Technische Universität Berlin **Neuronale Informationsverarbeitung**

C/C++ Übungsblatt 7 (Block 2)

Prof. Dr. Klaus Obermayer und Mitarbeiter

Klassen und Objekte

Verfügbar ab:	**06.12.2022**
Abgabe bis:	**13.12.2022**

Aufgabe 1: Studenten

5 Punkte

Gegeben ist eine Klasse `Studierender` mit den Attributen `nachname` und `vorname` vom Typ `std::string` und `matrikelnummer` vom Typ `unsigned int`.

Listing 1: studierender.hpp

```
1  /*
2   * Klasse, die einen Studierenden repraesentiert
3   */
4  #ifndef STUDIERENDER_HPP
5  #define STUDIERENDER_HPP
6
7  #include <string>
8
9  class Studierender
10 {
11 public:
12     std::string nachname;
13     std::string vorname;
14     unsigned int matrikelnummer;
15     Studierender(std::string nachname, std::string vorname, unsigned int
       matrikelnummer);
16 };
17
18 #endif
```

a) Implementieren Sie in der Datei `studierender.cpp` den Konstruktor. Dieser soll alle Attribute mit den übergebenen Parametern initialisieren. Verwenden Sie die Initialisierungsliste und lassen Sie den Anweisungsblock des Konstruktors leer.

b) Erstellen Sie eine Datei `studierender_test.cpp` welche über eine `main`-Funktion verfügt. Testen Sie darin die von Ihnen implementierte Klasse Studierender, indem Sie in der `main`-Funktion zwei Objekte vom Typ Studierender instanziieren. Eines der Objekte soll im Stack-, das andere im Heap-Speicher abgelegt werden. Geben Sie alle Attribute der beiden Objekte auf der Konsole aus.

Hinweis 1: Achten Sie darauf, dass allozierter Heap-Speicher vor dem Beenden des Programms wieder freigegeben wird.

Hinweis 2: Berücksichtigen Sie, dass zum Kompilieren bzw. Linken von `studierender_test.cpp` auch die Datei `studierender.cpp` (oder ihr Kompilat) übergeben werden muss.

c) Gegeben ist die Datei `uni.cpp` welche über eine `main`-Funktion verfügt.

Listing 2: uni.cpp

```cpp
/*
 * Studierendenverwaltung
 */
#include <iostream>
#include "studierender.hpp"

int main()
{

    // festlegen, wie viele Studierende verwaltet werden sollen
    int n = 0;
    do {
        std::cout << "Bitte geben Sie die Anzahl der Studierenden ein: ";
        std::cin >> n;

        // Stream-Status zuruecksetzen und bis zu 420 Zeichen leeren,
        // falls der User unpassendes oder zu viel eingegeben hat.
        std::cin.clear();
        std::cin.ignore(420, '\n');
    } while (n <= 0);

    // array deklarieren, dass gross genug ist fuer alle Studierenden
    // Studierender* Pointer auf ein Studierendenobjekt auf dem Heap-Speicher,
    // Studierender** Pointer auf einen Pointer auf ein Studierendenobjekt auf
    // dem Heap-Speicher (Also auf das erste Array-Objekt)
    Studierender** studierende;

    // Ihr Code hier ....
}
```

Implementieren Sie in dieser `main`-Funktion Folgendes:

- Erzeugen Sie ein Array mit `n` Elementen vom Typ `Studierender*` und speichern Sie die Adresse in `studierenden`.
- Erzeugen Sie nun in einer geeigneten Schleife insgesamt `n` Objekte vom Typ `Studierender` auf dem Heap-Speicher und weisen deren Adressen den Arrayeinträgen zu. Erzeugen Sie die `Studierender`-Objekte einzeln und initialisieren Sie die Attribute mit Benutzereingaben. Führen Sie den Benutzer hierbei mit geeigneten Konsolenausgaben durch das Programm.
- Geben Sie anschließend die Daten aller Studierenden mithilfe einer weiteren Schleife auf der Konsole aus.

Hinweis 1: Verwenden Sie für die Ein- und Ausgabe die C++-Bibliothek `iostream`.
Hinweis 2: Achten Sie darauf, dass allozierter Speicher vor dem Beenden des Programms wieder freigegeben wird.
Hinweis 3: Zum Kompilieren bzw. Linken muss auch die Datei `studierender.cpp` oder ihr Kompilat angegeben werden.

Aufgabe 2: Objekterzeugung Mensch **3 Punkte**

Wir empfehlen, die folgenden Teilaufgaben erst auf dem Papier zu lösen und anschließend das Ergebnis am Rechner zu prüfen. Abzugeben sind lediglich die Lösungen zu den Fragen.

Legen Sie für die folgenden Teilaufgaben die Klasse `Mensch` zu Grunde:

```
1  class Mensch
2  {
3  public:
4      double getGewicht();
5      void setGewicht(double gewicht);
6  private:
7      std::string name;
8      char geschlecht;
9      double gewicht;
10     Mensch** eltern;
11 };
12
13 double Mensch::getGewicht()
14 {
15     return gewicht;
16 }
17
18 void Mensch::setGewicht(double gewicht)
19 {
20     this->gewicht = gewicht;
21 }
```

a) Nun führt ein Testprogramm die folgende Anweisung aus:

```
Mensch emily;
```

Welche Werte haben die Attribute primitiven Typs des Objekts `klaus`?

b) Angenommen wir erweitern die Klasse `Mensch` um den Konstruktor:

```
1 Mensch::Mensch(std::string name, char geschlecht)
2   : name(name), geschlecht(geschlecht)
3 {
4 }
```

Anschließend führt das Testprogramm die folgende Anweisung aus:

```
Mensch* mika = new Mensch("Mika", 'd');
```

Welche Werte haben die Attribute `name` und `geschlecht` des Objekts `klaus`?

c) Angenommen wir erweitern die Klasse `Mensch` um den folgenden Konstruktor:

```
1 Mensch::Mensch(std::string name)
2   : name(name)
3 {
4     this->eltern = new Mensch*[2];
5     this->eltern[0] = new Mensch("Vater",'m');
6 }
```

Anschließend führt das Programm die folgende Anweisung aus:

```
Mensch *Jona = new Mensch("Jona");
```

Worauf zeigt `eltern[0]`? Wieviele Objekte der Klasse `Mensch` werden nun insgesamt erzeugt?

d) Wir erweitern die Klasse `Mensch` um die folgenden Methoden:

```
1 bool Mensch::istSchwerer(double x)
2 {
3     return gewicht > x; // Vergleich: unser gewicht > uebergebene Variable ?
4 }
5
6 bool Mensch::istSchwerer(Mensch* anderer)
7 {
8     return gewicht > anderer->getGewicht();
9 }
```

Was gibt dann das folgende Progamm auf der Konsole aus?

```
1 Mensch *eva = new Mensch("Eva", 'w');
2 eva->setGewicht(70.0);
3 double x = 70.0;
4 cout << eva->istSchwerer(x) << endl;
5 cout << x << endl;
6 cout << eva->istSchwerer(eva) << endl;
7 cout << eva->getGewicht() << endl;
```

e) Nach welcher Erweiterung (Aufgabe a, b, c oder d) gibt es keinen Default-Konstruktor mehr?

f) Warum wurde bei Teil d) die Anweisung `eva->setGewicht(70.0);` und nicht `eva->gewicht = 70.0;` verwendet?

Aufgabe 3: Point — 2 Punkte

Betrachten Sie die vorgegeben Dateien `mypoint.hpp`, `mypoint.cpp` und `testpoint.cpp`.

Teilaufgabe a): Implementieren Sie in der Klasse `MyPoint` eine Methode

```
double dist(MyPoint *q),
```

die den Abstand des 3D-Punktes, auf den `q` zeigt, zu dem Punkt zurückgibt, dessen `dist`-Methode aufgerufen wird. Verwenden Sie das Programm `testpoint.cpp` zum Testen Ihrer Klasse.

Hinweis 1: Verwenden Sie die Funktion `double std::sqrt(double x)` der Bibliothek `cmath` für die Berechnungen der Wurzel und die Methode `double std::pow(double a, double e)` für die Berechnung

einer Potenz. Dabei ist `double a` die Basis und `double e` der Exponent.
Hinweis 2: Der euklidische Abstand $dist(p, q)$ zweier Punkte $p = (p_x, p_y, p_z)$, $q = (q_x, q_y, q_z) \in \mathbb{R}^3$ ist

$$dist(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2 + (p_z - q_z)^2}$$

Teilaufgabe b): Warum ist die Methode `void show()` notwendig, um die Attribute der `Point`-Objekte in der `main`-Funktion ausgeben zu können?

Listing 3: mypoint.hpp

```
1  /*
2   * Diese Klasse repraesentiert einen Punkt
3   */
4
5  class MyPoint
6  {
7  private:
8    int x;
9    int y;
10   int z;
11 public:
12   MyPoint(int x, int y, int z);
13
14   // Ihr Code hier
15
16   void show();
17 };
```

Listing 4: mypoint.cpp

```
1  #include <iostream>
2  #include <cmath>
3  #include "mypoint.hpp"
4
5  // Konstruktor
6  MyPoint::MyPoint(int x, int y, int z)
7    : x(x), y(y), z(z)
8  {
9  }
10
11 // Ihr Code hier
12
13 // Methode, die die Attribute des Punktes auf die Konsole schreibt
14 void MyPoint::show()
15 {
16   std::cout << "(x,y,z)  =  (" << x << "," << y << "," << z << ")" << std::endl;
17 }
```

Listing 5: testpoint.cpp

```
1  #include <iostream>
2  #include "mypoint.hpp"
3
4  /*
5   * Programm zum testen von MyPoint.cpp
6   */
7  int main()
```

```
8  {
9    MyPoint* p = new MyPoint(1, -3, 2);
10   MyPoint* q = new MyPoint(4, 8, 2);
11
12   p->show();
13   q->show();
14   std::cout << "Abstand: " << p->dist(q) << std::endl;
15
16   delete p;
17   delete q;
18 }
```