PR3 - Übungsblatt B.2

Prof. Dr. Holger Peine
Hochschule Hannover
Fakultät IV – Abteilung Informatik
Raum 1H.2.60, Tel. 0511-9296-1830
Holger.Peine@hs-hannover.de

Thema

Klassen und Objekte, Operatoren, Vererbung

Termin

Ihre Arbeitsergebnisse zu diesem Übungsblatt stellen Sie bitte bis zum 08.01.2026 vor.

1 Item-Id konstant machen (1 Punkte)

basiert auf Vorlesung bis einschl. Abschnitt 10.e

L.10 Lesen Sie zunächst mindestens den Abschnitt L.10.1.3 im Dokument PR3_10_L.pdf im Vorlesungsordner.

Führen Sie die Einkaufswagen-Aufgaben des vorigen Übungsblatts fort, indem Sie das Attribut `itemId` zu einem konstanten Attribut machen. Machen Sie auch alle `CartItem`–Parameter und Rückgabewerte konstant, so weit das sinnvoll ist.

Je nachdem, wie Sie die vorigen Aufgaben umgesetzt hatten, erhalten Sie nun eine Fehlermeldung, dass es nicht ausreicht, das konstante Attribut `itemId` in Ihrem bereits vorhandenen Konstruktor zu initialisieren. Schreiben Sie deshalb einen Copy-Konstruktor für `CartItem`, der für die Kopie eine neue `itemId` vergibt, und einen `operator=`, der die `itemId` des linken Operanden unverändert lässt.

2 `struct` als C++-Klasse mit Operatoren (2 Punkte)

basiert auf Vorlesung bis einschl. Abschnitt 10.f

L.10 Lesen Sie zunächst die Kapitel L.10.1 und L.10.2 im Dokument PR3_10_L.pdf im Vorlesungsordner.

Das Übungsblatt A.3 befasste sich mit der Implementierung eines structs in C für 20-stellige ganze Zahlen. Schreiben Sie nun eine Klasse `Int20` in C++, die dieselbe Aufgabe erfüllt. Nutzen Sie Vereinfachungen, die Ihnen C++ gegenüber C bietet. Als Anlage zu diesem Übungsblatt steht eine Musterlösung der alten Übungsaufgabe (`int20.c`) zur Verfügung.

Im Einzelnen:

- Implementieren Sie einen Konstruktor statt einer Funktion `create20`.
- Implementieren Sie den Copy-Konstruktor und den `operator=`. Beide erhalten eine Referenz auf ein konstantes `Int20`-Objekt. Der `operator=` liefert eine Referenz auf das soeben initialisierte Objekt selbst. Vermeiden Sie redundanten Code.

- Implementieren Sie zwei Operatoren `operator+=` und `operator+` an Stelle der Funktion `add20`. Beide Operatoren haben konstante Referenzen als Parameter. Der operator+= hat eine normale Referenz als Rückgabe, der operator+ eine Objektkopie als Rückgabe. Vermeiden Sie redundanten Code.
- Implementieren Sie einen Vergleichsoperator `bool operator <` , der zwei `Int20`-Objekte vergleicht.
- Implementieren Sie statt der globalen Funktion `print20` eine Methode `print`.

Realisieren Sie die einzelnen Ziffern innerhalb der Klasse `Int20` als C-Array von Zeichen. Diesen Array sollten Sie "von Hand" verarbeiten (d.h. mit Schleifen), nicht als C-String (denn ihm fehlt ja das \0 am Ende).
Falls Sie ihn unbedingt als C-String verarbeiten wollen, können Sie die String-Funktionen der C-Standardbibliothek verfügbar zu machen, indem Sie in C++ statt `#include <string.h>` schreiben: `#include <cstring>` und für die Funktionen der Ein-/Ausgabe statt … `<stdio.h>` jetzt … `<cstdio>`. (Die folgenden Aufgaben befassen sich dann mit der internen Umstrukturierung der Klasse `Int20` durch neue sprachliche Mittel von C++).

Wenn Sie in C++ eine Zeichenkette in Anführungszeichen an einen Methodenparameter vom Typ „Zeiger auf `char*`“ übergeben, so sollten Sie im Methodenkopf den Parametertyp `const char*` vorsehen (und nicht `char*`). Warum? Weil eine in Anführungszeichen stehende Zeichenkette eine Zeichenkette mit prinzipiell unveränderlichem Inhalt ist. Wenn Sie stattdessen `char*` als Parametertyp deklarieren, erhalten Sie je nach Compiler-Einstellungen eine Warnung wie „`deprecated conversion from string constant to 'char*'`“.

3 Ausgabe-Operator, Default-Parameter (1 Punkt)

basiert auf Vorlesung bis einschl. Abschnitt 10.f

Die vorstehende Aufgabe befasste sich mit der Implementierung einer Klasse `Int20` für 20-stellige ganze Zahlen. Erweitern Sie das Programm nun um einen Operator für die Ausgabe.

Im Einzelnen:

- Implementieren Sie `print` als Methode mit einem Default-Parameter vom Typ `ostream&`. Default-Wert soll die Standardausgabe sein.
- Implementieren Sie einen `operator<<` zur Ausgabe auf einen beliebigen `ostream`.

4 `std::vector` (0 Punkte)

basiert auf Vorlesung bis einschl. Abschnitt 10.h

L.10 Lesen Sie zunächst die Kapitel L.10.1 bis L.10.3 im Dokument PR3_10_L.pdf im Vorlesungsordner.

Ändern Sie die Klasse `Int20` nun erneut, indem Sie das interne Array durch einen `vector` ersetzen. Dabei soll das erste Element des Vektors der Einerstelle der großen ganzen Zahl entsprechen, das zweite Element der Zehnerstelle, usw. Erweitern Sie die Klasse derart, dass Zahlen beliebiger Länge aufgenommen werden können.

Eine Umbenennung der Klasse in `IntBig` o. ä. wäre anzuraten, da eine Beschränkung auf 20 Ziffern nicht mehr vorliegt.

Die Klasse vector haben wir schon auf dem letzten Übungsblatt kennen gelernt.
Beispiel:

```
vector<int> zahlen;
zahlen.push_back(6);
zahlen.push_back(4);
zahlen.push_back(7);
int summe;
for (auto zahl: zahlen) {
  summe += zahl;
}
```

Mehr Informationen zur `vector`-Klasse finden Sie z. B. hier: http://cplusplus.com/reference/stl/vector/. Sehen Sie sich z. B. die folgenden Methoden genauer an, um einen Einstieg in diese Klasse zu bekommen:
`push_back`, `pop_back`, `operator[]`, `at`, `size`.

Wenn Sie Lust haben, können Sie weitere Operatoren ergänzen (z.B. Zugriff mit dem Index-Operator).

5 Einfache String-Klasse implementieren (0 Punkte)

basiert auf Vorlesung bis einschl. Abschnitt 10.h

L.10 Lesen Sie zunächst das Kapitel L.10.2 im Dokument PR3_10_L.pdf im Vorlesungsordner.

Implementieren Sie eine Klasse `MyString` als vereinfachte Version der Klasse `std::string`. `MyString` sollte intern einen C-String, also einen dynamisch allokierten `\0`-terminierten C-Array von `char`'s, zum Speichern der Zeichen benutzen. `MyString` sollte folgende Operationen unterstützen:

- `MyString empty {};` // Leerer String
- `MyString abc {"abc"};` // Mit `const char[]` initialisieren
- `MyString abc2 {abc};` // Mit anderem `MyString` initialisieren
- `abc = abc2;` // Zuweisung
- `cout << abc;` // Ausgabe auf `ostream`'s
- `int len {abc.length()};` // Stringlänge
- `bool equal {abc == abc2};` // Test auf Gleichheit
- `abc[2] = 'd';` // Indexzugriff auf einzelnes Zeichen mit …
  // … Exception `out_of_range(…)`, falls Index außerhalb des Strings
- `MyString def {"def"};`
- `MyString abcdef {abc + def};` // Verkettung von `MyString`'s mittels +
- `abcdef += abc;` // Anhängen mittels `+=`
- `MyString defdefdef {def * 3};` // Mehrfaches Verketten desselben `MyString`

Beim Umgang mit C-Arrays sollten Sie die C-String-Funktionen benutzen (in C++-Programmen mittels `#include <cstring>` verfügbar). Auf die Behandlung von Speicherfehlern dürfen Sie der Einfachheit halber verzichten. Sie können Ihre Klasse mit dem Programm `MyString_test.cc` in den Anlangen zu diesem Übungsblatt testen.

6 Virtuelle Aufrufe in Konstruktoren (0 Punkte)

basiert auf Vorlesung bis einschl. Abschnitt 11.b

In der Vorlesung wurde erläutert, in welcher Reihenfolge Konstruktoren in einer Ableitungshierarchie ausgeführt werden: Zuerst der Konstruktor der Basisklasse, dann der der Subklasse.

Es kommt zu Problemen, wenn im Konstruktor der Basisklasse eine virtuelle Methode aufgerufen wird. Da zum Zeitpunkt der Konstruktion des Basis-Objekts das Sub-Objekt noch gar nicht existiert, verhält sich der Aufruf der virtuellen Methode nicht polymorph. Schreiben Sie ein kleines Programm, das das Problem demonstriert. Wie verhält sich Ihr Programm, wenn die aufgerufene Methode pure virtual deklariert wird (ohne Definition)?

Wie verhält sich ein äquivalentes Programm in Java? Wie verhält sich das Java-Programm, wenn es in der Submethode auf Attribute der Subklasse zugreift?

7 Polymorphie (2 Punkte)

basiert auf Vorlesung bis einschl. Abschnitt 11.b

Schreiben Sie ein Programm zur Simulation des Verhaltens von Hase und Igel bei einem Wettrennen. Beide Tiere starten an der Position 0. Wer als erster die Position 99 erreicht hat, hat gewonnen.

Bildquelle: zwerge.de

Ein Igel (so nehmen wir an) bewegt sich, wenn er einmal eine Richtung eingeschlagen hat, gemütlich Schritt für Schritt immer in dieselbe Richtung. Der Igel wird also nach 99 Schritten das Ziel erreicht haben.

Bildquelle: zwerge.de

Ein Hase (so nehmen wir an) springt wild hin und her. Die Sprungweite soll in Ihrem Programm leicht veränderbar sein. Wenn die Sprungweite z. B. 10 ist, vollführt der Hase zufällige[1] Sprünge nach vorn oder hinten mit Sprungweiten im Bereich -10 bis 10. Dabei kommen auch kleinere Sprünge oder gar Sprünge auf der Stelle in Betracht. Wenn der Hase das Ziel erreicht hat, bleibt er (wie der Igel) stehen.

Implementieren Sie eine abstrakte Superklasse `Teilnehmer`, in der Sie die aktuelle Position des jeweiligen Rennteilnehmers speichern (Position liegt im Bereich 0 bis 99). Die Klasse `Teilnehmer` deklariert eine abstrakte Methode `weiter`, die von den Subklassen `Hase` und `Igel` gemäß der obigen Annahmen zum Bewegungsverhalten implementiert wird. Sie dürfen

[1] Einen im Intervall zwischen `N1` und `N2` gleichverteilten zufälligen `int` namens `randomNumber` bekommt man in C++ wie folgt:

```
#include <random>
using namespace std;
…
random_device r {};
default_random_engine engine {r()};
uniform_int_distribution<int> uniform_dist {N1, N2};
int randomNumber {uniform_dist(engine)};
```

auch weitere Methoden einführen, z.B. zur Ausgabe. Versuchen Sie möglichst nur den Code in den Subklassen `Hase` und `Igel` zu haben, der wirklich spezifisch für einen Hase bzw. einen Igel ist – anderer Code gehört in die Teilnehmer-Klasse oder ins Hauptprogramm.

Im Hauptprogramm erstellen Sie ein Array oder einen `vector` mit zwei Teilnehmern: Hase und Igel. Überlegen Sie, welchen Elementtyp der Array haben muss, damit polymorphe Funktionsaufrufe funktionieren können. Starten Sie im Hauptprogramm dann die Simulation für z. B. 100 Durchläufe (denn dann ist der Igel ja definitiv am Ende angekommen). Geben Sie die aktuelle Rennsituation Zeile für Zeile aus, etwa so:

```
Start:
H-----------------------------------------------------------------------------------------------------
I-----------------------------------------------------------------------------------------------------
Schritt 1:
------H-----------------------------------------------------------------------------------------------
-I-----------------------------------------------------------------------------------------------------
Schritt 2:
-----------H-----------------------------------------------------------------------------------------
--I-----------------------------------------------------------------------------------------------------
Schritt 3:
--------H-----------------------------------------------------------------------------------------
---I-----------------------------------------------------------------------------------------------------
Schritt 4:
----------H-----------------------------------------------------------------------------------------
----I-----------------------------------------------------------------------------------------------------
Schritt 5:
-------------H-----------------------------------------------------------------------------------------
-----I-----------------------------------------------------------------------------------------------------
Schritt 6:
-------------H-----------------------------------------------------------------------------------------
------I-----------------------------------------------------------------------------------------------------
Schritt 7:
-----H-----------------------------------------------------------------------------------------
-------I-----------------------------------------------------------------------------------------------------
```

Wenn Sie das Programm wie beschrieben erstellt haben, ist es nun leicht, den Igel gegen mehrere Hasen antreten zu lassen. Experimentieren Sie mit der Anzahl der Hasen und der Sprungweite der Hasen. Bei welchen Einstellungen geht der Igel nicht mehr als erster ins Ziel?