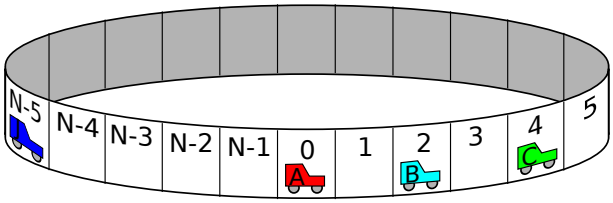
Universität Stuttgart
Institut für
Informationssicherheit

InformatikII - Programmierung in C++
Sommersemester 2023
Maria Unger-Zimmermann

Übungsblatt 10

26.06., 27.06. und 30.06

Zum Üben der Grundlagen objektorientierter Programmierung schreiben wir uns in dieser und der nächsten Übung einen kleinen Verkehrssimulator: Wir haben eine Ringstraße, die aus N Zellen besteht (nummeriert von 0 bis $N-1$, auf Zelle $N-1$ schließt sich wieder Zelle 0 an, so dass man immer weiter im Kreis fahren kann); ein Auto ist charakterisiert durch eine Position (die Nummer der Zelle, in der es ist) und seine Geschwindigkeit, das ist hier die Anzahl der Zellen, die es in einem Zeitschritt weiterfährt.



Im Verlauf der Aufgaben heute und nächste Woche wird ein Programm immer weiter erweitert werden – Sie können das alles in einem Projekt machen oder auch immer ein neues Projekt beginnen und den bisherigen Programmcode kopieren.

Problem 10.1: Klassendeklaration, Attribute

Zunächst schreiben wir zwar formal eine Klassendeklaration, behandeln sie aber so, wie wir das von den Strukturen aus Aufgabe 7.2 gewöhnt sind:

1. Definieren Sie eine globale Konstante `const int N = 50;` (vor den Definitionen aus den weiteren Teilaufgaben, so dass Sie sie dort verwenden können – vgl. Breymann Kap. 1.6.4 und 2.1.2).
2. Definieren Sie nun eine Klasse mit dem Namen `PKW` und drei `public`-Attributen:
   - `name` vom Typ `char` (zum Datentyp `char` s. Breymann, Kap. 1.6.5), unsere Autos bekommen später „Namen“ von A bis J, so dass wir sie in der Ausgabe unterscheiden können,
   - `pos` vom Typ `int` für die Position auf der Straße (wird später eine Zahl zwischen 0 und $N-1$),
   - `v` vom Typ `int` für die Geschwindigkeit.
3. Schreiben Sie nun eine Funktion (noch keine Methode!) `void drucke(PKW a)` mit einem Parameter vom Typ PKW und ohne Rückgabewert, die die Attribute des PKW ausdruckt, z.B. so: `PKW B: Position 2, Geschwindigkeit 7`.
4. Schreiben Sie eine weitere Funktion `void bewege(PKW& a)`, die das Auto a weiterfährt: die Position wird um die Geschwindigkeit erhöht – wenn sie größer oder gleich N wird, wird N abgezogen (wenn man hinten aus unserer Straße hinausfährt, kommt man vorne wieder rein). Wenn z.B. `a.pos==3` wäre und `a.v==4`, sollte hinterher `a.pos==7` sein, bei `a.pos==49`, `a.v==4` hinterher `a.pos==3`.

5. Definieren Sie (in `main` oder auch in einer eigenen Funktion, die Sie aus `main` aufrufen) eine Variable vom Typ `PKW`, geben Sie ihm den Namen 'B', die Position 2 und die Geschwindigigkeit 7. Rufen Sie damit die Funktion `drucke` auf, dann in einer Schleife zehn mal zunächst `bewege` und dann `drucke`, so dass Sie zehn Zeitschritte nachvollzogen haben (die Art der Vorwärtsbewegung ist noch sehr langweilig, nächste Woche wird es interessanter…).

Problem 10.2: Funktionen werden zu Methoden

Schöner als separate Funktionen `drucke` und `bewege` sind Methoden, bei denen wir gleich sehen, dass sie zur Klasse gehören (Breymann Kap. 3.2).

Bauen Sie aus den Funktionen aus Aufgabe 10.1 `drucke()` und `bewege()` Methoden der Klasse `PKW` und probieren Sie sie aus! Der Umbau ist leicht (Sie können die Funktionen kopieren, bevor sie eine Kopie ändern – es kann ohne weiteres Funktionen und Methoden nebeneinander geben.):

- In der Parameterliste entfällt der Parameter vom Typ `PKW` bzw. `PKW&`.
- Dann wird die Signatur (alles vor der öffnenden `{ }` als Funktionsprototyp (mit einem `;` statt des Funktionsrumpfs) in die Klassendefinition kopiert (am besten hinter die bisherigen Attribute).
- In der Funktionsdefinition (d.h. beim Funktionsrumpf) müssen wir nun angeben, zu welcher Klasse sie gehört, indem wir den Klassennamen vor den Funktionsnamen schreiben, durch `::` getrennt. Also etwa `PKW::drucke` statt drucke.
- Im Funktionsrumpf haben wir nun keinen Parameter mit Namen mehr. Statt z.B. `a.pos` können wir aber einfach `pos` schreiben, wenn wir über die Komponenten des aktuellen Objekts reden.
- Der Methodenaufruf sieht nun dem Komponentenzugriff ähnlich: Wenn `a` ein Objekt der Klasse `PKW` ist, ruft `a.drucke()` dessen Methode zum Drucken auf.

Problem 10.3: Konstruktoren

Versehen Sie die Klasse `PKW` nun noch mit einem Konstruktor (Breymann, Kap. 3.3), der folgende Parameter hat, die im Konstruktor einfach den jeweiligen Attributen zugewiesen werden sollen:

- `n` vom Typ `char`,
- `pos0` vom Typ `int`,
- `v0` vom Typ `int`.

(Durch diese Benennung müssen wir uns nicht um die Frage kümmern, was passiert, wenn ein Parameter so heißt wie ein Attribut.)

Da Konstruktoren heißen wie die Klasse und keine Angabe zur Rückgabe haben (nicht einmal void), könnte die Signatur in der Klassendeklaration lauten: `PKW(char n, int pos0, int v0);`

Wir sparen aber etwas Ärger, indem wir an der Stelle Standardwerte hinschreiben (Breymann, Kap. 2.2.4): `PKW(char n = '*', int pos0 = 0, int v0 = 0);`

Bei der Implementierung aber nicht, die liest sich als `PKW::PKW(char n, int pos0, int v0) {…}`

(Der Hintergrund davon ist, dass wir durch diesen Trick gleich einen Konstruktor gebaut haben, der keine Parameter braucht – wenn es so einen Konstruktor nicht gibt, kommt es zu etwas rätselhaften Fehlermeldungen, aber das führt hier zu weit…)

Probieren Sie Ihren Konstruktor aus, indem Sie (z.B. in `main`) ein Auto z.B. mit `PKW a1('B', 2, 7);` anlegen (seit C++11 ist u.A. auch `PKW a1 {'B', 2, 7};` möglich).

Nächste Woche wird das Programm weiter ausgebaut – es bekommt dann ein realistischeres Modell der Autofahrer und wird damit interessante Simulationen ermöglichen.