Inhaltsverzeichnis

Übung_if

Seien n1, n2, n3, n4, n5, max, und betrag Variablen des Typs int. Gehen Sie davon aus, dass diese Variablen schon vereinbart wurden und irgendwelche vernünftigen Werte enthalten. Lassen Sie den Ausführer alle Berechnungen mit Ganzzahlen durchführen und nicht mit Bruchzahlen. Befehlen Sie mit geeigneten if-Anweisungen dem Ausführer, die folgenden Arbeiten zu erledigen:

1. Falls die Variable n1 einen negativen Wert enthält, soll ihr der Wert 0 zugewiesen werden.

Wie macht man das (eleganter?) ohne if-Anweisung (nur mit einer Zuweisung)? Sie dürfen dabei die Funktion max verwenden, die zwei int-Parameter hat und den grösseren ihrer beiden Parameter als Ergebnis liefert (z.B. ist max(10, 15) gleich 15 und max(-3, -5) ist gleich -3).

2. Falls die Variable n2 einen positiven Wert enthält, soll ihr Wert um 10 Prozent erhöht werden (Ganzzahlrechnung!) und dann zusammen mit dem Text "Der Wert von n2 ist jetzt gleich " zur Standardausgabe ausgegeben werden (mit dem Befehl cout << ... << ... << endl;).

3. Falls die Variable n3 einen geraden Wert enthält, soll ihr Wert halbiert werden. Sonst soll ihr Wert verdoppelt werden. Um festzustellen, ob n3 gerade ist oder nicht, sollten Sie den Restoperator % verwenden.

4. Wie kann man die vorige Aufgabe lösen ohne einen Restoperator (%) zu benützen?

5. Die beiden Zahlen n1 und n2 sollen (jede auf einer eigenen Zeile) ausgegeben werden, und zwar in aufsteigend sortierter Reihenfolge (d.h. zuerst die kleinere und dann die grössere Zahl. Falls die beiden Zahlen gleich sind, können sie in beliebiger Reihenfolge ausgegeben werden).

6. Falls die Variable n1 den Wert 0 enthält, soll sie unverändert bleiben. Sonst soll ihr Wert in Richtung 0 um 1 verändert werden (z.B. soll der Wert -5 durch -4 ersetzt werden oder +7 durch +6).

7. Falls die Variable n2 den Wert 0 enthält, soll sie unverändert bleiben. Sonst soll ihr Wert um 1 vermindert werden (z.B. soll der Wert -5 durch -6 ersetzt werden oder +7 durch +6).

8. Die grösste der drei Zahlen n1 bis n3 soll der Variablen max zugewiesen werden.

9. Die grösste der fünf Zahlen n1 bis n5 soll der Variablen max zugewiesen werden.

10. Angenommen, die Variable betrag enthält einen Rechnungsbetrag. Wenn dieser Betrag grösser als 100 (aber nicht grösser als 500) ist, soll er um 3 Prozent (Rabatt) vermindert werden (Ganzzahlrechnung!). Falls der Betrag grösser als 500 (aber nicht grösser als 1000) ist soll er um 5 Prozent vermindert werden. Falls der Betrag grösser als 1000 ist, soll er um 6 Prozent vermindert werden. Gestalten Sie Ihre Lösung möglichst so, dass der Kollege2 sie leicht um zusätzliche Rabattstufen erweitern kann (z.B. 8 Prozent für Beträge über 10.000,- DM). Dieses Problem kann man mit einer einzigen (relativ komplizierten) if-Anweisung oder mit mehreren (relativ einfachen) if-Anweisungen lösen. Welche der beiden Lösungen finden Sie besser? Warum?

11. Führen Sie das Programm IfSwitch01 (im Skript, Abschnitt 5.) mit Papier, Bleistift und Radiergummi aus und nehmen Sie dabei an, dass der Benutzer den Wert 3 eingibt (siehe Zeile 20 des Programms). Ignorieren Sie dabei die erste if-Anweisung (in Zeile 12 bis 16). Führen Sie das Programm erneut aus mit der Eingabe 6.

Lösung if

```
1   // ---------- Teilaufgabe 1. ---------------------------------------
2      if (n1 < 0) n1 = 0;   // mit  if-Anweisung
3      n1 = std::max(n1, 0); // ohne if-Anweisung (eleganter?)
4
5   // ---------- Teilaufgabe 2. ---------------------------------------
6   if (n2 > 0) {
7      n2 = n2 * 110 / 100;
8      cout << "Der Wert von n2 ist jetzt gleich " << n2 << endl;
9   }
10
11   // ---------- Teilaufgabe 3. ---------------------------------------
12   if (n3 % 2 == 0) {
13      n3 = n3 / 2; // oder: n3 /= 3;
14   } else {
15      n3 = n3 * 2; // oder: n3 *= 2;
16   }
17
18   // ---------- Teilaufgabe 4. ---------------------------------------
19   if (n3 - (n3 / 2) * 2 == 0) {
20      n3 = n3 / 2; // oder: n3 /= 2;
21   } else {
22      n3 = n3 * 2; // oder: n3 *= 2;
23   }
24
25   // ---------- Teilaufgabe 5. ---------------------------------------
26   if (n1 <= n2) {
27      cout << "n1: " << n1 << endl;
28      cout << "n2: " << n2 << endl;
29   } else {
30      cout << "n2: " << n2 << endl;
31      cout << "n1: " << n1 << endl;
32   }
33
34   // ---------- Teilaufgabe 6. ---------------------------------------
35   if (n1 > 0) n1 = n1 + 1;
36   if (n1 < 0) n1 = n1 - 1;
37
38   // ---------- Teilaufgabe 7. ---------------------------------------
39   if (n1 != 0) n1 = n1 - 1;
40
41   // ---------- Teilaufgabe 8. ---------------------------------------
42   max = n1;
43   if (max < n2) max = n2;
44   if (max < n3) max = n3;
45
46   // ---------- Teilaufgabe 9. ---------------------------------------
47   max = n1;
48   if (max < n2) max = n2;
49   if (max < n3) max = n3;
50   if (max < n4) max = n4;
51   if (max < n5) max = n5;
52
53   // ---------- Teilaufgabe 10., Lösung A ----------------------------
54   if         (betrag > 1000) {
55      betrag = betrag * 94 / 100; // 6 Prozent Rabatt
56   } else if  (betrag >  500) {
57      betrag = betrag * 95 / 100; // 5 Prozent Rabatt
58   } else if  (betrag >  100) {
59      betrag = betrag * 97 / 100; // 3 Prozent Rabatt
60   }
61
62   // ---------- Teilaufgabe 10., Lösung B ----------------------------
63   if (                 betrag <=  100) ; // Nur zur Vollstaendigkeit
```

```
64   if ( 100 < betrag && betrag <=  500) betrag = betrag * 97 / 100;
65   if ( 500 < betrag && betrag <= 1000) betrag = betrag * 95 / 100;
66   if (1000 < betrag                  ) betrag = betrag * 94 / 100;
67
68  // ---------- Teilaufgabe 11. ---------------------------------------
69  /* Ausgabe des Programms IfSwitch01 fuer die Eingabe 3 (bzw. 6):
70
71  Programm IfSwitch01: Jetzt geht es los!
72  Bitte eine Ganzzahl eingeben: 3
73  Sie haben eine ungerade Zahl eingegeben!
74  Ihre Zahl liegt zwischen 1 und 100!
75  Ihre Eingabe liegt zwischen 2 und 5!
76  Programm IfSwitch01: Das war's erstmal!
77
78  Programm IfSwitch01: Jetzt geht es los!
79  Bitte eine Ganzzahl eingeben: 6
80  Sie haben eine gerade Zahl eingegeben!
81  Ihre Zahl liegt zwischen 1 und 100!
82  Sie haben eine komische Zahl eingegeben!
83  Programm IfSwitch01: Das war's erstmal!
```

Übung Schleifen ausführen

Führen Sie die folgenden Befehlsfolgen mit Papier, Bleistift und Radiergummi aus und geben Sie genau an, welche Zeichen und Zeilen zur Standardausgabe (d.h. zum Bildschirm) ausgegeben werden.

```
1   // --- 1. Die anna-Schleife: ---
2   int anna = 7;
3   while (anna > 0) {
4     cout << anna << " ";
5     anna /= 2;
6   }
7   cout << endl;;
8
9   // --- 2. Die berta-Schleife: ---
10  int berta = 7;
11  do {
12     berta /= 2;
13     cout << " -> " << berta;
14  } while (berta > 0);
15  cout << endl;;
16
17  // --- 3. Die celia-Schleife: ---
18  for (int celia = -3; celia < 5; celia += 2) {
19     cout << 2 * celia + 4 << " ";
20  }
21  cout << endl;;
22
23  // --- 4. Die dora-Schleife: ---
24  for (int dora = 3; 2*dora > -3; dora--) {
25     cout << dora << " ";
26  }
27  cout << endl;;
28
29  // --- 5. Die MÁX1-Schleife: ---
30  int const MAX1 = 3;
31  for (int i1 = 1; i1 <= MAX1; i1++) {
32     for (int i2 = 1; i2 <= 2*MAX1; i2++) {
33        cout << "*";
34     }
35     cout << endl;;
36  }
37
38  // --- 6. Die MAX2-Schleife: ---
39  int const MAX2 = 5;
40  for (int i1 = 1; i1 <= MAX2; i1++) {
41     for (int i2 = 1; i2 <= i1; i2++) {
42        cout << "++";
43     }
44     cout << endl;;
45  }
```

Lösung Schleifen ausführen

1. Die Ausgabe der **anna**-Schleife : **7 3 1**

2. Die Ausgabe der **berta**-Schleife: **-> 3 -> 1 -> 0**

3. Die Ausgabe der **celia**-Schleife: **-2 2 6 10**

4. Die Ausgabe der **dora**-Schleife : **3 2 1 0 -1**

5. Die Ausgabe der **MAX1**-Schleife:

```
******
******
******
```

6. Die Ausgabe der **MAX2**-Schleife:

```
++
++++
++++++
++++++++
++++++++++
```

Übung Bojen1

1.Stellen Sie die folgenden C++-Variablen als Bojen dar.

Damit Ihre Lösung einigermaßen einfach mit anderen Lösungen vergleichbar wird, sollten Sie sich an folgende Regeln halten:

R1: Stellen Sie die Variablen in der angegebenen Reihenfolge dar (erst `susi`, dann `felix` etc.).
R2: Wenn Sie eine Adresse festlegen müssen, dann nehmen Sie die Zahlen 10, 20, 30, ... , genau in dieser Reihenfolge.

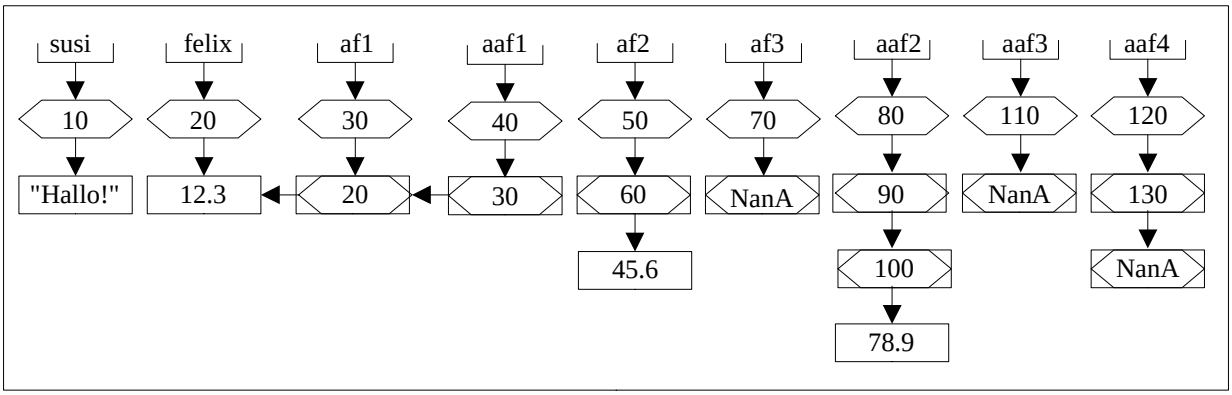
```
1   #define NanA 0
2   string    susi  = "Hallo!";
3   float     felix = 12.3;
4   float *   af1   = &felix;                        // Adr. von float 1
5   float * * aaf1  = &zaf1;                         // Adr. von Adr. von float 1
6   float *   af2   = new float(45.6);               // Adr. von float 2
7   float *   af3   = NanA;                          // Adr. von float 3
8   float * * aaf2  = new float *(new float(78.9)); // Adr. von Adr. von float 2
9   float * * aaf3  = NanA;                          // Adr. von Adr. von float 3
10  float * * aaf4  = new float *(NanA);             // Adr. von Adr. von float 4
```

2. Was wird durch die folgenden Befehle ausgegeben? Geben Sie Adressen genau so aus, wie Sie sie in der vorigen Aufgabe festgelegt haben, d.h. als Zahlen wie 10, 20, 30, ... etc.:

```
11  cout << "&susi:  " << &susi  << ", susi:   " << susi   << endl;
12  cout << "&felix: " << &felix << ", felix:  " << felix  << endl;
13  cout << "af1:    " << af1    << ", *af1:   " << *af1   << endl;
14  cout << "aaf1:   " << aaf1   << ", *aaf1:  " << *aaf1  <<
15                                  ", **aaf1: " << **aaf1 << endl;
16  cout << "af2:    " << af2    << ", *af2:   " << *af2   << endl;
17  cout << "af3:    " << af3                              << endl;
18  cout << "aaf2:   " << aaf2   << ", *aaf2:  " << *aaf2  <<
19                                  ", **aaf2: " << **aaf2 << endl;
20  cout << "aaf3:   " << aaf3                             << endl;
21  cout << "aaf4:   " << aaf4   << ", *aaf4:  " << *aaf4  << endl;
```

Lösung Bojen1

1. Die Variablen `susi`, `felix`, `af1`, ..., `aaf4` als Bojen dargestellt:



2. Die Ausgaben:

```
22  &susi:  10, susi:   Hallo!
23  &felix: 20, felix:  12.3
24  af1:    20, *af1:   12.3
25  aaf1:   30, *aaf1:  20, **aaf1: 12.3
26  af2:    60, *af2:   45.6
27  af3:    0
28  aaf2:   90, *aaf2:  100, **aaf2: 78.9
29  aaf3:   0
30  aaf4:   130, *aaf4: 0
```

3. Die Bojen in "nach links gekippter Darstellung":

```
31  |susi |--<0010>--[Hallo!]
32
33  |felix|--<0020>--[12.3]
34                     |
35  |af1  |--<0030>--[<0020>]
36                     |
37  |aaf1 |--<0040>--[<0030>]
38
39  |af2  |--<0050>--[<0060>]--[45.6]
40
41  |af3  |--<0070>--[<NanA>]
42
43  |aaf2 |--<0080>--[<0090>]--[<0100>]--[78.9]
44
45  |aaf3 |--<0110>--[<NanA>]
46
47  |aaf4 |--<0120>--[<0130>]--[<NanA>]
```

Übung Bojen2

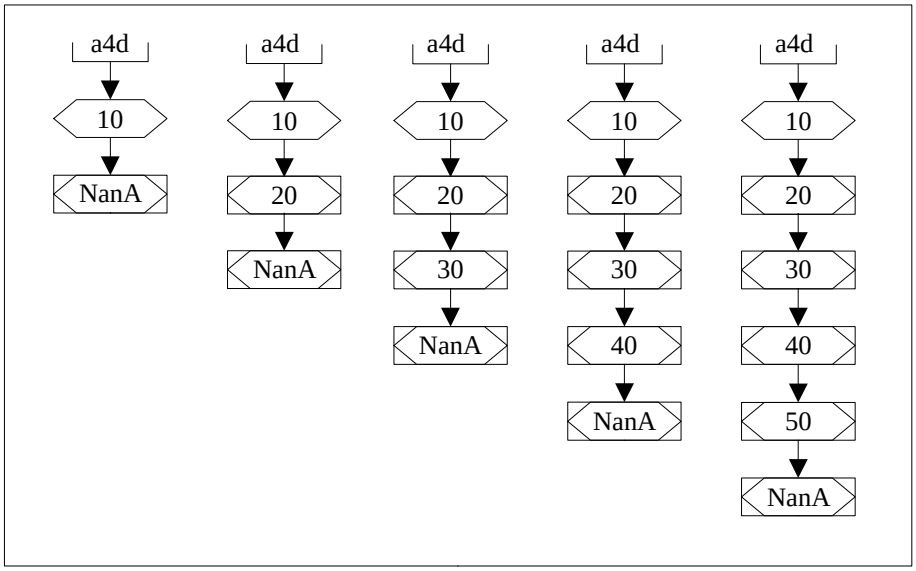
Betrachten Sie die folgende Vereinbarung einer Variablen namens a4d und die fünf Zuweisungen an diese Variable:

```
1      double * * * * a4d; // Adr. von Adr. von Adr. von Adr. von double
2
3            a4d =                    NanA;
4
5            a4d = new double * * * (NanA);
6
7          * a4d = new double * *   (NanA);
8
9        * * a4d = new double *     (NanA);
10
11     * * * a4d = new double       (12.345);
```

Wie sieht die Variable a4d als Boje dargestellt aus, nachdem der Ausführer die Zuweisung in Zeile 3 ausgeführt hat? Ebenso für die anderen vier Zuweisungen. Stellen Sie die Variablen a4d insgesamt also fünf Mal als Boje dar. Wenn Sie dabei eine Adresse festlegen müssen, dann verwenden Sie bitte die Zahlen 10, 20, 30 ... in dieser Reihenfolge (damit Ihre Lösung sich leichter mit anderen Lösungen vergleichen läßt).

Lösung Bojen2

Die fünf "Momentaufnahmen" der Variable `a4d`:



Die gleichen Bojen in der "nach links gekippten Darstellung" (die sich leichter per Programm erzeugen lässt):

```
1   |a4d|--< 10 >--[<NanA>]
2   |a4d|--< 10 >--[< 20 >]--[<NanA>]
3   |a4d|--< 10 >--[< 20 >]--[< 30 >]--[<NanA>]
4   |a4d|--< 10 >--[< 20 >]--[< 30 >]--[< 40 >]--[<NanA>]
5   |a4d|--< 10 >--[< 20 >]--[< 30 >]--[< 40 >]--[< 50 >]--[12.345]
```

Übung Adressen

```
1   #include <iostream>   // cout, cin, <<, >>, endl
2   #include <string>     // size, length, capacity, [], at, clear,
3   #include <vector>     // size, capacity, at, [], push_back, insert
4   using namespace std;
5
6   // Meldungen auf Deutsch:
7   string const           mldDE01 = "Fehler 1: Der Server ist gestoert!";
8   string const           mldDE02 = "Fehler 2: Kein Kaffee im Automat!";
9
10  // Meldungen auf Englisch:
11  string const           mldEN01 = "Error  1: The server is down!";
12  string const           mldEN02 = "Error  2: The Coffee machine is empty!";
13
14  string const * const   adrDE01 = &mldDE01; // Meldung 1 auf Deutsch
15  string const * const   adrDE02 = &mldDE02; // Meldung 2 auf Deutsch
16
17  string const * const   adrEN01 = &mldEN01; // Meldung 1 auf Englisch
18  string const * const   adrEN02 = &mldEN02; // Meldung 2 auf Englisch
19
20  string const * const * akt01; // Meldung 1 in der aktuellen Sprache
21  string const * const * akt02; // Meldung 2 in der aktuellen Sprache
22
23  vector<string const * const * const *> protokoll;
24  // ------------------------------------------------------------------------
25  void legSpracheFest(string const sprache) {
26     if        (sprache == "DE") {
27        akt01 = &adrDE01;
28        akt02 = &adrDE02;
29     } else if (sprache == "EN") {
30        akt01 = &adrEN01;
31        akt02 = &adrEN02;
32     } else {
33        cout << sprache << " ist keine erlaubte Sprache!" << endl;
34     }
35  } // legSpracheFest
36  // ------------------------------------------------------------------------
37  void erstelleProtokoll() {
38     protokoll.push_back(&akt02);
39     protokoll.push_back(&akt01);
40     protokoll.push_back(&akt02);
41  } // machWas
42  // ------------------------------------------------------------------------
43  void gibProtokollAus() {
44     cout << "-----------------------------------------------------\n";
45     for (int i=0; i<protokoll.size(); i++) {
46        cout << ***(protokoll.at(i)) << endl;
47     }
48  } // gibProtokollAus
49  // ------------------------------------------------------------------------
50  int main() {
51     erstelleProtokoll();
52     legSpracheFest("EN"); gibProtokollAus();
53     legSpracheFest("DE"); gibProtokollAus();
54  } // main
```

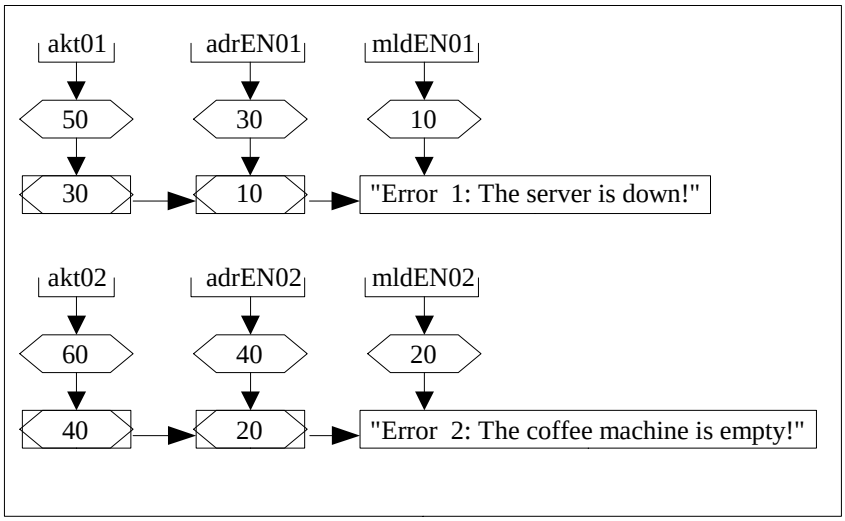
1. Was gibt dieses Programm `AdressUebung01` zur Standardausgabe (zum Bildschirm) aus?
2. Wie sehen die Variablen `akt01`, `akt02`, `adrEN01`, `adrEN02`, `mldEN01` und `mldEN02` nach Ausführung von Zeile 52 als Bojen aus?

Lösung Adressen

Die Ausgabe des Programms `AdressUebung01`:

```
1   ------------------------------------------------
2   Error  2: The Coffee machine is empty!
3   Error  1: The server is down!
4   Error  2: The Coffee machine is empty!
5   ------------------------------------------------
6   Fehler 2: Kein Kaffee im Automat!
7   Fehler 1: Der Server ist gestoert!
8   Fehler 2: Kein Kaffee im Automat!
```

Die Variablen `akt01`, `akt02`, `adrEN01`, `adrEN02`, `mldEN01` und `mldEN02` nach Ausführung von Zeile 52 als Bojen dargestellt:



Übung Klassen 1

Betrachten Sie die folgende Klassen-Definition:

```
1   class Zaehler {
2
3   private:
4      struct Zahl {
5         int            wert;                  // Die wichtige Komponente!
6         unsigned       anzahl;                // Hilfskomponente
7         string   const name;                  // Hilfskomponente
8         Zahl(int w, unsigned a, string n);    // Allgemeiner Konstruktor
9      }; // struct Zahl
10
11     Zahl * av;                               // Wird von den Konstruktoren
12                                              // initialisiert
13  public:
14     Zaehler();                               // Der Standard-Konstruktor
15     Zaehler(int z, string n="zaehler");      // Ein allgemeiner Konstruk.;
16     Zaehler(Zaehler const & orig);           // Der Kopier-Konstruktor
17     ~Zaehler();                              // Der Destruktor
18
19     bool
20     operator==(Zaehler const & rechts) const; // Vergleichs-Operator
21
22     Zaehler const &
23     operator= (Zaehler const & rechts);      // Zuweisungs-Operator
24
25     friend ostream &
26     operator<<(ostream & os, Zaehler const & rechts); // Ausgabe-Operator
27
28     int  getWert  () const; // Liefert den Wert der   wert-Komponente
29     int  getAnzahl() const; // Liefert den Wert der anzahl-Komponente
30     void setWert  (int z);  // Setzt   den Wert der   wert-Komponente
31  }; // class Zaehler
```

Regel 1: Klassen, die innerhalb einer Klasse K vereinbart werden, gelten immer als Klassenelemente (und nicht als Objektelemente).

Regel 2: Operatoren sind spezielle Methoden (nämlich Methoden "mit merkwürdige Namen" wie == oder = oder << oder ...).

Regel 3: Als `friend` gekennzeichnete Methoden sind keine Elemente der Klasse (weder Klassenelemente noch Objektelemente).

Beantworten Sie die folgenden Fragen:

1. Wieviele Konstruktoren enthält diese Klasse `Zaehler`?
2. Wieviele Klassenelemente enthält diese Klasse (einschliesslich Konstruktoren/Destruktor)?
3. Geben Sie die Anzahl und die Namen aller Objektattribute an.
4. Geben Sie die Anzahl und die Namen aller Objektmethoden an.
5. Geben Sie die Anzahl und die Namen aller privaten Elemente der Klasse an.
6. Geben Sie die Anzahl und die Namen aller öffentlichen Elemente der Klasse an.
7. Wieviele Elemente enthält die Klasse `Zaehler` insgesamt?

Lösung Klassen 1

1. Wieviele Konstruktoren enthält diese Klasse `Zaehler`?

3 Konstruktoren

2. Wieviele Klassenelemente enthält diese Klasse (einschliesslich Konstruktoren/Destruktor)?

5 Klassenelemente (3 Konstruktoren, 1 Destruktor und eine innere Klasse namens `Zahl`)

3. Geben Sie die Anzahl und die Namen aller Objektattribute an.

1 Objektattribut (namens `av`)

4. Geben Sie die Anzahl und die Namen aller Objektmethoden an.

5 Objektmethoden (namens `operator==`, `operator=`, `getWert`, `setAnzahl` und `setWert`)

5. Geben Sie die Anzahl und die Namen aller privaten Elemente der Klasse an.

2 Elemente (1 Typ namens `Zahl` und 1 Objektattribut namens `av`).

6. Geben Sie die Anzahl und die Namen aller öffentlichen Elemente der Klasse an.

9 Elemente (3 Konstruktoren namens `Zaehler`, 1 Destruktor namens `~Zaehler`, 5 Objektmethoden namens `operator==`, `operator=`, `getWert`, `getAnzahl` und `setWert`).

7. Wieviele Elemente enthält die Klasse `Zaehler` insgesamt?

11 Elemente.

Übung Klassen 2

Schreiben Sie eine Klasse namens `LongAlsReihung`, die folgende Elemente enthält:

1. Ein privates (`private`) Objektattribut namens `wert` vom Typ `long`.

2. Einen öffentlichen (`public`) Konstruktor mit einem `long`-Parameter, der das Attribut `wert` mit seinem Parameter initialisiert.

3. Eine öffentliche Methode, die der folgenden Deklaration entspricht:

```
1   public char operator[](unsigned n);
2   // Liefert die n-te Dezimalziffer des Attributs wert als Zeichen
3   //(liefert also z.B. '7', nicht 7!).
4   // Ist n gleich 0 so wird die     Einerziffer von wert geliefert.
5   // Ist n gleich 1 so wird die    Zehnerziffer von wert geliefert.
6   // Ist n gleich 2 so wird die Hunderterziffer von wert geliefert.
7   // etc.
```

4. Eine öffentliche Methode, die der folgenden Deklaration entspricht:

```
8   public char operator()(unsigned n, unsigned b);
9   // Interpretiert das Attribut wert als b-er-Zahl, d.h. als Zahl
10  // im Zahlensystem mit der Basis b. Liefert die n-te Ziffer
11  // dieser b-er-Zahl.
12  // Falls b groesser als 16 ist, wird 16 als Basis genommen.
13  // Falls b kleiner  als  2 ist, wird  2 als Basis genommen.
```

Anmerkung: Weil in der Klasse `LongAlsReihung` eine Objektmethode namens `Operator()` definiert wird, bezeichnet man Objekte dieser Klasse auch als Funktionsobjekte. Ist `lar` ein `LongAlsReihung`-Objekt, so bezeichnet z.B. der Ausdruck `lar(2, 10)` die Hunderterziffer seines `wert`-Attributs. Dieser Ausdruck sieht genau so aus wie ein Aufruf einer Funktion namens `lar`.

Lösung Klassen 2

```
1   class LongAlsReihung {
2   private:
3      long wert;
4   public:
5      LongAlsReihung(long wert) : wert(wert) {}
6
7      char operator[](unsigned index) {
8         long w = wert;
9         while (index > 0) {
10           w = w/10;
11           index--;
12        }
13        return w%10 + '0';
14     } // operator[]
15
16     char operator()(unsigned index, unsigned basis) {
17        static char hex[] = {'0', '1','2', '3', '4', '5', '6', '7',
18                             '8', '9','A', 'B', 'C', 'D', 'E', 'F'};
19        basis = max<unsigned>( 2, basis);
20        basis = min<unsigned>(16, basis);
21
22        long w = wert;
23        while (index > 0) {
24           w = w/basis;
25           index--;
26        }
27        return hex[w%basis];
28     } // operator[]
29  }; // class LongAlsReihung
```

Übung Klassen 3

Vereinbaren Sie eine Klasse namens `KlasseA` und vereinbaren Sie darin die folgenden Elemente:

1. Ein privates Objektatrribut namens `zahl` vom Typ `int`.
2. Ein privates Klassenattribut namens `anz` vom Typ `int`.
3. Einen öffentlichen Standardkonstruktor, der das Attribut `zahl` mit `17` initialisiert und `anz` um `1` erhöht.
4. Einen öffentlichen allgemeinen Konstruktor mit einem `int`-Parameter, der das Attribut `zahl` mit seinem Parameter initialisiert und `anz` um `1` erhöht.
5. Einen öffentlichen allgemeinen Konstruktor mit einem `string`-Parameter `s`, der `s` mit `atoi` in einen `int`-Wert umwandelt, mit dem Ergebnis das Attribut `zahl` initialisiert und ausserdem `anz` um `1` erhöht.
6. Einen öffentlichen Kopierkonstruktor, der das neue Objekt mit dem doppelten Wert des `zahl`-Attributs des alten Objekts initialisert und `anz` um `1` erhöht.
7. Einen öffentlichen Destruktor, der `anz` um `1` vermindert.
8. Eine öffentliche Objektmethode namens `getZahl`, die `0` Parameter hat und den Wert des Attributs `zahl` liefert.
9. Eine öffentliche Klassenmethode namens `getAnz`, die `0` Parameter hat und den Wert des Attributs `anz` liefert.

Lösung Klassen 3

In dieser Lösung geben alle Konstruktoren Meldungen aus um damit sichtbar zu machen, dass sie aufgerufen wurden.

```
1   // Datei KlasseA.cpp
2   /* ---------------------------------------------------------------------
3   Eine Klasse, die in einer Uebung anhand einer verbalen Beschreibung ihrer
4   Elemente vereinbart werden soll.
5   --------------------------------------------------------------------- */
6   #include <iostream>   // cout, cin, <<, >>, endl
7   #include <cstdlib>    // atof, atoi, atol, rand, malloc, calloc, free, exit,
8                         // abs, div, getenv, system, bsearch, qsort, ...
9   #include <string>     // size, length, capacity, [], at, clear,
10  using namespace std;
11
12  class KlasseA {
13  private:
14     int    zahl;
15     static
16     int    anz;
17  public:
18     KlasseA()         : zahl(17) {
19        cout << "    Standardkonstruktor,   zahl: " << zahl << endl;
20        anz++;
21     } // Standardkonstruktor
22
23     KlasseA(int zahl) : zahl(zahl) {
24        cout << "Allgemeiner Konstruktor 1, zahl: " << zahl << endl;
25        anz++;
26     } // Allgemeiner Konstruktor 1
27
28     KlasseA(string s) : zahl(atoi(s.c_str())) {
29        cout << "Allgemeiner Konstruktor 2, zahl: " << zahl << endl;
30        anz++;
31     } // Allgemeiner Konstruktor 2
32
33     KlasseA(KlasseA const & orig) : zahl(2*orig.zahl) {
34        cout << "      Kopierkonstruktor,   zahl: " << zahl << endl;
35        anz++;
36     } // Kopierkonstruktor
37
38     ~KlasseA() {anz--;}
39
40     int    getZahl() {return zahl;}
41     static
42     int    getAnz () {return anz;}
43  }; // class KlasseA
44  // ---------------------------------------------------------------------
45  int KlasseA::anz = 0;
46  // ---------------------------------------------------------------------
47
```

```
48  int main() {
49     cout << "KlasseA: Jetzt geht es los!"                << endl;
50     cout << "-----------------------------------"        << endl;
51     KlasseA a1;
52     KlasseA a2(23);
53     KlasseA a3("10");
54     KlasseA a4(a3);
55     cout << "-----------------------------------"        << endl;
56     cout << "a1.getZahl()     : " << a1.getZahl()        << endl;
57     cout << "a2.getZahl()     : " << a2.getZahl()        << endl;
58     cout << "a3.getZahl()     : " << a3.getZahl()        << endl;
59     cout << "a4.getZahl()     : " << a4.getZahl()        << endl;
60     cout << "KlasseA::getAnz(): " << KlasseA::getAnz()    << endl;
61     cout << "-----------------------------------"        << endl;
62     {
63        KlasseA a1(15);
64        KlasseA a2(30);
65        cout << "KlasseA::getAnz(): " << KlasseA::getAnz() << endl;
66     } // Ende eines Blocks
67     cout << "-----------------------------------"        << endl;
68     cout << "KlasseA::getAnz(): " << KlasseA::getAnz()    << endl;
69     cout << "-----------------------------------"        << endl;
70
71     cout << "KlasseA: Das war's erstmal!" << endl;
72  } // main
73  /* ------------------------------------------------------------------------
74  Ausgabe des Programms KlasseA:
75
76  KlasseA: Jetzt geht es los!
77  -----------------------------------
78      Standardkonstruktor,   zahl: 17
79  Allgemeiner Konstruktor 1, zahl: 23
80  Allgemeiner Konstruktor 2, zahl: 10
81        Kopierkonstruktor,   zahl: 20
82  -----------------------------------
83  a1.getZahl()     : 17
84  a2.getZahl()     : 23
85  a3.getZahl()     : 10
86  a4.getZahl()     : 20
87  KlasseA::getAnz(): 4
88  -----------------------------------
89  Allgemeiner Konstruktor 1, zahl: 15
90  Allgemeiner Konstruktor 1, zahl: 30
91  KlasseA::getAnz(): 6
92  -----------------------------------
93  KlasseA::getAnz(): 4
94  -----------------------------------
95  KlasseA: Das war's erstmal!
96  ------------------------------------------------------------------------ */
```

Übung Schleifen programmieren

1. Programmieren Sie eine Schleife, die die Ganzzahlen von 1 bis 10 (alle auf einer Zeile, durch je ein Blank voneinander getrennt) zur Standardausgabe (d.h. zum Bildschirm) ausgibt, etwa so:

```
1 2 3 4 5 6 7 8 9 10
```

2. Programmieren Sie eine Schleife, die alle durch 3 teilbaren Ganzzahlen zwischen 10 und 40 zur Standardausgabe ausgibt, etwa so:

```
12 15 18 21 24 27 30 33 36 39
```

3. Programmieren Sie für jede der folgenden Zahlenfolgen eine Schleife, die die Zahlenfolge zur Standardausgabe ausgibt:

3.1. **-5 -2 1 4 7 10 13 16 19**
3.2. **1 2 4 8 16 32 64 128 256 512 1024 2048 4096**
3.3. **3 4 6 10 18 34 66 130 258 1026 2050 4098**
3.4. **1 2 4 7 11 16 22 29 37 46 56 67 79 92**

Für die folgenden Aufgaben nehmen Sie bitte an, dass eine Ganzzahlvariable namens norbert vereinbart und "mit Daten gefüllt" wurde, etwa so:

```
1   int norbert;
2   cin >> norbert;
```

4. Befehlen Sie dem Ausführer, den grössten Teiler von norbert (der kleiner als norbert ist) auszugeben. Falls norbert eine Primzahl ist, soll als grösster Teiler 1 ausgegeben werden

5. Befehlen Sie dem Ausführer, den kleinsten Teiler von norbert (der grösser als 1 ist) auszugeben. Falls norbert eine Primzahl ist, soll als kleinster Teiler der Wert von norbert selbst ausgegeben werden.

6. Befehlen Sie dem Ausführer, die Meldung "norbert ist prim" bzw. "norbert ist nicht prim" auszugeben, je nachdem, ob die Variable norbert eine Primzahl enthält oder nicht.

Für die folgenden Aufgaben nehmen Sie bitte an, dass eine Stringvariable namens sarah vereinbart und "mit Daten gefüllt" wurde, etwa so:

```
3   string sara;
4   getline(cin, sara);
```

7. Befehlen Sie dem Ausführer zu zählen, wie oft das Zeichen 'x' in sara vorkommt. Wenn er damit fertig ist, soll er die Anzahl der 'x' ausgeben.

8. Befehlen Sie dem Ausführer die Dezimalziffern ('0', '1', ... '9') in sarah zu zählen und diese Anzahl auszugeben.

9. Befehlen Sie dem Ausführer die Buchstaben in sarah zu zählen und diese Anzahl auszugeben. Als Buchstaben sollen alle Zeichen von 'a' bis 'z' und von 'A' bis 'Z' gelten.

10. Befehlen Sie dem Ausführer, die Anzahl der führenden Nullen in sarah zu zählen und diese Anzahl auszugeben.

11. Befehlen Sie dem Ausführer zu zählen, wie oft in sarah ein Zeichen 'a' unmittelbar vor einem Zeichen 'b' steht. Auch diese Anzahl soll der Ausführer ausgeben.

Lösung Schleifen programmieren

```
5       // 1. Auszugeben ist die Folge 1 2 3 4 5 6 7 8 9 10
6       for (int i = 1; i <= 10; i++) cout << i << " ";
7
8       // 2. Auszugeben ist die Folge 12 15 18 21 24 27 30 33 36 39
9       for (int j = 12; j <= 39; j += 3) cout << j << " ";
10
11      // 3.1. Auszugeben ist die Folge -5 -2 1 4 7 10 13 16 19
12      for (int k = -5; k <= 19; k += 3) cout << k << " ";
13
14      // 3.2. Auszugeben ist die Folge 1 2 4 8 16 32 64 128 256 1024 2048 4096
15      for (int i = 1; i <= 4096; i *= 2) cout << i << " ";
16
17      // 3.3. Auszugeben ist die Folge 3 4 6 10 18 34 66 130 258 1026 2050 4098
18      for (int j = 1; j <= 4096; j *= 2) cout << (j+2) << " ";
19
20      // 3.4. Auszugeben ist die Folge 1 2 4 7 11 16 22 29 37 46 56 67 79 92
21      int zahl = 1;
22      for (int k = 1; k <= 14; k++) {
23         cout << zahl << " ";
24         zahl += k;
25      }
26
27      // Alternative Lösung (mit Dank an Herrn Gauss und Frau Mehrnaz Goldeh):
28      for (int n=0; n < 14; n++) {
29         cout << (n*(n+1)/2 + 1) << " ";
30      }
31
32      // 4. Der grösste Teiler von norbert:
33      int teiler1 = norbert - 1;
34      while (norbert % teiler1 != 0) teiler1--;
35      cout << "Der groesste Teiler von norbert ist " << teiler1 << endl;
36
37      // 5. Der kleinste Teiler von norbert:
38      int teiler2 = 2;
39      while (norbert % teiler2 != 0) teiler2++;
40      cout << "Der kleinste Teiler von norbert ist " << teiler2 << endl;
41
42      // 6. Enthält norbert eine Primzahl oder nicht?
43      string meldung("norbert ist prim");
44      for (int teiler = 2; teiler < norbert; teiler++) {
45         if (norbert % teiler == 0) {
46            meldung = "norbert ist nicht prim";
47         }
48      }
49
50      // 7. Wie oft kommt 'x' in sarah vor?
51      int anzahl_x = 0;
52      for (int i=0; i < sarah.length(); i++) {
53         if (sarah.at(i) == 'x') anzahl_x++;
54      }
55      cout << "Anzahl 'x' in sarah:         " << anzahl_x << endl;
56
57      // 8. Wieviele Dezimalziffern kommen in sarah vor?
58      int anzahlZiffern = 0;
59      for (int i=0; i < sarah.length(); i++) {
60         char c = sarah.at(i);
61         if ('0' <= c && c <= '9') anzahlZiffern++;
62      }
63      cout << "Anzahl Ziffern in sarah:     " << anzahlZiffern << endl;
64
65       // 8. Wieviele Dezimalziffern kommen in sarah vor? Bessere Loesung:
```

```
66     anzahlZiffern = 0;
67     for (int i=0; i < sarah.length(); i++) {
68        if (isdigit(sarah.at(i))) anzahlZiffern++;
69     }
70     cout << "Anzahl Ziffern in sarah:     " << anzahlZiffern << endl;
71
72     // 9. Wieviele Buchstaben kommen in sarah vor?
73     int anzahlBuchstaben = 0;
74     for (int i=0; i < sarah.length(); i++) {
75        char c = sarah.at(i);
76        if (('a' <= c && c <= 'z') ||
77            ('A' <= c && c <= 'Z')) anzahlBuchstaben++;
78     }
79     cout << "Anzahl Buchstaben in sarah:  " << anzahlBuchstaben << endl;
80
81      // 9. Wieviele Buchstaben kommen in sarah vor? Bessere Loesung:
82     anzahlBuchstaben = 0;
83     for (int i=0; i < sarah.length(); i++) {
84        if (isalpha(sarah.at(i))) anzahlBuchstaben++;
85     }
86     cout << "9.  : Anzahl Buchstaben in sarah: " << anzahlBuchstaben << endl;
87
88
89     // 10. Wieviele führende Nullen kommen in sarah vor?
90     int anzahlFuehrendeNullen = 0;
91     int i1                    = 0;
92     while (sarah.at(i1++) == '0') anzahlFuehrendeNullen++;
93     cout << "Anzahl fuehrender Nullen:    " << anzahlFuehrendeNullen << endl;
94
95
96     // 11. Wie oft kommt 'a' unmittelbar vor 'b' in sarah?
97     int anzahlAvorB = 0;
98     int i2          = 0;
99     while (i2 < sarah.length() - 1) {
100       if (sarah.at(i2) == 'a' && sarah.at(i2+1) == 'b') {
101          anzahlAvorB++;
102          i2 += 2;
103       } else {
104          i2 += 1;
105       }
106    }
107    cout << Anzahl 'a' vor 'b' in sarah: " << anzahlAvorB << endl;
```

Übung Befehlsarten

Betrachten Sie die folgende C++-Quelldatei namens BefehlsArten01.cpp:

```
1   // Datei BefehlsArten01.cpp
2
3   #include <iostream>
4   using namespace std;
5
6   enum Farbe {rot, gruen, blau};
7
8   class Zahl {
9      private:
10        int wert;
11        int anzahlZugriffe;
12     public:
13        Zahl(int w=0) {
14           wert = w;
15        } // Konstruktor Zahl
16        void setWert(int w=0) {
17           wert = w;
18           anzahlZugriffe++;
19        } // Objekt-Methode setWert
20        int getWert() {
21           return wert;
22           anzahlZugriffe++;
23        } // Objekt-Methode getWert
24        int getAnzahlZugriffe() {
25           return anzahlZugriffe;
26        } // Objekt-Methode getAnzahlZugriffe
27  }; // class Zahl
28
29  int const ANZAHL_FARBEN = 3;
30  Zahl      z1(17);
31
32  int summe(int n1, int n2, int n3=0, int n4=0, int n5=0) {
33     return n1 + n2 + n3 + n4 + n5;
34  }
35
36  void main() {
37     cout << "BefehlsArten01: Jetzt geht es los!"      << endl;
38     int int1=17;
39     int int2, int3=13; // Nur int2 wird initialisiert!
40     cout << "Die Summe von int1 bis int3 ist gleich " <<
41             summe(int1, int2, int3, ANZAHL_FARBEN)    << endl;
42     cout << "z1.getWert(): "           <<
43              z1.getWert()             << ", " <<
44             "z1.getAnzahlZugriffe(): " <<
45              z1.getAnzahlZugriffe()    << endl;
46     cout << "BefehlsArten01: Das war's erstmal!"      << endl;
47  } // main
```

Zur Erinnerung: Es gibt 3 Arten von Befehlen (des Programmierers an den Ausführer): Vereinbarungen, Ausdrücke und Anweisungen. Jede der folgenden Zeilengruppen enthält genau einen Befehl. Geben Sie an, zu welcher Art er gehört:

1. Zeile 6, 2. Zeile 8 bis 27, 3. Zeile 29, 4. Zeile 30, 5. Zeile 32 bis 34, 6. Zeile 36 bis 47?
7. Zeile 10, 8. Zeile 16 bis 19, 9. Zeile 33, 10. Zeile 37, 11. Zeile 38, 12. Zeile 39?

13. Wieviele und was für Befehle stehen direkt in der Datei BefehlsArten01.cpp (ignorieren Sie dabei die Zeilen 1 bis 4)?
14. Geben Sie Ausdrücke an, die in der Datei BefehlsArten01.cpp vorkommen.

Lösung Befehlsarten

1. (Zeile 6) Vereinbarung (eines Aufzählungstyps namens Farbe)
2. (Zeile 8 bis 27) Vereinbarung (eines Klassentyps namens Zahl)
3. (Zeile 29) Vereinbarung (einer int-Konstanten namens ANZAHL_FARBEN)
4. (Zeile 30) Vereinbarung (einer Zahl-Variablen namens z1)
5. (Zeile 32 bis 34) Vereinbarung (einer Funktion namens summe)
6. (Zeile 36 bis 47) Vereinbarung (einer Prozedur namens main)
7. (Zeile 10) Vereinbarung (eines Objekt-Attributs, einer Variablen namens wert)
8. (Zeile 16 bis 19) Vereinbarung (einer Objekt-Variablen, einer Prozedur namens setWert)
9. (Zeile 33) Anweisung (return-Anweisung)
10. (Zeile 37) Anweisung (Ausgabe-Anweisung)
11. (Zeile 38) Vereinbarung (einer int-Variablen namens int1)
12. (Zeile 39) Vereinbarung (zweier int-Variablen namens int2 und int3)

13. In der Datei BefehlsArten01.cpp stehen 6 Vereinbarungen (siehe 1. bis 6.)

14. Ausdrücke:

0, 3, 17, 13	vom Typ int
"BefehlsArten01: Jetzt geht es los!"	vom Typ char *
n1 + n2 + n3 + n4 + n5	vom Typ int
"Die Summe von int1 bis int3 ist gleich "	vom Typ char *
summe(int1, int2, int3, ANZAHL_PARAMETER)	vom Typ int
int1, int2, int3	vom Typ int
ANZAHL_PARAMETER	vom Typ int
"z1.getWert(): "	vom Typ char *
z1.getWert()	vom Typ int
"z1.getAnzahlZugriffe(): "	vom Typ char *
z1.getAnzahlZugriffe(),	vom Typ int

Wenn der Ausführer das Programm BefehlsArten01 ausführt, führt er zuerst die 6 Vereinbarungen aus der Datei BefehlsArten01.cpp aus, d.h. er erzeugt den Aufzählungstyp Farbe, den Klassentyp Zahl, die int-Konstante ANZAHL_FARBEN, die Zahl-Variable z1, die Funktion summe und die Prozedur main. Dann führt er die Prozedur main aus.

Übung C++-Strings

Schreiben Sie Unterprogramme entsprechend den folgenden Spezifikationen:

```
1   bool nurZiffern(string const & s);
2      // Liefert true, wenn s nur Dezimalziffern (Zeichen zwischen '0' und
3      // '9') enthaelt und sonst false.
4   // ---------------------------------------------------------------------
5   int index(string const & text, string const & gesucht);
6      // Falls der String gesucht im String text nicht vorkommt, wird -1 als
7      // Ergebnis geliefert. Sonst wird der kleinste Index geliefert, bei dem
8      // im String text der String gesucht beginnt. Beispiele:
9      // index("abcde", "de") ist gleich  3
10     // index("abcde", "bc") ist gleich  1
11     // index("abcab", "ab") ist gleich  0 (und nicht 3!)
12     // index("abcde", "xy") ist gleich -1
13     // index("abcde", "")   ist gleich  0
14  // ---------------------------------------------------------------------
15  string toString(int n);
16     // Wandelt den int-Wert n in einen String um und liefert diesen als Er-
17     // gebnis. Der Ergebnis-String besteht aus einem Vorzeichen ('+' bzw.
18     // '-') gefolgt von Dezimalziffern ('0' bis. '9'). Beispiele:
19     // toString(715) ist gleich "+715"
20     // toString(-12) ist gleich "-12"
21     // toString(0)   ist gleich "+0"
22  // ---------------------------------------------------------------------
23  string toString(int n, int basis);
24     // Wandelt den int-Wert n in einen String um, genauer: in eine Zahl zur
25     // Basis basis. Der Ergebnis-String beginnt immer mit einem Vorzeichen
26     // ('+' bzw. '-'). Die basis muss zwischen 2 und 16 liegen (sonst wird
27     // die Fehlermeldung "toString-Error" als Ergebnis geliefert). Beispiele:
28     // toString(715, 10) ist gleich "+715"
29     // toString(-12, 10) ist gleich "-12"
30     // toString(  7,  2) ist gleich "+111";
31     // toString(255, 16) ist gleich "+FF";
32     // toString(-8,   8) ist gleich "-10";
33     // toString(123,  1) ist gleich "toString-Error"
34     // toString(123, 17) ist gleich "toString-Error"
35  // ---------------------------------------------------------------------
36  int toInt(string const & s);
37     // Die Dezimalzahl in s wird in einen int-Wert umgewandelt und als Er-
38     // gebnis geliefert. Wenn das erste Zeichen von s ein Minuszeichen ist,
39     // dann ist das Ergebnis negativ. Ansonsten werden alle Zeichen in s,
40     // die keine Dezimalziffern sind, ignoriert. Beispiele:
41     // toInt("+123")   ist gleich  123
42     // toInt("-23")    ist gleich  -23
43     // toInt("12DM30") ist gleich 1230 ("DM" wird ignoriert)
44     // toInt("-ab1cd") ist gleich   -1 ("ab" und "cd" werden ignoriert)
45     // toInt("ab-1")   ist gleich    1 ("ab-" wird ignoriert).
46  // ---------------------------------------------------------------------
47  string a2b(string const & s);
48     // Ersetzt in einer Kopie von s jedes Zeichen 'a' durch 'b' und liefert
49     // die Kopie als Ergebnis.
50  // ---------------------------------------------------------------------
51  string a2bb(string const & s);
52     // Ersetzt in einer Kopie von s jedes Zeichen 'a' durch zwei Zeichen 'b'
53     // und liefert die Kopie als Ergebnis:
54  // ---------------------------------------------------------------------
55  string aa2bbb(string const & s);
56     // Ersetzt in einer Kopie von s von rechts nach links jeden Teilstring
57     // "aa" durch "bbb" und liefert die Kopie als Ergebnis. Beispiele:
58     // aa2bbb("aaxyaaxyaa") ist gleich "bbbxybbbxybbb"
59     // aa2bbb("abc")        ist gleich "abc"
60     // aa2bbb("aaa")        ist gleich "abbb" (und nicht gleich "bbba"!)
61  // ---------------------------------------------------------------------
```

Lösung C++-Strings

```
1   // ---------------------------------------------------------------------
2   bool nurZiffern(string const & s) {
3      // Liefert true, wenn s nur Dezimalziffern (Zeichen zwischen '0' und
4      // '9') enthaelt und sonst false.
5      for (string::size_type i=0; i<s.size(); i++) {
6         char c = s[i];
7         if (c < '0' || '9' < c) return false;
8      }
9      return true;
10  } // nurZiffern
11  // ---------------------------------------------------------------------
12  int index(string const & text, string const & gesucht) {
13     // Falls der String gesucht im String text nicht vorkommt, wird -1 als
14     // Ergebnis geliefert. Sonst wird der kleinste Index geliefert, bei dem
15     // im String text der String gesucht beginnt. Beispiele:
16     // index("abcde", "de") ist gleich  3
17     // index("abcde", "bc") ist gleich  1
18     // index("abcde", "xy") ist gleich -1
19     // index("abcab", "ab") ist gleich  0 (und nicht gleich 3)
20     // index("abcde", "")   ist gleich  0
21
22     // Weil string::size_type ein vorzeichenloser Ganzzahltyp ist, wird
23     // zuerst ein Sonderfall abgefangen und behandelt:
24     if (gesucht.size() > text.size()) return -1;
25
26     // Jetzt klappt die Subtraktion (text.size() - gesucht.size()):
27     for (string::size_type i=0; i<=(text.size() - gesucht.size()); i++) {
28        if (text.substr(i, gesucht.size()) == gesucht) return i;
29     }
30     return -1;
31  } //
32  // ---------------------------------------------------------------------
33  string toString(int n) {
34     // Wandelt den int-Wert n in einen String um und liefert diesen als Er-
35     // gebnis. Der Ergebnis-String besteht aus einem Vorzeichen ('+' bzw.
36     // '-') gefolgt von Dezimalziffern ('0' bis. '9'). Beispiele:
37     // toString(715) ist gleich "+715"
38     // toString(-12) ist gleich "-12"
39     // toString(0)   ist gleich "+0"
40
41     // Das Vor-Zeichen bestimmten und n nicht-negativ machen:
42     string vorzeichen = "+";
43     if (n<0) {
44        vorzeichen = "-";
45        n = -n;
46     }
47
48     // Alle Ziffern der Zahl n (mind. eine '0') nach erg bringen:
49     string erg;
50     string ziffer("x");
51     do {
52        ziffer[0] = n % 10 + '0';
53        n         = n / 10;
54        erg.insert(0, ziffer);
55     } while (n>0);
56
57     // Das Vor-Zeichen einfuegen und das Ergebnis zurueckliefern:
58     erg.insert(0, vorzeichen);
59     return erg;
60  } // toString
61  // ---------------------------------------------------------------------
62  string toString(int n, int basis) {
63     // Wandelt den int-Wert n in einen String um, genauer: in eine Zahl zur
```

```
64     // Basis basis. Der Ergebnis-String beginnt immer mit einem Vorzeichen
65     // ('+' bzw. '-'). Die basis muss zwischen 2 und 16 liegen (sonst wird
66     // die Fehlermeldung "toString-Error" als Ergebnis geliefert). Beispiele:
67     // toString(715, 10) ist gleich "+715"
68     // toString(-12, 10) ist gleich "-12"
69     // toString(  7,  2) ist gleich "+111";
70     // toString(255, 16) ist gleich "+FF";
71     // toString(-8,   8) ist gleich "-10";
72     // toString(123,  1) ist gleich "toString-Error"
73     // toString(123, 17) ist gleich "toString-Error"
74
75     // Falsche basis-Werte abfangen und behandeln:
76     if (basis < 2 || 16 < basis) return "toString-Error";
77
78     // Tabelle zum Umwandeln von Zahlen zwischen 0 und 15 in Ziffern zwischen
79     // '0' und 'F':
80     char tab[] = "0123456789ABCDEF"; // Alle erlaubten Ziffern
81
82     // Das Vor-Zeichen bestimmen und n nicht-negativ machen:
83     string vorzeichen = "+";
84     if (n<0) {
85        vorzeichen = "-";
86        n = -n;
87     }
88
89     // Alle Ziffern der Zahl n (mind. eine '0') nach erg bringen:
90     string erg;
91     string ziffer("x");
92     do {
93        ziffer[0] = tab[n % basis];
94        n         =     n / basis;
95        erg.insert(0, ziffer);
96     } while (n>0);
97
98     // Das Vor-Zeichen einfuegen und das Ergebnis zurueckliefern:
99     erg.insert(0, vorzeichen);
100    return erg;
101 } // toString
102 // ------------------------------------------------------------------------
103 int toInt(string const & s) {
104    // Die Dezimalzahl in s wird in einen int-Wert umgewandelt und als Er-
105    // gebnis geliefert. Wenn das erste Zeichen von s ein Minuszeichen ist,
106    // dann ist das Ergebnis negativ. Ansonsten werden alle Zeichen in s,
107    // die keine Dezimalziffern sind, ignoriert. Beispiele:
108    // toInt("+123")   ist gleich  123
109    // toInt("-23")    ist gleich  -23
110    // toInt("12DM30") ist gleich 1230 ("DM" wird ignoriert)
111    // toInt("-ab1cd") ist gleich   -1 ("ab" und "cd" werden ignoriert)
112    // toInt("ab-1")   ist gleich    1 ("ab-" wird ignoriert).
113
114    int  erg = 0;
115    char c;
116
117    // Das Vorzeichen (als int-Wert +1 bzw. -1) bestimmen:
118    int vorzeichen = +1;
119    if (s[0] == '-') {
120       vorzeichen = -1;
121    }
122
123    // Alle Ziffern im s eine Zahl erg umwandeln:
124    for (string::size_type i=0; i<s.size(); i++) {
125       c = s[i];
126       if ('0' <= c && c <= '9') {
127          erg = erg*10 + (c - '0');
128       }
```

```
129    }
130
131    // erg mit dem richtigen Vorzeichen als Ergebnis liefern:
132    return vorzeichen * erg;
133 } // toInt
134 // ---------------------------------------------------------------------
135 string a2b(string const & s) {
136    // Ersetzt in einer Kopie von s jedes Zeichen 'a' durch 'b' und liefert
137    // die Kopie als Ergebnis.
138
139    string kopie(s);
140    for (string::size_type  i=0; i<kopie.size(); i++) {
141       if (kopie[i] == 'a') kopie[i] = 'b';
142    }
143    return kopie;
144 } // a2b
145 // ---------------------------------------------------------------------
146 string a2bb(string const & s) {
147    // Ersetzt in einer Kopie von s jedes Zeichen 'a' durch zwei Zeichen 'b'
148    // und liefert die Kopie als Ergebnis:
149    string kopie(s);
150    for (int i=kopie.size()-1; i>=0; i--) {
151       if (kopie[i] == 'a') kopie.replace(i, 1, "bb");
152    }
153    return kopie;
154 } // a2bb
155 // ---------------------------------------------------------------------
156 string aa2bbb(string const & s) {
157    // Ersetzt in einer Kopie von s von rechts nach links jeden Teilstring
158    // "aa" durch "bbb" und liefert die Kopie als Ergebnis. Beispiele:
159    // aa2bbb("aaxyaaxyaa") ist gleich "bbbxybbbxybbb"
160    // aa2bbb("abc")        ist gleich "abc"
161    // aa2bbb("aaa")        ist gleich "abbb" (und nicht gleich "bbba"!)
162
163    string kopie     (s);
164    string zuErsetzen("aa");
165    string erSatz    ("bbb");
166    int i = kopie.size()-(zuErsetzen.size());
167    while (i>=0) {
168       if (kopie.substr(i, zuErsetzen.size()) == zuErsetzen) {
169          kopie.replace(i, zuErsetzen.size(), erSatz);
170          i -= zuErsetzen.size();
171       } else {
172          i--;
173       }
174    }
175    return kopie;
176 } // aa2bbb
177 // ---------------------------------------------------------------------
```

Übung Vectoren

Schreiben Sie Unterprogramme entsprechend den folgenden Spezifikationen:

```
1   // ------------------------------------------------------------------------
2   int max(vector<int> const & v);
3      // Liefert die groesste Zahl aus v (und INT_MIN falls v leer ist).
4   // ------------------------------------------------------------------------
5   int sum(vector<int> const & v);
6      // Liefert die Summe aller Zahlen in v.
7   // ------------------------------------------------------------------------
8   bool enthaeltDoppelGaenger(vector<int> const & v);
9      // Liefert true fals mind. eine Zahl in v mehr als einmal vorkommt,
10     // und sonst false.
11  // ------------------------------------------------------------------------
12  void drehRum(vector<int> & v);
13     // Dreht die Reihenfolge der Elemente in v herum (aus (3, 15, 7) wird
14     // (7, 15, 3) etc.).
15  // ------------------------------------------------------------------------
16  vector<int> verEinigung(vector<int> const & v1, vector<int> const & v2);
17     // Verlaesst sich darauf, dass v1 und v2 *Mengen* von int-Werten reprae-
18     // sentieren ("keine Doppelgaenger"). Liefert die *Vereinigung* von v1
19     // und v2.
20  // ------------------------------------------------------------------------
21  bool sindDisjunkt(vector<int> const & v1, vector<int> const & v2);
22     // Verlaesst sich darauf, dass v1 und v2 *Mengen* von int-Werten reprae-
23     // sentieren ("keine Doppelgaenger"). Liefert true, falls die Mengen v1
24     // und v2 disjunkt sind ("keine gemeinsamen Elemente haben), sonst false.
25  // ------------------------------------------------------------------------
26  bool enthaelt(vector<int> const & v1, vector<int> const & v2);
27     // Verlaesst sich darauf, dass v1 und v2 *Mengen* von int-Werten reprae-
28     // sentieren ("keine Doppelgaenger"). Liefert true, falls die Menge v1
29     // die Menge v2 enthaelt, sonst false.
30  // ------------------------------------------------------------------------
31  void sort(vector<int> & v);
32     // Sortiert die Elemente von v in aufsteigender Reihenfolge.
33  // ------------------------------------------------------------------------
```

Lösung Vectoren

Zu einigen der folgenden Lösungen gab es keine ensprechende Aufgabe in der Übung Vektoren:

```
34  // ------------------------------------------------------------------------
35  int max(vector<int> const & v) {
36     // Liefert die groesste Zahl aus v (und INT_MIN falls v leer ist).
37
38     int erg = INT_MIN;
39     for (int i=0; i<v.size(); i++) {
40        if (erg < v[i]) erg = v[i];
41     }
42     return erg;
43  } // max
44  // ------------------------------------------------------------------------
45  int sum(vector<int> const & v) {
46     // Liefert die Summe aller Zahlen in v.
47
48     int erg = 0;
49     for (int i=0; i<v.size(); i++) {
50        erg += v[i];
51     }
52     return erg;
53  } // sum
54  // ------------------------------------------------------------------------
55  bool enthaeltDoppelGaenger(vector<int> const & v) {
56     // Liefert true fals mind. eine Zahl in v mehr als einmal vorkommt,
57     // und sonst false.
58
59     for (int i=0; i<v.size()-1; i++) {
60        for (int j=i+1; j<v.size(); j++) {
61           if (v[i] == v[j]) return true;
62        }
63     }
64     return false;
65  } // enthaeltDoppelGaenger
66  // ------------------------------------------------------------------------
67  bool istPalindrom(vector<int> const & v) {
68     // Liefert true, wenn v ein Palindrom ist (d.h. wenn v "von vorn nach
69     // hinten gelesen die gleichen Elemente enthaelt wie von hinten nach
70     // vorn gelesen") und sonst false.
71
72     for (int i=0; i<v.size()/2; i++) {
73        if (v[i] != v[v.size()-(i+1)]) return false;
74     }
75     return true;
76  } // istPalindrom
77  // ------------------------------------------------------------------------
78  void drehRum(vector<int> & v) {
79     // Dreht die Reihenfolge der Elemente in v herum (aus (3, 15, 7) wird
80     // (7, 15, 3) etc.).
81
82     int tmp;
83     int j;
84     for (int i=0; i<v.size()/2; i++) {
85        j    = v.size()-(i+1);
86        tmp  = v[i];
87        v[i] = v[j];
88        v[j] = tmp;
89     }
90  } // drehRum
91  // ------------------------------------------------------------------------
92  vector<int> durchSchnitt(vector<int> const & v1, vector<int> const & v2) {
93     // Verlaesst sich darauf, dass v1 und v2 *Mengen* von int-Werten reprae-
94     // sentieren ("keine Doppelgaenger"). Liefert den *Durchschnitt* von v1
```

```
95     // und v2.
96
97     vector<int> erg;
98     for (int i1=0; i1<v1.size(); i1++) {
99        for (int i2=0; i2<v2.size(); i2++) {
100          if (v1[i1] == v2[i2]) {
101             erg.push_back(v1[i1]);
102             break;
103          }
104       }
105    }
106    return erg;
107 } // durchSchnitt
108 // ------------------------------------------------------------------------
109 vector<int> verEinigung(vector<int> const & v1, vector<int> const & v2) {
110    // Verlaesst sich darauf, dass v1 und v2 *Mengen* von int-Werten reprae-
111    // sentieren ("keine Doppelgaenger"). Liefert die *Vereinigung* von v1
112    // und v2.
113
114    vector<int> erg(v1);
115    for (int i2=0; i2<v2.size(); i2++) {
116       erg.push_back(v2[i2]);
117       for (int i1=0; i1<v1.size(); i1++) {
118          if (v1[i1] == erg.back()) {
119             erg.pop_back();
120             break;
121          }
122       }
123    }
124    return erg;
125 } // verEinigung;
126 // ------------------------------------------------------------------------
127 vector<int> minus(vector<int> const & v1, vector<int> const & v2) {
128    // Verlaesst sich darauf, dass v1 und v2 *Mengen* von int-Werten reprae-
129    // sentieren ("keine Doppelgaenger"). Liefert die Menge v1 - v2 (alle
130    // Elemente von v1, die nicht in v2 sind).
131
132    vector<int> erg;
133    for (int i1=0; i1<v1.size(); i1++) {
134       erg.push_back(v1[i1]);
135       for (int i2=0; i2<v2.size(); i2++) {
136          if (erg.back() == v2[i2]) {
137             erg.pop_back();
138             break;
139          }
140       }
141    }
142    return erg;
143 } // minus
144 // ------------------------------------------------------------------------
145 bool sindDisjunkt(vector<int> const & v1, vector<int> const & v2) {
146    // Verlaesst sich darauf, dass v1 und v2 *Mengen* von int-Werten reprae-
147    // sentieren ("keine Doppelgaenger"). Liefert true, falls die Mengen v1
148    // und v2 disjunkt sind ("keine gemeinsamen Elemente haben), sonst false.
149
150    for (int i1=0; i1<v1.size(); i1++) {
151       for (int i2=0; i2<v2.size(); i2++) {
152          if (v1[i1] == v2[i2]) return false;
153       }
154    }
155    return true;
156 } // sindDisjunkt
157 // ------------------------------------------------------------------------
158 bool enthaelt(vector<int> const & v1, vector<int> const & v2) {
159    // Verlaesst sich darauf, dass v1 und v2 *Mengen* von int-Werten reprae-
```

```
160    // sentieren ("keine Doppelgaenger"). Liefert true, falls die Menge v1
161    // die Menge v2 enthaelt, sonst false.
162
163    bool istDrin;
164    for (int i2=0; i2<v2.size(); i2++) {
165       istDrin = false;
166       for (int i1=0; i1<v1.size(); i1++) {
167          if (v1[i1] == v2[i2]) {
168             istDrin = true;
169             break;
170          }
171       }
172       if (!istDrin) return false;
173    }
174    return true;
175 } // enthaelt
176 // ------------------------------------------------------------------------
177 void sort(vector<int> & v) {
178    // Sortiert die Elemente von v in aufsteigender Reihenfolge.
179
180    int tmp;
181    for (int bis=v.size()-1; bis>=0; bis--) {
182       for (int j=0; j<bis; j++) {
183          if (v[j] > v[j+1]) { // wenn falsch rum, dann
184             tmp    = v[j];    // vertauschen
185             v[j]   = v[j+1];
186             v[j+1] = tmp;
187          }
188       }
189    }
190 } // sort
191 // ------------------------------------------------------------------------
192 ostream & operator << (ostream & os, vector<int> const & v) {
193    if (v.size() == 0) {
194       os << "Ein vector<int> mit null Komponenten!";
195    } else {
196       for (int i=0; i<v.size(); i++) {
197          os << v[i] << " ";
198       }
199    }
200    return os;
201 } // operator <<
202 // ------------------------------------------------------------------------
```