

Vorname

Nachname

Matrikel-Nr

Schreiben Sie jede Lösung einer Aufgabe auf die Vorderseite eines neuen Blatts (**nicht** mehrere Lösungen auf ein Blatt) und lassen Sie die **Rückseiten** aller Blätter **leer**. Kennzeichnen Sie jedes Blatt, das Sie abgeben, mit Ihrem **Nachnamen** (Matrikel-Nr, Schuhgröße etc. sind **nicht** nötig). Diese Klausur besteht aus 6 Aufgaben (siehe Rückseite).

Diese Klausur ist mein **dritter Versuch** (bitte ankreuzen): **Ja** ☐ **Nein** ☐

Aufgabe 1 (15 Punkte): Betrachten Sie das folgende C++-Unterprogramm:

```
1 void aufgabe1() {
2     int r[7] = {3, 0, 4, 6, 1, 5, 2};
3
4     for (int i=0; i<sizeof(r)/sizeof(r[0]); i++) {
5         printf("%d ", r[r[i]]);      // Mit eckigen Klammern
6     }
7     printf("\n");
8
9 }
```

1.1 Was gibt dieses Unterprogramm zur Standardausgabe (zum Bildschirm) aus?

1.2 Der printf-Befehl in Zeile 5 enthält den Ausdruck `r[r[i]]`. Geben Sie einen alternativen Ausdruck an, der exakt das Gleiche leistet, aber *keine eckigen Klammern* enthält.

Aufgabe 2 (15 Punkte): Betrachten Sie die folgenden Variablenvereinbarungen:

```
1 float      v0 = 1.5;
2 float &     v1 = v0;
3 float *     v2 = &v1;
4 float *     v3 = v2;
5 float * *   v4 = &v3;
6 float * * & v5 = v4;
```

1. Stellen Sie die Variablen v0 bis v5 als Bojen dar.

2. Führen Sie (mit Papier und Bleistift oder im Kopf) die folgenden beiden Befehle aus:

```
7     v1 = v1 + 0.5;
8     **v5 = **v5 + 0.5;
```

Wie sieht v0 nach Ausführung dieser Befehle aus? Stellen Sie als Antwort v0 erneut als Boje dar.

Aufgabe 3 (15 Punkte): Betrachten Sie die folgende Klasse namens Knoten und die nachfolgende Prozedur aufgabe3, in der eine Liste solcher Knoten vereinbart wird:

```
1 struct Knoten {
2     int      slue;
3     float    datn;
4     Knoten * next;
5     Knoten(int s, float d, Knoten * n) : slue(s), datn(d), next(n) {}
6 }; // struct Knoten
7
8 #define NanA 0
9 void aufgabe3() {
10     Knoten * list = NanA;
11
12     list = new Knoten(10, 1.0, list); // slue 10, datn 1.0
13     list = new Knoten(20, 2.0, list); // slue 20, datn 2.0
14     list = new Knoten(30, 3.0, list); // slue 30, datn 3.0
15     ...
16 }
```

Geben Sie Befehle an, die ab Zeile 15 stehen könnten und Folgendes leisten:

1. Ein Knoten der Liste `list` hat den `slue` („Schlüssel“) 30. Dieser `slue` wird um 5 vermindert.
2. Ein Knoten der Liste `list` hat die `datn` („Daten“) 2.0. Diese `datn` werden um 0.5 erhöht.
3. Ein Knoten der Liste `list` hat den `slue` („Schlüssel“) 10. Dieser `slue` wird um 7 erhöht.
4. Der Knoten mit dem `slue` 20 wird aus der Liste `list` entfernt (und mit dem `delete`-Befehl gelöscht).

Anmerkung: Die 1. Teilaufgabe sollen Sie durch *eine* einzige Zuweisung lösen. Ebenso für 2. und 3. Zur Lösung von Teilaufgabe 4. sind *mehrere* Befehle erforderlich.

Aufgabe 4 (15 Punkte): Schreiben Sie (in C++) ein Unterprogramm entsprechend der folgenden Deklaration:

```
1  int anzKleinAB(string * r, int len);
2      // Verlaesst sich darauf, dass r die Anfangsadresse einer Reihung
3      // von string-Variablen und len die Laenge dieser Reihung ist.
4      // Zaehlt, wie oft in den string-Komponenten der Reihung die
5      // Zeichenfolge "ab" vorkommt und liefert diese Anzahl als Ergebnis.
6      // Beispiel: Sei eine Reihung sr wie folgt vereinbart:
7      // string sr[5] = {"abcabcab", "cbababac", "ab", "a", ""};
8      // Dann ist anzKleinAB(sr, 5) gleich 6.
```

Aufgabe 5 (15 Punkte): Definieren Sie eine Klasse namens `Klaus`, die folgende Elemente enthält:

1. Ein `private` Objektattribut namens `anna` vom Typ `float`.
2. Eine öffentliche Objektmethode namens `getAnna`, die den Wert von `anna` liefert.
3. Ein `private` Klassenattribut namens `bert` vom Typ `float` mit dem Anfangswert 3.5.
4. Eine öffentliche Klassenmethode namens `getBert`, die den Wert von `bert` liefert.
5. Einen öffentlichen Standardkonstruktor, der `anna` mit 0.01 initialisiert.
6. Einen öffentlichen Konstruktor mit einem `float`-Parameter, der `anna` mit dem Wert dieses Parameters initialisiert.
7. Einen öffentlichen Kopierkonstruktor, der den Wert von `anna` verdoppelt. Dazu ein Beispiel:

```
1  Klaus k1(1.4); // k1 wird mit dem Konstruktor aus 6. initialisiert
2  Klaus k2(k1);  // k2 wird mit dem Kopierkonstruktor aus 7. initialisiert
```

Nach Ausführung dieser Befehle soll das Attribut `k2.anna` den Wert 2.8 (das Doppelte von 1.4) haben.

8. Einen Destruktor, der nur den folgenden Befehl enthält:

```
3  printf("Klaus-Obj. mit anna: %5.2f wird zerstoert!\n", anna);
```

Wichtige Erläuterung 1: Deuten Sie durch eine Kommentarzeile an, dass Sie die Klasse `Klaus` in einer Kopfdatei namens `Klaus.hpp` definieren und verhindern Sie (wie üblich) ein mehrfaches Inkludieren dieser Kopfdatei.

Wichtige Erläuterung 2: Sie sollen nicht nur die Klasse `Klaus`, sondern auch alle ihre Elemente definieren. Elemente, die Sie nicht innerhalb der Klassendefinition definieren wollen bzw. dürfen, sollen Sie (wie üblich) in einer Implementierungsdatei namens `Klaus.cpp` definieren. Machen Sie auch den Beginn dieser Datei durch eine Kommentarzeile deutlich.

Anmerkung: Es ist erlaubt, die Teilaufgaben 5. und 6. gemeinsam mit *einem* Konstruktor zu lösen.

Aufgabe 6 (15 Punkte): Beantworten Sie die folgenden Fragen möglichst kurz, aber genau:

6.1. Was macht der Ausführer, wenn der Programmierer eine Klasse ohne Konstruktoren definiert?

6.2. Was ist ein Standardkonstruktor?

6.3. Woran erkennt man einen Kopierkonstruktor einer Klasse namens `Klasse07`?

6.4. Welche der folgenden Vereinbarungen sind nur Deklarationen und welche sind Definitionen?

```
1  int otto = 17;
2  extern int const KLAUS;
3  int minus(int n) {return -n;}
4  int plus (int n);
5  typedef int ganz;
```

6.5. Die folgenden Vereinbarungen dürfen Sie beim Lösen der nachfolgenden Teilaufgaben voraussetzen:

```
6  int a      = 2;
7  int b      = 5;
8  int r[3] = {10, 20, 30};
```

Geben Sie einen (möglichst kurzen) Ausdruck an, in dem ein *präfix notierter Operator* vorkommt. Ebenso: Einen Ausdruck mit einem *postfix notierten Operator*, einen Ausdruck mit einem *infix notierten Operator* und einen Ausdruck mit einem *mixfix notierten Operator* (insgesamt also 4 Ausdrücke). Die Ausdrücke dürfen beliebig einfach und kurz (oder lang und kompliziert) sein.

Lösung 1 (15 Punkte) : Das Unterprogramm aufgabe11.1 Ausgabe des Unterprogramms: **6 3 1 2 0 5 4**1.2 Ein Ersatz für den Ausdruck `r[r[i]]` ohne eckige Klammern: **`*(r+(r+i))`****Lösung 2** (15 Punkte) : Die Variablen v0 bis v5 als Bojen dargestellt:

```

1  |v0|---<236>---[ 1.5]<---+
2  |   |           |           |
3  |v1|---+           |           |
4  |   |           |           |
5  |v2|---<228>---[<236>]       |
6  |   |           |           |
7  |v3|---<224>---[<236>]---+
8  |   |           |           |
9  |v4|---<220>---[<224>]       |
10 |   |           |           |
11 |v5|---+           |           |

```

Nach Ausführung der beiden Befehle

```

12      v1 =    v1 + 0.5;
13      **v5 = **v5 + 0.5;

```

sieht die Variable v0 wie folgt aus:

```

14 |v0|---<236>---[ 2.5]

```

Lösung 3 (15 Punkte) : Die Klasse Knoten und das Unterprogramm aufgabe3

- Ein Knoten der Liste `list` hat den `slue` („Schlüssel“) 30. Dieser `slue` wird um 5 vermindert.
- Ein Knoten der Liste `list` hat die `datn` („Daten“) 2.0. Diese `datn` werden um 0.5 erhöht.
- Ein Knoten der Liste `list` hat den `slue` („Schlüssel“) 10. Dieser `slue` wird um 7 erhöht.
- Der Knoten mit dem `slue` 20 wird aus der Liste `list` entfernt.

```

1.  list->slue          = list->slue - 5;
2.  list->next->datn     = list->next->datn + 0.5;
3.  list->next->next->slue = list->next->next->slue + 7;

4.  Knoten * tmp = list->next;
    list->next = list->next->next;
    delete tmp;

```

Lösung 4 (15 Punkte) : Die Funktion `anzKleineBuchstaben`:

```

1  int anzKleinAB(string * r, int len) {
2      // Verlaesst sich darauf, dass r die Anfangsadresse einer Reihung
3      // von string-Variablen und len die Laenge dieser Reihung ist.
4      // Zaehlt, wie oft in den string-Komponenten der Reihung die
5      // Zeichenfolge "ab" vorkommt und liefert diese Anzahl als Ergebnis.
6
7      int erg = 0;
8      for (int i=0; i<len; i++) {
9          string s = r[i];
10         // Achtung: s.length() ist meist vorzeichenlos, dagegen
11         // ist j vorzeichenbehaftet. Problem bei Werten unter 0!
12         for (int j=1; j<s.length(); j++) { // Ohne Subtraktion!
13             if (s.at(j-1) == 'a' && s.at(j) == 'b') {
14                 erg++;
15                 j++;
16             }
17         }
18     }
19     return erg;
20 } // anzKleinAB

```

Lösung 5 (15 Punkte) : Eine Klasse namens Klaus mit bestimmten Elementen vereinbaren

```

1 // Datei Klaus.hpp
2 #ifndef Klaus_hpp
3 #define Klaus_hpp
4
5 class Klaus {
6     float anna;
7 public:
8     float getAnna() {return anna;}
9 private:
10    static float bert;
11 public:
12    static float getBert() {return bert;}
13
14    Klaus()          {anna = 0.01;}    // Standardkonstruktor
15    Klaus(float a) {anna = a; }      // Allgemeiner Konstruktor
16    Klaus(Klaus const & original) { // Kopierkonstruktor
17        anna = 2*original.anna;
18    }
19    ~Klaus() {
20        printf("Klaus-Obj. mit anna: %5.2f wird zerstört!\n", anna);
21    }
22 }; // class Klaus
23
24 #endif // #ifndef Klaus_hpp
25
26 // Datei Klaus.cpp
27 #include Klaus.hpp
28 float Klaus::bert = 3.5;

```

Lösung 6 (15 Punkte):

6.1. Was macht der Ausfühler, wenn der Programmierer eine Klasse ohne Konstruktoren definiert?

Er „schenkt“ der Klasse einen Standardkonstruktor und einen Kopierkonstruktor.

6.2. Was ist ein Standardkonstruktor?

Ein Konstruktor, den man mit 0 aktuellen Parametern („ohne Parameter“) aufrufen kann.

6.3. Woran erkennt man einen Kopierkonstruktor einer Klasse namens Klasse07?

Daran, dass er genau einen Parameter des Typs Klasse07 const & oder aber der Typs Klasse07 & hat.

6.4. Welche der folgenden Vereinbarungen sind nur Deklarationen und welche sind Definitionen?

```

1 int otto = 17;                // Definition
2 extern int const KLAUS;      // Deklaration
3 int minus(int n) {return -n;} // Definition
4 int plus (int n);             // Deklaration
5 typedef int ganz;            // Deklaration

```

6.5. Die folgenden Vereinbarungen dürfen Sie beim Lösen der nachfolgenden Teilaufgaben voraussetzen:

```

6 int a    = 2;
7 int b    = 5;
8 int r[3] = {10, 20, 30};

```

Geben Sie einen (möglichst kurzen) Ausdruck an, in dem ein *präfix notierter Operator* vorkommt. Ebenso: Mit einem *postfix notierten Operator*, mit einem *infix notierten Operator* und mit einem *mixfix notierten Operator* (insgesamt sollen Sie also 4 kurze Ausdrücke angeben).

-a a++ a*b r[a]