

Übung 02: Informationstechnik (IT)

Marc Weber, Daniel Grimm

Institutsleitung

Prof. Dr.-Ing. Dr. h.c. J. Becker

Prof. Dr.-Ing. E. Sax

Prof. Dr. rer. nat. W. Stork

Institut für Technik der Informationsverarbeitung (ITIV)



Teil 1: Besprechung Aufgabenblatt 2

Aufg. 2.01: Verständnisfragen Lsg. (1)

- a) Die Quelldatei wird zur Übersetzung an den Compiler übergeben.
- b) Der Linker bindet eine Objektdaten mit anderen Modulen zu einer ausführbaren Datei.
- c) Beginnt die Zeile mit einem Doppelkreuz # am Anfang, so ist die Zeile für den Präprozessor bestimmt.
- d) Ein Datentyp bestimmt
 - i. die Art der Darstellung der Daten auf dem Bildschirm → Falsch
 - ii. die Art der internen Darstellung der Daten → Richtig
 - iii. die Anzahl der benötigten Speicherplätze in Bytes → Richtig

Aufg. 2.01: Verständnisfragen Lsg. (2)

- e) Wird eine interne Variable ohne Initialisierung definiert, so wird
- i. der Typ und Name der Variablen festgelegt → Richtig
 - ii. der Variablen automatisch ein Anfangswert zugewiesen → Falsch
 - iii. der entsprechende Speicherplatz für die Variable reserviert → Richtig
- f) Zur Definition einer Variablen, die einmal initialisiert wird und später nicht mehr verändert werden soll, wird das Schlüsselwort const verwendet.
- g) In C++ ist der Index des **ersten** Arrayelements 0.
- h) In C++ ist der Index des **letzten** Arrayelements N - 1.

Aufg. 2.01: Verständnisfragen Lsg. (3)

i) Benennen Sie alle Fehler im folgenden Codeausschnitt:

```
float index = 0;
float[5] myArray =
    {0, 1, 2, 3, 4, 5};
myArray[index] = 10;
```

```
int index = 0;
float myArray[6] =
    {0, 1, 2, 3, 4, 5};
myArray[index] = 10;
```

j) Nach Ausführung der folgenden Anweisungen speichert die Variable **x** den Wert:

→ 10

```
int x = 1, y = 10;
if( y > 0 ) {
    if( y < 5 ) {
        x = 5;
    } else {
        x = 10;
    }
}
```

Aufg. 2.01: Verständnisfragen Lsg. (4)

k) Nach Ausführung der folgenden Schleife speichert die Variable **x** den folgenden Wert:

- i. 1 → Falsch
- ii. 2 → Richtig
- iii. 3 → Falsch

```
int x = 0, y = 1;  
while( ++y < 3 ) {  
    x += y;  
}
```

- Aufg. 2.01: Verständnisfragen



Aufg. 2.02: Definition und Initialisierung von Variablen Lsg.

Aussage	Zulässig	Sinnvoll	Bemerkung
<code>int a = 23;</code>	✓	✓	Normale Zuweisung
<code>int b = 5.7;</code>	✓	✗	b hat danach den Wert 5
<code>char c = 300;</code>	✓	✗	Wert ist zu groß für ein Byte
<code>double f = 1.2E4;</code>	✓	✓	f enthält die Zahl 12000
<code>float d = 9 / 4;</code>	✓	✗	da 9 und 4 Ganzzahlen sind, ist das Ergebnis der Division 2
<code>char g = '\\';</code>	✓	✓	\\ ' ist ein Zeichen
<code>double äquatum = 4;</code>	✗	✗	ä ist im Namen nicht erlaubt
<code>const float epi;</code>	✗	✗	Initialisierung fehlt
<code>float e = 9.0 / 4;</code>	✓	✓	9.0 wird als double erkannt und das Ergebnis der Division ist somit 2.25
<code>short h = 32769;</code>	✓	✗	Überschreitet die Grenzen von short h hat dann einen Wert von -32767

- Aufg. 2.02: Definition und Initialisierung von Variablen



Aufg. 2.03: Gültigkeitsbereich Lsg.

Gegeben sei der folgende Programmausschnitt:

```
double fq = 0.6180339887;    //Fibonacci-Quotient
```

```
int b = 10000;
```

b und **fq** sind globale Variable: stehen im ganzen Programm allen Funktionen zur Verfügung

```
int hash ( unsigned int x ) {  
    double temp = x * fq - (int) ( x * fq );  
    return b * temp;  
}
```

x, Übergabeparameter → lokale Variable: nur innerhalb der Funktion **hash()** verfügbar

Wegen $0 \leq \text{temp} < 1$ und $b = 10000$
→ $0 \leq b * \text{temp} \leq 9999$

temp, lokale Variable: nur innerhalb der Funktion **hash()** verfügbar

- In welchem Bereich des Programms kann auf die Variablen **fq**, **b**, **x** und **temp** zugegriffen werden?
- Welchen Wertebereich haben die Return-Werte der Funktion **hash()**?

- Aufg. 2.03: Gültigkeitsbereich



Fragen?

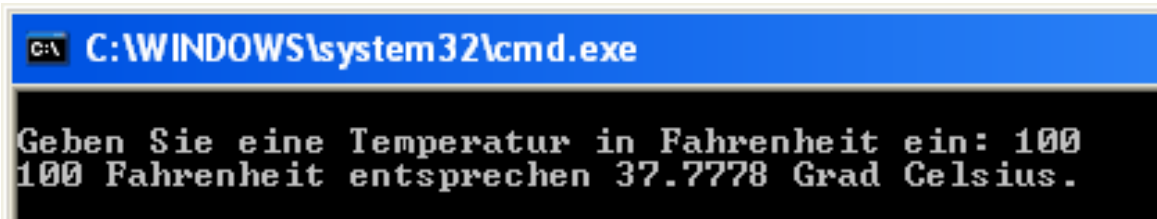
Aufg. 2.04: Ein- und Ausgabe

Schreiben Sie ein C++ Programm, das einen Temperaturwert in der Einheit Fahrenheit im Dialog einliest und in die Einheit Celsius umrechnet und ausgibt.

Hinweis: Verwenden Sie zum Umrechnen die folgende Formel:

$$5 \cdot (\text{Fahrenheit} - 32) = 9 \cdot \text{Celsius}$$

Beispielausgabe:



```
C:\WINDOWS\system32\cmd.exe

Geben Sie eine Temperatur in Fahrenheit ein: 100
100 Fahrenheit entsprechen 37.7778 Grad Celsius.
```

Aufg. 2.04: Ein- und Ausgabe Lsg. (1)

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

Anlegen und Initialisieren der Variablen

```
int main() {
```

```
double cels = 0.0, fahr = 0.0;
```

Ausgabe des Textes

```
cout << "Geben Sie eine Temperatur in Fahrenheit ein: ";
```

```
cin.sync();
```

```
cin.clear();
```

```
cin >> fahr;
```

Einlesen einer Eingabe von der Tastatur und diese speichern in die Variable **fahr**

```
cels = 5.0 * ( fahr - 32.0 ) / 9.0;
```

Berechnen der Temperatur in Celsius und das Ergebnis speichern in die Variable **cels**

```
cout << fahr << " Fahrenheit entsprechen "
     << cels << " Grad Celsius." << endl;
```

```
return 0;
```

```
}
```

Ausgabe des Ergebnisses auf der Kommandozeile

- Aufg. 2.04: Ein- und Ausgabe



Aufg. 2.05: Mehrfachentscheidungen Lsg.

Transformieren Sie den folgenden Codeausschnitt unter Anwendung einer **switch**-Anweisung und vermeiden Sie dabei redundanten Code zur Ausgabe:

```
unsigned int number;
cout << "Geben Sie eine Zahl zwischen 1 und 5 ein: ";
cin >> number;

if( number == 0 ) {
    cout << "Eingabe zu klein!\n";
} else if( number == 1 ) {
    cout << "Eingabe ok und ungerade\n";
} else if( number == 2 ) {
    cout << "Eingabe ok und gerade\n";
} else if( number == 3 ) {
    cout << "Eingabe ok und ungerade\n";
} else if( number == 4 ) {
    cout << "Eingabe ok und gerade\n";
} else if( number == 5 ) {
    cout << "Eingabe ok und ungerade\n";
} else {
    cout << "Eingabe zu gross!\n";
}
```

```
switch( number ) {
case 0:
    cout << "Eingabe zu klein!\n";
    break;
case 1:
case 3:
case 5:
    cout << "Eingabe ok und ungerade\n";
    break;
case 2:
case 4:
    cout << "Eingabe ok und gerade\n";
    break;
default:
    cout << "Eingabe zu gross!\n";
}
```

- Aufg. 2.05:
Mehrfachentscheidungen



Aufg. 2.06: Anwendung von Kontrollstrukturen

Lsg. (1)

Schreiben Sie die erforderlichen Anweisungen, um

- a) zwei Gleitpunktzahlen im Dialog einzulesen und die größere Zahl von der kleineren zu subtrahieren und das Ergebnis auszugeben.

```
double zahl1; double zahl2;
cout << "Bitte geben Sie die erste Zahl ein:";
cin >> zahl1;
cout << "Bitte geben Sie die zweite Zahl ein:";
cin >> zahl2;
if( zahl1 < zahl2 ) {
    cout << endl << "Ergebnis: " << zahl1 - zahl2 << endl;
} else {
    cout << endl << "Ergebnis: " << zahl2 - zahl1 << endl;
}
```

Aufg. 2.06: Anwendung von Kontrollstrukturen

Lsg. (2)

Hinweis: Bitte beachten Sie, dass Sie auch Schleifen nur mit Bedingungen und ohne Anweisungen schreiben können.

- b) Gleitpunktzahlen einzulesen und aufzusummieren, bis ihre Summe 100 überschreitet.

```
double x = 0.0; double sum = 0.0;
while( cin >> x && ( sum += x ) < 100.0 )
;
```

`cin` hat einen Rückgabewert, welcher angibt, ob das Einlesen erfolgreich war bzw. ob der eingelesene Wert zum Variablentyp passt

Alternativlösung:

```
double x = 0.0;
double sum = 0.0;

while( sum <= 100.0 ) {
    cin >> x;
    sum = sum + x;
}
```

- c) positive Ganzzahlen solange einzulesen, bis eine negative Zahl eingegeben wird.

```
double n = 0.0;
while( cin >> n && n >= 0.0 )
;
```

Aufg. 2.06: Anwendung von Kontrollstrukturen

Lsg. (3)

- d) den Anwender aufzufordern, drei verschiedene Ganzzahlen einzugeben. Die Aufforderung wird wiederholt, solange zwei von drei Zahlen übereinstimmen.

```
int n1 = 1; int n2 = 2; int n3 = 3;
cout << "Geben Sie drei verschiedene Ganzzahlen ein." << endl;
cout << "Die erste Ganzzahl: ";
cin >> n1;
do {
    cout << "\nDie zweite Ganzzahl: ";
    cin >> n2;
} while( n1 == n2 );
do {
    cout << "\nDie dritte Ganzzahl: ";
    cin >> n3;
} while( n1 == n3 || n2 == n3 );
cout << "Ihre Eingabe: " << n1 << ", " << n2 << ", " << n3 << endl;
```

- Aufg. 2.06: Anwendung von Kontrollstrukturen



Aufg. 2.07: Mehrdimensionale Arrays und Schleifen

Erstellen Sie ein C++ Programm, das im Dialog eine 3x3 Matrix einliest, ihre Determinante berechnet und das Ergebnis auf dem Bildschirm ausgibt. Verwenden Sie zur internen Speicherung der Matrix ein mehrdimensionales Array.

Hinweis:

$$\det A = \det \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \\ = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31} - a_{12}a_{21}a_{33} - a_{11}a_{23}a_{32}.$$

Beispielausgabe:

```
C:\Windows\system32\cmd.exe
Zeile 1, Spalte 1: 11
Zeile 1, Spalte 2: 12
Zeile 1, Spalte 3: -13
Zeile 2, Spalte 1: 21
Zeile 2, Spalte 2: -22
Zeile 2, Spalte 3: 23
Zeile 3, Spalte 1: -31
Zeile 3, Spalte 2: 32
Zeile 3, Spalte 3: 33

Eingegebene Matrix:
11      12      -13
21      -22      23
-31      32      33

Die Determinante betraegt: -32824
```

Aufg. 2.07: Mehrdimensionale Arrays und Schleifen Lsg. (1)

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

2-dim. Array zur Speicherung der
einggegebenen Matrix

```
int main() {
    double matrix[3][3] = { { 0, 0, 0 }, { 0, 0, 0 }, { 0, 0, 0 } };
    /* Einlesen der Matrix */
    for( int zeile = 1; zeile <= 3; ++zeile ) {
        for( int spalte = 1; spalte <= 3; ++spalte ) {
            cout << "Zeile " << zeile << ", Spalte " << spalte << ": " ;
            cin >> matrix[zeile - 1][spalte - 1];
        }
    }
    /* optionale Ausgabe der Matrix, hilfreich zum Ueberpruefen mit
       http://www.arndt-bruenner.de/mathe/determinanten.htm */
    cout << endl << "Eingegebene Matrix:" << endl;
    for( int zeile = 1; zeile <= 3; ++zeile ) {
        for( int spalte = 1; spalte <= 3; ++spalte ) {
            cout << matrix[zeile - 1][spalte - 1] << "\t";
        }
        cout << endl;
    }
}
```

Speichern der Daten im Array ab Index 0!

Aufg. 2.07: Mehrdimensionale Arrays und Schleifen Lsg. (2)

```
/* Berechnung */
double determinante = matrix[0][0] * matrix[1][1] * matrix[2][2];
determinante += matrix[0][1] * matrix[1][2] * matrix[2][0];
determinante += matrix[0][2] * matrix[1][0] * matrix[2][1];
determinante -= matrix[0][2] * matrix[1][1] * matrix[2][0];
determinante -= matrix[0][1] * matrix[1][0] * matrix[2][2];
determinante -= matrix[0][0] * matrix[1][2] * matrix[2][1];

/* Ausgabe */
cout << endl << "Die Determinante betraegt: " << determinante << endl;

return 0;
}
```

- Aufg. 2.07: Mehrdimensionale Arrays und Schleifen



Vielen Dank für Ihre Aufmerksamkeit



Marc Weber
Karlsruher Institut für Technologie (KIT) – ITIV
marc.weber3@kit.edu