

Vorname

Nachname

Matrikel-Nr

**Aufgabe 1** (15 Punkte): Schreiben Sie (in C++) ein Unterprogramm namens **rundeAb** entsprechend der folgenden Deklaration:

```

1 vector<unsigned> rundeAb(vector<unsigned> const & vu);
2 // Liefert eine Kopie des Vektors vu, in der jede int-Komponente so
3 // abgerundet wurde, dass ihre letzte Ziffer gleich 0 oder gleich 5 ist.
4 // Genauer: Falls die letzte Dezimalziffer einer Zahl in vu
5 // zwischen 0 und 4 (einschliesslich) liegt, dann wird diese Ziffer
6 // durch 0 ersetzt und falls die letzte Dezimalziffer zwischen 5 und 9
7 // (einschliesslich) liegt, wird sie durch 5 ersetzt.
8
9 // Beispiel: Wenn ein Vektor vu1 die unsigned-Komponenten
10 // 199, 57, 235, 914, 2222 und 120 enthaelt,
11 // dann enthaelt der Vektor rundeAb(vu1) die Komponenten
12 // 195, 55, 235, 910, 2220 und 120.
```

**Aufgabe 2** (15 Punkte): Schreiben Sie (in C++) ein Unterprogramm namens **macheGleichLang** entsprechend der folgenden Deklaration:

```

1 vector<string> macheGleichLang(vector<string> const & vs);
2 // Liefert eine Kopie des Vektors vs, in der alle string-Komponenten
3 // gleich lang sind. "Zu kurze" string-Komponenten werden dazu vorn mit
4 // Blanks verlaengert.
5
6 // Beispiel: Wenn ein Vektor vs die string-Komponenten
7 // "ab", "123456" und "xyz" enthaelt,
8 // dann enthaelt der Vektor macheGleichLang(vs) die string-Komponenten
9 // "  ab", "123456" und "  xyz":
```

**Aufgabe 3** (15 Punkte): Betrachten Sie das folgende Unterprogramm:

```

1 int main() {
2     struct Knubbel {
3         string s;
4         Knubbel * zak1;
5         Knubbel * zak2;
6         Knubbel(string s, Knubbel * zak1, Knubbel * zak2) :
7             s(s), zak1(zak1), zak2(zak2) {}
8     }; // struct Knubbel
9
10    Knubbel * const Z1 = new Knubbel("Alice", NULL, NULL);
11    Z1->zak1 = new Knubbel("Berta", NULL, Z1);
12    Z1->zak2 = new Knubbel("Celia", Z1, Z1->zak1);
13    Z1->zak1->zak1 = Z1->zak2;
14    ...
15 } // main
```

Stellen Sie sich vor, dass der Ausführer dieses Unterprogramm bis zur Zeile **14** ausgeführt hat. Wie sieht die Zeiger-Konstante **Z1** und alles, "was an Z1 dranhängt", in diesem Moment aus? Stellen Sie die Zeiger-Konstante **Z1** und alles, was in ihr dranhängt, als **Bojen** dar.

**Aufgabe 4** (15 Punkte): Schreiben Sie eine **Funktions-Schablone** namens **max**, auf die folgende Beschreibung passt:

Schablonen-Parameter:      Einer (ein Typ T)  
Unterprogramm-Parameter:   Zwei (vom Typ T)  
Rückgabe-Typ:              Ebenfalls der Typ T.

Wenn man eine Instanz der Schablone **max** aufruft, dann soll sie ihre beiden Parameter mit dem Operator **>=** vergleichen und den grösseren der beiden als Ergebnis liefern (falls die beiden Parameter **gleich** gross sind, soll der **erste** als Ergebnis geliefert werden).

Betrachten Sie dann die folgenden 3 Paare von Variablen:

```
1  string s1("Hallo"); // Paar 1
2  string s2("Hello"); // Paar 1
3
4  char   c1('A');      // Paar 2
5  char   c2('a');      // Paar 2
6
7  short  g1(123);      // Paar 3
8  long   g2(234);      // Paar 3
```

Der Wert der jeweils grösseren Variablen soll ausgegeben werden. Welche Variable eines Paares grösser ist, sollen Sie natürlich mit Hilfe von Instanzen Ihrer Schablone **max** berechnen lassen (und nicht etwa "im Kopf" ausrechnen). Geben Sie drei Ausgabe-Befehle der Form

```
9 cout << max ... << endl;
```

an, die den jeweils grösseren Wert eines Paares ausgeben.

**Aufgabe 5** (15 Punkte): Betrachten Sie das folgende C++-Programm namens Prog01:

```
1 // Datei Prog01.cpp
2
3 class Zahl {
4 private:
5     int wert;
6 public:
7     static int const MIN = -500;
8     static int const MAX = +800;
9     static int const ERR = MIN-1;
10
11     Zahl(int wert=ERR) {
12         set(wert);
13     } // Konstruktor Zahl
14     int get() {
15         return wert;
16     } // get
17     void set(int wert=ERR) {
18         if (MIN <= wert && wert <= MAX) {
19             this->wert = wert;
20         } else {
21             this->wert = ERR;
22         }
23     } // set
24     friend Zahl operator +(Zahl const & z1, Zahl const & z2) {
25         if (z1.wert == ERR || z2.wert == ERR) {
26             return Zahl(ERR);
27         } else {
28             return Zahl(z1.wert + z2.wert);
29         }
30     } // operator +
31 }; // Zahl
```

```
32
33 int main() {
34     ...
35 }
36
37 Zahl    z1(123);
38 Zahl *  z2 = new Zahl(234);
39 Zahl *  z3 = &z1;
40 Zahl &  z4 = z1;
41
42 // Ende der Datei Prog01.cpp
```

Stellen Sie sich den Moment vor, in dem der Ausführer damit beginnt, die Befehle in **Zeile 34** auszuführen.

5.1. **Wieviele Module** existieren in diesem Moment?

5.2. Geben Sie die **Namen** dieser Module an. Falls ein Modul **mehrere** Namen hat, genügt es, wenn Sie **einen** dieser Namen angeben.

5.3. Geben Sie für jeden Modul M an, welche **Elemente** sich in M befinden. Geben Sie für jedes Element einen zusammengesetzten Namen an (der aus einem Modulnamen und dem Namen des betreffenden Elements besteht).

**Aufgabe 6** (15 Punkte): Beantworten Sie die folgenden Fragen **kurz**, aber **genau** (möglichst so ähnlich, wie es in den Wiederholungen zu Beginn der Vorlesungen geschah):

6.1. Definition des Begriffs **binärer Baum**?

6.2. Wann ist ein binärer Baum **sortiert**?

6.3. Was ist ein **Modul**?

6.4. Was ist eine **Klasse**?

6.5. Was ist eine **Variable**?

6.6. Was ist ein **Typ**?

6.7. Eigentlich gibt es in allen Programmiersprachen nur **3 Arten von Befehlen**. Wie heißen diese 3 Befehlsarten auf **Deutsch** und auf **Englisch**?

6.8. Woraus besteht ein C++-Programm (oder: Was sind die **größten sinnvollen Teile** eines C++-Programms?)

6.9. Was ist eine **Operation** (im engeren Sinne des Wortes)?

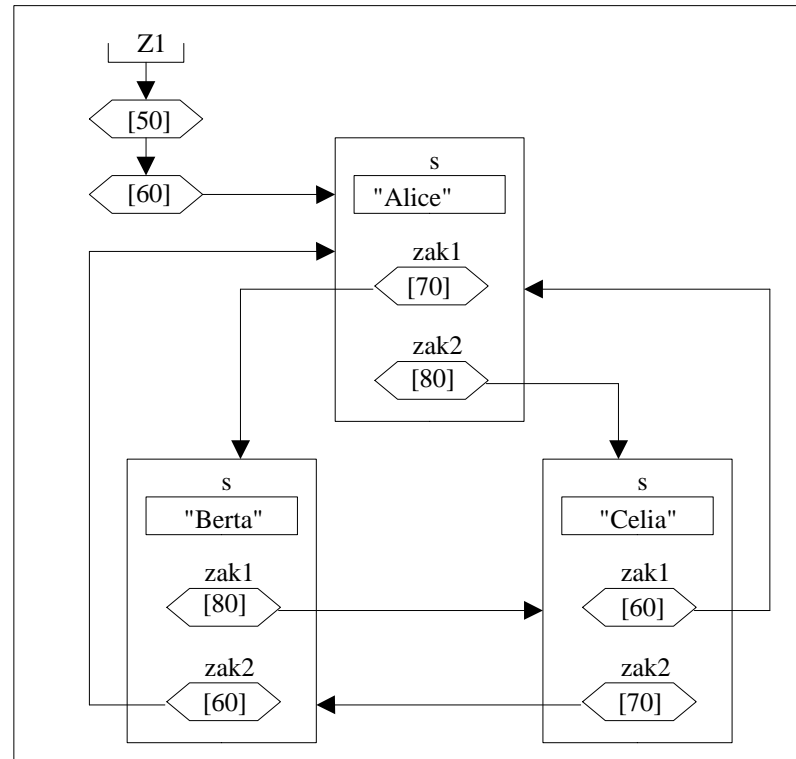
**Lösung 1** (15 Punkte):

```
1 vector<unsigned> rundeAb(vector<unsigned> const & vu) {
2     vector<unsigned> erg(vu);
3     unsigned         ziff;
4
5     for (int i=0; i<erg.size(); i++) {
6         ziff = erg[i] % 10;           // Letzte Ziffer nach ziff
7         erg[i] = erg[i] / 10;       // Letzte Ziffer aus erg[i] entfernen
8         if (ziff <= 4 ) {              // Neue letzte Ziffer,
9             ziff = 0;                  // naemlich 0
10        } else {                      // beziehungsweise
11            ziff = 5;                  // 5 nach ziff bringen.
12        }
13        erg[i] = erg[i] * 10 + ziff; // Neue letzte Ziffer nach erg[i]
14    } // for
15    return erg;
16 } // rundeAb

1 vector<unsigned> rundeAb_elegant(vector<unsigned> const & vu) {
2     // Eine elegantere Version der Funktion rundeAb:
3     vector<unsigned> erg(vu);
4     for (int i=0; i<erg.size(); i++) erg[i] -= erg[i] % 5;
5     return erg;
6 } // rundeAb_elegant
```

**Lösung 2** (15 Punkte):

```
1 vector<string> makeGleichLang(vector<string> const & vs) {
2     int maxLaenge = 0;
3     for (int i=0; i<vs.size(); i++) {
4         if (vs[i].size() > maxLaenge) maxLaenge = vs[i].size();
5     }
6
7     string const BLANKS(maxLaenge, ' ');
8
9     vector<string> erg(vs.size());
10    for (int i=0; i<vs.size(); i++) {
11        string s1 = BLANKS + vs[i];
12        string s2 = s1.substr(s1.size()-maxLaenge, maxLaenge);
13        erg[i] = s2;
14    }
15    return erg;
16 } // makeGleichLang
```

**Lösung 3** (15 Punkte):**Lösung 4** (15 Punkte):

```

1 template <typename T>
2 T max(T p1, T p2) {
3     if (p1 <= p2) {
4         return p1;
5     } else {
6         return p2;
7     }
8 } // max
9
10 ...
11 cout << "max(s1,s2):" << max(s1, s2) << endl;
12 cout << "max(c1,c2):" << max(c1, c2) << endl;
13 cout << "max<long>(g1, g2):" << max<long>(g1, g2) << endl;
14 ...
  
```

**Lösung 5** (15 Punkte):

5.1. Es existieren in diesem Moment **3 Module**, nämlich

5.2. die Klasse **Zahl** und die Objekte **z1** und **\*z2**.

5.3. Elemente der Module:

**Zahl::MIN, Zahl::MAX, Zahl::ERR, Zahl::Zahl**

**z1.wert, z1.get(), z1.set()**

**z2->wert, z2->get(), z2->set()**

**Lösung 6** (15 Punkte):6.1. Definition des Begriffs **binärer Baum**?

Ein binärer Baum ist entweder ein **leerer Baum** oder er besteht aus einem **Knoten K**, an dem zwei Bäume hängen. Diese Bäume bezeichnet man auch als den linken und rechten Unterbaum des Knotens K.

6.2. Wann ist ein binärer Baum **sortiert**?

Wenn für jeden Knoten **K** des Baumes gilt: Im **linken** Unterbaum von K sind alle Schlüssel **kleiner** als der Schlüssel des Knotens K und im **rechten** Unterbaum von K sind alle Schlüssel **größer** als der Schlüssel von K.

6.3. Was ist ein **Modul**?

Ein **Behälter** für Variablen, Konstanten, Unterprogramme, Typen etc., der aus mindestens 2 Teilen besteht, einem **sichtbaren** und einem **unsichtbaren** Teil.

6.4. Was ist eine **Klasse**?

Ein **Modul** und ein **Bauplan** für Module.

6.5. Was ist eine **Variable**?

Ein **Behälter** für **Werte** (dessen Inhalt man jederzeit verändern kann, z.B. mit Hilfe einer Zuweisung).

6.6. Was ist ein **Typ**?

**Antwort1:** Ein **Bauplan** für Variablen.

**Antwort2:** Eine Menge von **Werten** und eine Menge von **Operationen**, die man auf die Werte anwenden kann. (Das Wort **Operation** ist hier in seinem **weiteren** Sinne gemeint).

6.7. Eigentlich gibt es in allen Programmiersprachen nur **3 Arten von Befehlen**. Wie heissen diese 3 Befehlsarten auf **Deutsch** und auf **Englisch**?

**Vereinbarungen** (declarations), **Ausdrücke** (expressions), **Anweisungen** (statements)

6.8. **Woraus besteht ein C++-Programm** (oder: Was sind die **größten sinnvollen Teile** eines C++-Programms?)

**Antwort1:** Ein C++-Programm besteht aus **Dateien** (besser: aus Compilations).

**Antwort 2:** Ein C++-Programm besteht aus **Vereinbarungen**, die in Dateien stehen.

6.9. Was ist eine **Operation** (im **engeren** Sinne des Wortes)?

Ein **Unterprogramm** mit einem **Operator** (z.B. + oder \* oder & etc.) als Namen.