

Vorname

Nachname

Matrikel-Nr

Aufgabe 1 (20 Punkte): Betrachten Sie die folgenden Vereinbarungen:

```
1 int funk01(int n) {
2     int erg = 0;
3     while (n != 1) {
4         erg++;
5         if (n % 2 == 0) {
6             n = n/2;
7         } else {
8             n = 3*n + 1;
9         }
10    }
11    return erg;
12 } // funk01
13
14 int anzahl1 = funk01(12);
```

Mit welchem int-Wert wird die Variable **anzahl1** initialisiert? Geben Sie als Lösung der Aufgabe nur diesen Wert an. Ermitteln Sie den Wert, indem Sie den Funktionsaufruf **funk01(12)** mit Papier und Bleistift (und Radiergummi?) ausführen.

Aufgabe 2 (20 Punkte): Schreiben Sie ein Unterprogramm namens **histo** entsprechend der folgenden Deklaration:

```
1 vector<int> histo(string const & s);
2     // Zaehlt, wie oft die kleinen Buchstaben 'a' bis 'z' in s vorkommen und
3     // liefert das Ergebnis als ein vector<int>-Objekt der Laenge 26. Die
4     // erste Komponente des Ergebnisses enthaelt die Anzahl der 'a' in s,
5     // die zweite Komponente enthaelt die Anzahl der 'b', ..., die letzte
6     // Komponente enthaelt die Anzahl der 'z' in s.
```

Es gibt eine Lösung, die ca. **10 Zeilen** lang ist. Ihre Lösung darf nicht länger als etwa **20 Zeilen** (á 80 Zeichen) lang sein (d.h. Sie dürfen **nicht** mit einer **switch**- oder **if**-Anweisung **26 verschiedene Fälle** unterscheiden).

Aufgabe 3 (20 Punkte): Schreiben Sie ein Unterprogramm namens **merge** entsprechend der folgenden Deklaration:

```
1 vector<string> merge(vector<string> const & v1, vector<string> const & v2);
2     // Verlaesst sich darauf, dass v1 und v2 aufsteigend sortiert sind.
3     // Kopiert die Strings aus v1 und die aus v2 so in einen Vektor erg,
4     // dass dieser ebenfalls aufsteigend sortiert ist. Dann wird der Vektor
5     // erg als Ergebnis geliefert. Beispiel:
6     //
7     // Vector:      Darin enthaltene Strings:
8     // v1:          "charly", "golf"
9     // v2:          "bravo", "delta", "golf", "india"
10    // erg:         "bravo", "charly", "delta", "golf", "golf", "india"
```

Aufgabe 4 (20 Punkte): Betrachten Sie das folgende Programm, welches nur aus einer Datei besteht:

```

1 #include <iostream>
2 using namespace std;
3 // -----
4 class Bett {
5 public:
6     void deckeZu () {zugedeckt = true; }           // Definition
7     void deckeAuf() {zugedeckt = false;}           // Definition
8     static int getNr() {return nr;}                // Definition
9     Bett(bool zugedeckt = true) : zugedeckt(zugedeckt) {}; // Definition
10 private:
11     static int nr;                                // Deklaration (!)
12     bool zugedeckt;                               // Definition
13 }; // class Bett
14
15 int Bett::nr = 100;                               // Definition (!)
16 // -----
17 void main() {
18     cout << "Hallo!" << endl;
19     ...
20 } // main
21 // -----
22 Bett * pb1 = new Bett;
23 Bett * pb2 = pb1;
24 Bett b1;
25 Bett & b2 = b1;

```

Stellen Sie sich vor, dass der Ausführer dieses Programm schon **bis Zeile 18** ausgeführt hat.

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

4.2. Wie **heissen** diese **Module**? Falls ein Modul keinen **einfachen** Namen hat, dann geben Sie bitte einen **zusammengesetzten** Namen an, mit dem man den Modul bezeichnen kann. Falls ein Modul **mehrere** Namen hat dann genügt es, wenn Sie **einen** davon angeben.

4.3. Geben Sie für jeden Modul an, welche **Elemente** sich darin befinden. Es genügt, wenn Sie (hinter dem Namen des Moduls) die **einfachen Namen** der Elemente angeben.

Aufgabe 5 (20 Punkte): Betrachten Sie die folgenden Vereinbarungen:

```

1 struct Knoten {
2     int    daten;
3     Knoten * lub;
4     Knoten * rub;
5     Knoten(int daten=0, Knoten * lub=NULL, Knoten * rub=NULL) :
6         daten(daten), lub(lub), rub(rub) {}
7 }; // struct Knoten
8
9 Knoten * const LETZT = new Knoten;
10 Knoten *      erst  = new Knoten(17, LETZT, new Knoten(22, LETZT, LETZT));
11
12 int *    p1 = new int(123);
13 int * & p2 = p1;

```

Stellen Sie die Konstante **LETZT** und die Variablen **erst**, **p1** und **p2** als **Bojen** dar.

Lösung 1: Die Variable **anzahl1** wird mit dem int-Wert **9** initialisiert.

Lösung 2: Die Funktion **histo**:

```
1 vector<int> histo(string const & s) {
2     // Zaehlt, wie oft die kleinen Buchstaben 'a' bis 'z' in s vorkommen und
3     // liefert das Ergebnis als ein vector<int>-Objekt der Laenge 26. Die
4     // erste Komponente des Ergebnisses enthaelt die Anzahl der 'a' in s,
5     // die zweite Komponente enthaelt die Anzahl der 'b', ..., die letzte
6     // Komponente enthaelt die Anzahl der 'z' in s.
7     vector<int> erg(26, 0);
8     for (unsigned i=0; i<s.size(); i++) {
9         char c = s[i];
10        if ('a' <= c && c <= 'z') {
11            erg[c-'a']++;
12        }
13    }
14    return erg;
15 } // histo
```

Lösung 3: Die Funktion **merge**:

```
1 vector<string> merge(vector<string> const & v1, vector<string> const & v2) {
2     // Verlaesst sich darauf, dass v1 und v2 aufsteigend sortiert sind.
3     // Kopiert die Strings aus v1 und die aus v2 so in einen Vektor erg,
4     // dass dieser ebenfalls aufsteigend sortiert ist. Dann wird der Vektor
5     // erg als Ergebnis geliefert. Beispiel:
6     //
7     // Vector:   Darin enthaltene Strings:
8     // v1:       "charly", "golf"
9     // v2:       "bravo", "delta", "golf", "india"
10    // erg:       "bravo", "charly", "delta", "golf", "golf", "india"
11    vector<string> erg(v1.size() + v2.size());
12    unsigned i1=0; // fuer v1;
13    unsigned i2=0; // fuer v2;
14    unsigned ie=0; // fuer erg;
15    while (true) {
16        // Falls v1 keine unbearbeitete Komponente mehr enthaelt:
17        if (i1 >= v1.size()) {
18            // Die restlichen Komponenten von v2 nach erg kopieren:
19            for (; i2 < v2.size(); ) erg[ie++] = v2[i2++];
20            break; // fertig!
21        }
22        // Falls v2 keine unbearbeitete Komponente mehr enthaelt:
23        if (i2 >= v2.size()) {
24            // Die restlichen Komponenten von v1 nach erg kopieren:
25            for (; i1 < v1.size(); ) erg[ie++] = v1[i1++];
26            break; // fertig!
27        }
28        // Falls v1 und v2 noch unbearbeitete Komponenten enthalten:
29        if (v1[i1] < v2[i2]) {
30            erg[ie++] = v1[i1++]; // Eine Komponente von v1 nach erg
31        } else {
32            erg[ie++] = v2[i2++]; // Eine Komponente von v2 nach erg
33        }
34    } // while (true)
35    return erg;
36 } // merge
```

Lösung 4: Wenn der Ausführer das Programm bis Zeile 18 ausgeführt hat, gilt:

4.1. Es existieren **3 Module**.

4.2. Die Module heissen **Bett**, ***pb1** (oder *pb2) und **b1** (oder b2).

4.3. Die Module enthalten folgende **Elemente**:

Bett: **getNr** (Methode), **Bett** (Konstruktor), **nr** (Attribut)

*pb1: **deckeZu**, **deckeAuf** (Methoden), **zugedeckt** (Attribut)

b1: **deckeZu**, **deckeAuf** (Methoden), **zugedeckt** (Attribut)

Lösung 5: Die Bojen von LETZT, erst, p1 und p2:

