

# **Algorithmen und Programme Übung**

Protokolliert von Rouven Czerwinski

Version vom 17. November 2011

## Inhaltsverzeichnis

<b>1 Übung 1 (3.11.2011)</b>	<b>3</b>
1.1 Aufgabe 1 . . . . .	3
1.2 Einführung in C++ . . . . .	4
1.2.1 Wertezuweisung . . . . .	5
1.2.2 Gültigkeitsbereich von Variablen . . . . .	5
1.2.3 Ein- und Ausgabe in der Konsole . . . . .	6
<b>2 Übung 2 (10.11.2011)</b>	<b>7</b>
2.1 Vergleichs- und Verknüpfungsoperatoren . . . . .	7
2.2 Definition von Funktionen (Unterprogramme) . . . . .	8
2.3 Vom Quellcode zum ausführbaren Programm . . . . .	8
2.4 Schleifen . . . . .	9
<b>3 Übung 3 (17.11.2011)</b>	<b>10</b>
3.1 Aufgabe 3 . . . . .	11

## Tabellenverzeichnis

# 1 Übung 1 (3.11.2011)

Benutzername: aupvz

Passwort: aupvz1112

## 1.1 Aufgabe 1

Terminierend: Algorithmus hält an

Determiniert: Algorithmus liefert bei selber Eingabe selbes Ergebnis

Deterministisch: Alg. liefert bei selber Eingabe selbes Ergebnis über die selben Zwischenergebnisse

	terminierend	determiniert	deterministisch
a)	ja	ja	ja
b) Zahl $\notin \{-1, 0, 1\}$	ja	nein	nein
Zahl $\in \{-1, 0, 1\}$	nein	nein	nein

a) Terminierend?

=> Keine Schleifen

=> Algorithmus hält direkt an

=> terminierend

Determiniert?

-> Eindeutig reproduzierbares Ergebnis für sämtliche Eingabewerte?

$p \in \mathbb{Z}, q = 0 \Rightarrow \text{erg} = -1, q \neq 0 \Rightarrow \text{erg} = p \% q$

=> determiniert

deterministisch?

- Gibt es Zwischenergebnisse? (falls nein -> deterministisch)

- Hier ja: Ergebnis der Abfrage  $q \neq 0$

- Zwischenergebnis immer gleich bei gleichem q

=> deterministisch

b)

- Rechner wählt beliebige Zahl
  - > Zufallszahl zwischen -10 und 20
  - > nicht determiniert
  - > nicht deterministisch
- Trotzdem terminierend?
- Bsp für Zahl = 2
  1. Zahl = 2
  2. Zahl = 2 \* 2 = 4
  3. Zahl > 10? nein => gehe zu 2
  2. Zahl = 4 \* 4 = 16
  3. Zahl > 10? ja => Ausgabe und Ende=> terminierend
- Aber: Abbruchbedingung der Schleife wird nur erreicht, wenn Zahl im Laufe des Alg. wächst, dies gilt nicht für die Zahl  $\in \{-1, 0, 1\}$  => dann nicht terminiert

## 1.2 Einführung in C++

Datentypen

Anlegen einer Variable

Datentyp	Bedeutung	Beispiel
int	Integer-Zahl (ganze Zahl)	-2;45
float	Gleitkommazahl	-4.342;7.543
char	Character (Zeichen)	'a'; 'C'; '?'; '7'
bool	Wahr-Falsch-Variable	true, false
string	Zeichenkette	"Wort"; "2plus 3=5"

```
Datentyp Variablenname;
```

Wichtig: Jede Anweisung muss mit einem Semikolon abgeschlossen werden.

Bsp:

```
float kommazahl;
```

Zu beachten bei Namensgebung

- C++ ist "case sensitiv", dh. Groß- und Kleinschreibung wird beachtet

```
=> int zahl; und int Zahl; // sind zwei verschiedene Variablen
```

- Keine Sonderzeichen: bool grüner100€schein; geht nicht!
- Keine reservierten Worte: char string;
- Keine Nummern am Anfang: int 5teZahl; geht nicht!

### 1.2.1 Wertezuweisung

```
int i; // dies ist ein Kommentar
i = 3; // i wird der Wert 3 zugewiesen
int k;
k = i; // k ist ebenfalls 3
k = i + 5; // k = 3+5 = 8
k = k*k; // k = 8*8 = 64
k = k/i; // k = 64/3 = 21 Achtung:ganzzahlige Division
k = k*(k+2); // k = 3*(21+2) = 69
k = k%50; // k = 69%50 = 19
float f;
f = 1;
f = 1/3; // f=0.33333...
char c;
c = '?';
string s;
s = "Text";
s = s + "zeile"; // s = "Textzeile"
bool b;
b = true;
b = !b; // Negation von b => b = nicht true => b = false
```

### 1.2.2 Gültigkeitsbereich von Variablen

- Bereiche werden durch {} gekennzeichnet
- Variablen sind nur innerhalb desjenigen Bereichs gültig, in dem sie deklariert wurden
- außerhalb jeden Bereichs deklarierte Variablen heißen global und sind überall verfügbar
- Variablen sind nur NACH ihrer Deklaration verfügbar

Bsp:

```
int weltweit = 4; //global
{
    int a;
    {
        a = 1;
    }
}
```

```

    int b = 2;
    a = weltweit + b;
}
int c;
c = a + weltweit;
c = a + b; //UNGUELTIG!, da b in diesem Bereich nicht bekannt ist
}

```

### 1.2.3 Ein- und Ausgabe in der Konsole

- Nutzung der Funktionen cout (Ausgabe) und cin (Eingabe)
- Einbindung von <iostream> nötig
- Befehle müssen `using namespace std;` freigeschaltet werden
- Werte werden mittels << und >> übergeben

Bsp:

```

int zahl;
cout << "Bitte Zahl eingeben:";
cin >> zahl;
cout << "Das Quadrat lautet" << zahl*zahl << endl; // endl -> Zeilenumbruch

```

## 2 Übung 2 (10.11.2011)

### 2.1 Vergleichs- und Verknüpfungsoperatoren

Operator	Bedeutung
<	kleiner
>	größer
>=	größer gleich
==	gleich
!=	ungleich
&&	logisches UND
	logisches ODER

Nutzung z.B. in if Abfragen

```
if (Bedingung) // WENN Bedingung erfuehlt
{
    Anweisung; // fuehre Anweisung aus
}

if (3 > 5)
{
    cout >> "PC kaputt" << endl;
}

int a = 6;
if (zahl == a)
{
    cout << "Zahl gleich" << a << endl;
}
else // wird ausgefuehrt wenn die Bedingung nicht erfuehlt ist.
{
    cout << "Zahl ist ungleich" << a << endl;
}

int b = 9;
if (zahl > a && zahl > b)
{
    cout << "Zahl ist am Groessten" << endl;
}

bool bigger;
bigger = zahl > b;
if (bigger || zahl < b)
{
    cout << "Zahl ungleich" << b << endl;
}
else
{
    cout << "Zahl gleich" << b << endl;
}
```

## 2.2 Definition von Funktionen (Unterprogramme)

```
Rueckgabetyyp Funktionsname(Parameter1,Parameter2,...)
{
    Anweisung;
    return rueckgabewert;    // muss vom Datentyp Rueckgabetyyp sein
}

int Addiere(int a, int b)
{
    int sum;
    sum = a +b;
    return sum;
}

void Ausgabe(int zahl)
{
    cout << "Zahl ist" << zahl << endl;
}
```

Funktionen vom Typ void haben keinen Rueckgabewert und somit keine **return** Anweisung.

Aufbau eines C++-Programms:

```
// Einbinden von Bibliotheken
#include <iostream>
using namespace std; // noetig fuer cout und cin
// Funktionsdefinitionen
int Addiere(int a, int b)
{
    int sum;
    sum = a +b;
    return sum;
}

void Ausgabe(int zahl)
{
    cout << "Zahl ist" << zahl << endl;
}

// Hauptprogramm
int main()
{
    int zahl1 = 4;
    int zahl2 = 5;
    int summe;
    summe = Addiere(zahl1,zahl2);
    Ausgabe(summe);
    return 0;    // Im Hauptprogramm nicht notwendig
}
```

## 2.3 Vom Quellcode zum ausführbaren Programm

Quellcode(Text) → kompilieren (Übersetzung in Maschinsprache) → Compilerfehler?

Falls ja, muss der Quellcode überprüft werden.

Falls Nein (syntaktische Korrektheit des Programms) → Programm ausführen → Laufzeitfehler?



Falls ja, neuerliche Bearbeitung des Quellcodes.  
Falls nein, Programmende  
(evtl. mit TikZ in Blockdiagramme umsetzen)

## 2.4 Schleifen

Schleifen werden genutzt, um Anweisungen/Blöcke mehrmals auszuführen.

**for-Schleife:** Verwendung, wenn Anzahl der Durchläufe bekannt

```
for (Startanweisung, Bedingung, Zaehlanweisung)
{
    Anweisung; // fuehre aus, solange Bedingung erfuehlt
}

int z = 0;
for (int i = 1, i < 10, i++)
{
    z = z + i;
    cout << z << endl;
}
```

Schleifendurchlauf	i	z
1	1	1
2	2	3
3	3	6
...	...	...
9	9	$36 + 9 = 45$

Danach Abbruch, weil i nicht mehr kleiner als 10 ist.

**while-Schleife:** Verwendung, wenn Anzahl der Durchläufe nicht bekannt.

```
while (Bedingung)
{
    Anweisung; // SOLANGE WIE Bedingung ERFUELLT
}

int eingabe = 0;
while (engage != 8)
{
    cout << "Bitte 8 eingeben!" << endl;
    cin >> eingabe;
}
cout << "Danke, Du hast " << eingabe << " eingegeben";
```

### 3 Übung 3 (17.11.2011)

Wertebereich	terminierend	determinierend	deterministisch
$a > 20$	ja	ja	nein
$10 < a \leq 20$	ja	nein	nein
$2 \leq a \leq 10$	ja	ja	ja
$1 < a < 2$	ja	ja	ja
$a \leq 1$	nein	nein	nein

Fallunterscheidungen nötig

- $a > 20$ :
  - 1. +2. if-Bedingung erfüllt
  - Zufallszahl wird erst abgezogen, dann wieder hinzuaddiert  
 $\Rightarrow$  terminierend, determiniert, nicht deterministisch
- $10 < a \leq 20$ :
  - nur 1. if-Bedingung erfüllt
  - Zufallszahl wird hinzuaddiert  
 $\Rightarrow$  terminierend, nicht determiniert, nicht deterministisch
- $10 \leq a$ :
  - if-Bedingung nicht erfüllt
  - while-Schleife nur für  $a < 2$   
 $\Rightarrow$  für  $2 \leq a \leq 10$  wird nur der Eingabewert zurückgegeben  
 $\Rightarrow$  terminierend, determiniert, deterministisch
- $1 < a < 2$ :
  - a wächst bis  $a \Rightarrow 2$ , dann wird Schleife beendet  
 $\Rightarrow$  terminierend, determiniert, deterministisch
- $a \leq 1$ :
  - a muss wachsen, damit Abbruchbedingung erfüllt wird  
 $\Rightarrow$  nur für  $a > 1$  der Fall  
 $\Rightarrow$  nicht terminierend, nicht determiniert, nicht deterministisch

### 3.1 Aufgabe 3

#### 1. Durchschnitt

a) Eingabewerte: Start- und Endwert des Wertebereichs  
Rückgabewerte: Durchschnittswert, ganze Zahl  
⇒ Funktionskopf: `int Durchschnitt(int start, int end)`

b) Vorgehen:

- i. Aufsummieren aller Zahlen von `start` bis `end`
- ii. Bestimmen der Anzahl der Summanden
- iii. Teilen der Summe durch die Anzahl
- iv. Rückgabe des Ergebnisses

Nötige Hilfsmittel:

- Variable zur Speicherung der Summe
- Schleife (for oder while)
- Variable für die Anzahl

a) Lösung mittels for-Schleife:

```
int Durchschnitt_FOR(int start, int end)
{
    int summe = 0; // Initialisierung mit 0
    for (int i=start; i<=end; i=i+1)
    {
        summe =summe + i;
    }
    int anzahl = end - start;
    int schnitt = summe / Anzahl;
    return schnitt;
}
```

Lösung mittels while-Schleife:

```
int Durchschnitt_WHILE(int start, int end)
{
    int summe = 0; // Initialisierung mit 0
    int anzahl = end - start;

    int i=start;
    while (i<=end)
    {
        summe = summe + i;
        i = i +1;
        Anzahl = Anzahl + 1;
    }
    int schnitt = summe / Anzahl;
    return schnitt;
}
```

Hauptprogramm:

```
int main()
{
    int ds = Durchschnitt_FOR(4, 9);
    cout << "Durschnitt ist " << ds << endl;
}
```

## 2. Quersumme

Quersumme(538) =  $5 + 4 + 8 = 16$ ;

a) Problem: Wie können die Ziffern separiert werden?

Lösung: Nutzung des Operators %10 und /10

b) Vorgehen:

Aufsummieren von hinten:

$QS(538) = 8 + QS(53) = 8 + 3 + QS(5) = 8 + 3 + 5 + QS(0)$

Algorithmus: Solang die Zahl größer als Null:

i. Ermitteln und Aufsummieren der letzten Ziffer

ii. Abtrennen des letzten Ziffer

Nötige Hilfsmittel:

- Variable für Summe
- Schleife (bis Zahl == 0)
- Modulo Operator

c) Funktionskopf:

i. zu übergeben: Ganze Zahl  $\Rightarrow$  int zahl

ii. Rückgabe: Quersumme  $\Rightarrow$  int summe

```
int Quersumme(int zahl)
```

d) Programm:

```
int Quersumme(int zahl)
{
    int summe = 0;
    while (zahl > 0)
    {
        int Ziffer = zahl % 10;
        summe = summe + ziffer;
        zahl = zahl / 10;
    }
    return summe;
}
```

Ablauf Vor der Schleife:  $\text{summe} = 0$ ,  $\text{zahl} = 538 \Rightarrow$  Abbruch, da  $\text{zahl}$

Durchlauf	Ziffer	summe	zahl
1.	$538 \% 10 = 8$	$0 + 8 = 8$	$538 / 10 = 53$
2.	$53 \% 10 = 3$	$8 + 3 = 11$	$53 / 10 = 5$
3.	$5 \% 10 = 5$	$11 + 5 = 16$	$5 / 10 = 0$

nicht mehr  $> 0 \Rightarrow$  Rückgabe der Summe