 = 14197 Rest      0
14197 / 2 = 7098 Rest      1
7098 / 2 = 3549 Rest       0
3549 / 2 = 1774 Rest       1
1774 / 2 = 887 Rest        0
887 / 2 = 443 Rest        1
443 / 2 = 221 Rest        1
221 / 2 = 110 Rest        1
110 / 2 = 55 Rest         0
55 / 2 = 27 Rest         1
27 / 2 = 13 Rest         1
13 / 2 = 6 Rest          1
6 / 2 = 3 Rest          0
3 / 2 = 1 Rest          1
1 / 2 = 0 Rest          1
```

Ergebnis: 1101110111010101

Dualdarstellung

- Um das Ergebnis wieder ins Dezimalsystem zurückzurechnen, muss nur ausgerechnet werden:

$$\begin{aligned}(1101110111010101)_2 &= 1 * 2^{15} + 1 * 2^{14} + 0 * 2^{13} \\ &\quad + 1 * 2^{12} + 1 * 2^{11} + 1 * 2^{10} \\ &\quad + 0 * 2^9 + 1 * 2^8 + 1 * 2^7 \\ &\quad + 1 * 2^6 + 0 * 2^5 + 1 * 2^4 \\ &\quad + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 \\ &\quad + 1 * 2^0 \\ &= 32768 + 16384 + 4094 + 2048 \\ &\quad + 1024 + 256 + 128 + 64 + 16 \\ &\quad + 4 + 1 \\ &= 56789\end{aligned}$$

und man erhält die ursprüngliche Darstellung im Dezimalsystem zurück.

Dualdarstellung

- Das Dualsystem ist das Zahlensystem, in dem Computer rechnen.
- Für Menschen hat das aber erhebliche Nachteile:
 - Wegen der kleinen Basis sind die Zahlen zu lang und umständlich zu handhaben.
 - Es fehlt eine "Größenvorstellung" von Dualzahlen.
- Der Mensch braucht zusätzliche Zahlensysteme, die eine einfache Umrechnung ins Dualsystem erlauben, dabei aber "menschenfreundlicher" sind als das Dualsystem.

Hexadezimaldarstellung

- Das Hexadezimalsystem hat die Basis 16 und benutzt die Ziffern 0, 1, 2, ..., 9, und für die Darstellung der Ziffern 10..15 die Zeichen a, b, ..., f bzw. A, B, ..., F.
- Die besondere Eignung der Basiszahl 16 liegt darin begründet, dass es sich um eine Potenz von 2 ($16 = 2^4$) handelt.
- Damit ergibt sich folgende Umrechnungstabelle für Ziffern (nächste Seite!)

Hexadezimaldarstellung

Hexadezimal	Dezimal	Dual
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Hexadezimaldarstellung

- Die 16 eine Potenz von 2 ist, kann für das Umrechnen ins Dualsystem und umgekehrt ein ähnliches verfahren verwendet werden, wie bei der Umrechnung ins Oktalsystem. Jetzt werden allerdings 4 Ziffern der Dualdarstellung zusammengefasst.:

Umrechnung Hex <=> Dual <=> Oktal																
Hexadezimal	5				9				D							
Dual	0	1	0	1	1	0	0	1	1	1	0	1				
Oktal	2				6				3				5			

Hexadezimaldarstellung

- Die Umrechnung von Hexadezimalzahlen in Dezimalzahlen und umgekehrt folgt dem von den Dualzahlen bekannten Schema.
- In der einen Richtung ermittelt man die Reste durch fortlaufende Division durch 16:

```

56789 / 16 = 3549 Rest 5
3549 / 16 = 221 Rest 13 (entspricht d)
221 / 16 = 13 Rest 13 (entspricht d)
13 / 16 = 0 Rest 13 (entspricht d)
Ergebnis: ddd5
    
```

- In der anderen Richtung rechnet man einfach aus:
 $(ddd5)_{16} = 13 * 16^3 + 13 * 16^2 + 13 * 16^1 + 5 * 16^0$
 $= 53248 + 3328 + 208 + 5$
 $= 56789$
- Das Hexadezimalsystem ist das üblicherweise in der maschinennahen Programmierung verwendete Zahlensystem.
- Eine gewisse Vertrautheit mit diesem Zahlensystem ist daher unumgänglich.

Bits and Bytes

- Die kleinste Informationseinheit auf einem Digitalrechner bezeichnen wir als **Bit (Binary digit, bzw. engl bit = ein bisschen, nicht das Bier aus der Eifel)**.
- Ein Bit kann die logischen Werte 0 (Bit gelöscht) und 1 (Bit gesetzt) annehmen.
- Alle Informationen auf einem Rechner (Programme und Daten) sind als Folgen von Bits gespeichert.
- Den Speicherinhalt eines Rechners kann man zu einem bestimmten Zeitpunkt als sehr lange Folge von Bits ansehen.
- Um Teile dieser Information gezielt ansprechen und manipulieren zu können, muss der Bit-Folge eine Struktur gegeben werden.
- Zunächst fasst man jeweils 8 Bit zusammen und nennt diese - Größe ein Byte

Dirk Seeber , Informatik , Teil 6

17

Bits und Bytes

Byte							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	0	0	1	1	0	1

- Als Dualzahl interpretiert kann ein Byte also Zahlen von 0 - 255 (hex. 00 - ff) darstellen.
- Bit 7 bezeichnet man als das höchstwertige (most significant) Bit, Bit 0 als das niedrigstwertige (least significant) Bit.
- Als Dualzahl interpretiert hat das höchstwertige Bit den Stellenwert $2^7 = 128$, das niedrigstwertige den Wert $2^0 = 1$.

Dirk Seeber , Informatik , Teil 6

18

Bits and Bytes

- Im Speicher will man auch vorzeichenbehaftete Zahlen ablegen
- **Achtung!!!**
Es ist nicht zwingend eine negative Zahl, sondern es ist Platz für ein Vorzeichen vorgesehen.
- Zur Speicherung negativer Zahlen wählt man das **Zweierkomplement**.
- Das Zweierkomplement einer Dualzahl erhält man, in dem man zunächst das Bitmuster invertiert (man nennt dies das **Einerkomplement**) und anschließend 1 addiert.

Zweierkomplement

Ausgehend von einer Dualzahl bildet man zunächst das Einerkomplement, indem aus Einsen Nullen und aus Nullen Einsen werden.

Dualzahl (92)	0	1	0	1	1	1	0	0
Einerkomplement	1	0	1	0	0	0	1	1

Anschließend zählt man zum Einerkomplement 1 hinzu.

Ein möglicher Überlauf wird ignoriert.

Einerkomplement	1	0	1	0	0	0	1	1
1 hinzu zählen	0	0	0	0	0	0	0	1
Eventuell Übertrag						1	1	
Zweierkomplement (-92)	1	0	1	0	0	1	0	0

Zweierkomplement

- Die besondere Eignung des Zweierkomplements zur Darstellung negativer Zahlen erkennt man, wenn man eine Zahl und ihr Zweierkomplement addiert.

Dualzahl (92)		0	1	0	1	1	1	0	0
Zweierkomplement (-92)		1	0	1	0	0	1	0	0
Summe (0)	(1)	0	0	0	0	0	0	0	0

Dualzahl (1)		0	0	0	0	0	0	0	1
Zweierkomplement (-1)		1	1	1	1	1	1	1	1
Summe (0)	(1)	0	0	0	0	0	0	0	0

Dualzahl (127)		0	1	1	1	1	1	1	1
Zweierkomplement (-127)		1	0	0	0	0	0	0	1
Summe (0)	(1)	0	0	0	0	0	0	0	0

Dirk Seeber , Informatik , Teil 6

21

Zweierkomplement

- Auch andere Rechenaufgaben liefern korrekte Ergebnisse
- Beispiel: $55 - 11 = 44$.

Dualdarstellung von 55		0	0	1	1	0	1	1	1
Zweierkomplement von 11		1	1	1	1	0	1	0	1
Ergebnis: Dualdarstellung von 44	(1)	0	0	1	0	1	1	0	0

- Das Zweierkomplement eignet sich also zur Darstellung negativer Zahlen und zum Rechnen mit vorzeichenbehafteten Zahlen.
- In einem 8-stelligen Datenwort unseres Beispielrechners lassen sich die Zahlen von -128 bis +127 darstellen.

Dirk Seeber , Informatik , Teil 6

22

Bits und Bytes

- Es liegt an uns, ein im Rechner abgelegtes Muster aus Nullen und Einsen als vorzeichenlose oder vorzeichenbehaftete Zahl zu lesen.
- Beispiel:
 - $11011000 = 216 = \text{vorzeichenlos}$
 - $11011000 = -40 (= 00101000 + 1) = \text{vorzeichenbehaftete}$
- Die Bereiche darstellbarer Zahlen werden später noch mal angesprochen.
- $(400)_{16} = (2000)_8 = 1024$ Bytes werden zu einem Kilobyte (KB) zusammengefasst. Das Wort "Kilo" stammt aus dem Griechischen und bedeutet Tausend. Ist in diesem Zusammenhang allerdings irreführend.
- Der "Fehler" zieht sich konsequent durch das gesamte Bemessungssystem für die Speichergröße. (**Tabelle!!!**)

Bits und Bytes

Maßeinheit	Abk.	Hexadezimal	Oktal	Dezimal	2^x
Kilobyte	KB	400	2 000	1 024	2^{10}
Megabyte	MB	100 000	4 000 000	1 048 576	2^{20}
Gigabyte	GB	40 000 000	10 000 000 000	1 073 741 824	2^{30}
Terabyte	TB	10 000 000 000	20 000 000 000 000	1 099 511 627 776	2^{40}

Bit - Operationen

- Obwohl auf einem Rechner nur einzelne Bytes adressiert werden können, kann man auch auf einzelne Bits eines Bytes zugreifen.
- Dazu verwendet man so genannte Bit-Operationen.
- Diese können nur auf ganzzahlige Operanden angewandt werden und erlauben die Manipulation einzelner Bits in den Daten.
- Im Einzelnen handelt es sich um folgende Operatoren:
 - ~ bitweises Komplement (Einerkomplement)
 - & bitweises "und"
 - | bitweises "oder"
 - ^ bitweises "entweder oder" (exklusives oder , xor)
 - << bitshift nach links
 - >> bitshift nach rechts
- **ACHTUNG:** Die 4 erstgenannten sind eng verwandt mit den logischen Operatoren, die bereits behandelt wurden.

Bit - Operationen

Das bitweise Komplement einer Zahl (~x):								
x	0	1	0	1	1	1	0	0
~x	1	0	1	0	0	0	1	1
Das bitweise "und" zweier Zahlen (x & y):								
x	1	0	1	0	1	0	1	0
y	1	1	0	0	1	1	0	0
x & y	1	0	0	0	1	0	0	0
Das bitweise "oder" zweier Zahlen (x y):								
x	1	0	1	0	1	0	1	0
y	1	1	0	0	1	1	0	0
x y	1	1	1	0	1	1	1	0

Bit - Operationen

Das bitweise "entweder oder" zweier Zahlen ($x \wedge y$):								
x	1	0	1	0	1	0	1	0
y	1	1	0	0	1	1	0	0
$x \wedge y$	0	1	1	0	0	1	1	0
Das bitweise Verschieben um n-Bit nach links ($x \ll n$):								
x	1	1	1	0	0	1	1	1
n	2							
$x \ll n$	1	0	0	1	1	1	0	0
Das bitweise Verschieben um n-Bit nach rechts ($x \gg n$):								
x	1	1	1	0	0	1	1	1
n	2							
$x \gg n$	0	0	1	1	1	0	0	1

Dirk Seeber , Informatik , Teil 6

27

Bit - Operationen

- Verwendung finden diese Operatoren vorwiegend in der maschinennahen Programmierung, da dort häufig ganz bestimmte Bitmuster erzeugt werden müssen oder gefragt wird, ob in einem Bitmuster ein bestimmtes Bit gesetzt ist.
- Mit Hilfe solcher Operatoren kann man z.B. 8 ja/nein - Informationen (so genannte Flags) in einer char-Variablen (pro Flag ein Bit) platzsparend unterbringen oder gezielt manipulieren.
- Beispiele:
 - Setze das n-te Bit in x:

$$x = x | (1 \ll n)$$
 - Lösche das n-te Bit in x:

$$x = x \& \sim(1 \gg n)$$
 - Invertiere das n-te Bit in x:

$$x = x \wedge (1 \ll n)$$
 - Prüfe, ob das n-te Bit in x gesetzt ist:

$$\text{if } (x \& (1 \ll n))$$

Dirk Seeber , Informatik , Teil 6

28

Ausnahmebehandlung (exception)

- In Anwendungen tritt manchmal ein Fehler auf, d.h. das Programm macht nicht das, was der Benutzer eigentlich erwartet. Dabei sind folgende Fehlerarten unterscheidbar:
 - **Bedienungsfehler**, der Benutzer verwendet das Programm nicht im Sinne der Bedienungsanleitung.
 - **Programmfehler**, im Programm wird nicht gemäß der Spezifikation reagiert.
 - **Umgebungsfehler**, der korrekte Ablauf des Programms kann nicht sichergestellt werden, weil die Umgebung nicht die Anforderungen erfüllt (Speicher, Plattenplatz, etc.).

Ausnahmebehandlung (exception)

- Je nach Fehlerart sind die Reaktionen und Fehlermeldungen unterschiedlich:
 - Bedienungsfehler dürfen nie Programmabbrüche hervorrufen. Fehlbedienungen müssen dem Benutzer verständlich per Programm erläutert werden.
 - Programmierfehler müssen vom Programmierer behoben werden.
 - Umgebungsfehler sollten nicht zu Programmabbrüchen führen – dies kann aber nicht immer vermieden werden.
- Um das gewünschte Verhalten zu erreichen ist neben einem sauberem Programmentwurf und sorgfältigem Testen das Konzept der **Ausnahmebehandlung** einzusetzen.
- Gibt es allerdings nur in C++.

Ausnahmebehandlung (exception)

- Grundidee der Ausnahmebehandlung
Ein Programm soll so strukturiert werden, dass es getrennte Bereiche im Code gibt für den normalen Fall und kritische Bereiche für den Fehlerfall.
- Der kritische Programmblock wird „versuchsweise“ ausgeführt:
 - im Fehlerfall wird der normale Ablauf unterbrochen und
 - zum Fehlerbehandlungsblock gesprungen
 - tritt kein Fehler ein, wird der normale Ablauf bis zum Ende ausgeführt und
 - Der Fehlerblock ignoriert.

Ausnahmebehandlung (exception)

- Im Fehlerfall wird dabei ein Ausnahmeobjekt erzeugt, das im Fehlerbehandlungsblock auswertbar ist und Daten enthält, die zum Wiederaufsetzen oder zu Fehlermeldungen verwendbar sind.
- Der Ablauf kann wie folgt charakterisiert werden:
 - Eine Funktion (bis jetzt main) versucht (engl. try) den kritischen Block auszuführen.
 - Wenn ein Fehler festgestellt wird, wirft (engl. throw) sie eine Ausnahme (engl. exception) aus.
 - Die Ausnahme wird vom Fehlerbehandlungsblock aufgefangen (engl. catch).

Ausnahmebehandlung (exception)

- Allgemeines Beispiel:

```
// Programmcode
// Programmcode
// Programmcode
try {
    // Programmcode
    if ( FehlerAufgetreten )
        throw "Info";
    // Programmcode
    if ( andererFehlerAufgetreten )
        throw "andere Info";
    // Programmcode
}

catch (const char *Text) {
    // ggfs. Reparatur- und/oder
    // Aufräumarbeiten
    cout << Text << endl;
}
```

unkritischer Bereich

kritischer Bereich

Bereich der Fehlerbehandlung

Dirk Seeber , Informatik , Teil 6

33

Ausnahmebehandlung (exception)

- Exception Beispiel (Wurzel einer negativen Zahl):

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

int main()
{
    float zahl;
    cout << "positive Zahl eingeben: ";
    try
    {
        cin >> zahl;
        if (zahl < 0)
            throw "keine positive Zahl!";
        cout << "Wurzel von " << zahl << " ist ";
        cout << sqrt(zahl) << endl;
    }

    catch (const char * text)
    {
        cout << "Fehler: " << text << endl;
    }
    return 0;
}
```

Dirk Seeber , Informatik , Teil 6

34

Ausnahmebehandlung (exception)

- Ausnahmebehandlung unterschiedlichen Typs
In einem try-Block können unterschiedliche Ausnahmen geworfen werden, für jeden Datentyp aber höchstens eine.
- Dabei wird der erste passend catch-Block ausgeführt:
 - die Auswahl erfolgt anhand des **Datentyps** des Ausnahmeobjekts
 - das Sprungziel ist durch den Datentyp definiert;
 - wenn kein passender catch-Block gefunden wird, wird in der „Schachtelungsebene“ höher gesucht, bis einer gefunden wird oder falls keiner vorhanden ist, wird das Programm beendet.
- Dies ermöglicht eine differenzierte Fehlerbehandlung.

Dirk Seeber , Informatik , Teil 6

35

Ausnahmebehandlung (exception)

- Exception Beispiel (unterschiedliche catch-Blöcke):

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

int main()
{
    const int maxValue = 100;
    int zahl1, zahl2, ergebnis;
    cout << "zwei Zahlen eingeben: ";
    try
    {
        cin >> zahl1 >> zahl2;
        if (zahl1 > maxValue)
            throw zahl1 - maxValue;
        if (zahl2 > maxValue)
            throw zahl2 - maxValue;
        if (0 == zahl2)
            throw "Nulldivision nicht moeglich!";
        ergebnis = zahl1 / zahl2;
        cout << zahl1 << " / " << zahl2 << " = ";
        cout << ergebnis << endl;
    }
}
```

Dirk Seeber , Informatik , Teil 6

36

Ausnahmebehandlung (exception)

- Exception Beispiel (unterschiedliche catch-Blöcke) (Fortsetzung):

```
catch (const int wert)
{
    cout << "Fehler: Wert um " << wert;
    cout << " zu gross!" << endl;
}

catch (const char * text)
{
    cout << "Fehler: " << text << endl;
}

return 0;
}
```

Dirk Seeber , Informatik , Teil 6

37

Ausnahmebehandlung (exception)

- **Ausnahmebehandlung über Funktionsgrenzen**
Eine throw-Anweisung erzeugt ein Ausnahmeobjekt, das eine Fehlermeldung und/oder Daten zum Wiederaufsetzen enthält.
- Das Ausnahmeobjekt wird in der Aufrufhierarchie zur Fehlerbehandlung weitergereicht:
 - jede Funktion, die kein passendes catch enthält, wird ordentlich beendet,
 - dann wird in der Aufrufumgebung weiter nach einem catch gesucht.
- Damit ist die Fehlerbehandlung über mehrere Aufrufebenen hinweg möglich:
 - eine Bibliotheksfunktion erkennt Fehler, kann sie aber nicht behandeln
 - die Aufrufumgebung kann Fehler behandeln, aber nicht selbst erkennen.

Dirk Seeber , Informatik , Teil 6

38