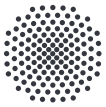
Universität Stuttgart
Institut für
Informationssicherheit

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

Übungsblatt 11

03.07., 04.07. und 07.07

Problem 11.1: Eine weitere Klasse

1. Definieren Sie eine weitere Klasse `strasse`, die am Anfang nur ein `public`-Attribut `autos` hat, das ein Vektor von `PKW` ist.

   Das Einfügen von Autos in den Vektor machen wir immer so, dass die Autos in der Reihenfolge auf der Straße sind, in der sie im Vektor stehen: `auto[1]` direkt (d.h., ohne ein anderes Auto dazwischen) vor `auto[0]` und dieses wiederum direkt vor dem letzten Auto im Vektor (wir sind ja auf einer Ringstraße – im Bild sieht man Auto 'J' gerade wieder in Zelle $N-5$).

2. Fügen Sie (z.B. in `main`) zehn Autos 'A' bis 'J' dem Vektor hinzu (mit `char` kann man rechnen: 'A' $+ 1$ gibt 'B' etc.), das Auto mit Index i im Vektor soll auf der Straße an Position $2i$ starten; heute fahren alle Autos gleich schnell, nämlich mit Geschwindigkeit 7.

3. Schreiben Sie für die Klasse `strasse` eine Methode `size_t strasse::erster_PKW()`. Sie soll für dasjenige Auto, das auf der Straße am weitesten links ist (`pos` ist mninimal), seinen Index im Vektor `autos` zurückgeben. (In unserem Beispiel ist es am Anfang der Index 0 vom Auto 'A', wenn später Autos über Zelle $N-1$ hinausgefahren sind, kommen auch andere Autos infrage).

4. Schreiben Sie nun eine Methode `void strasse::drucke()`, die den Zustand jedes Autos druckt – beim Auto, das am weitesten links ist, beginnend.

   (Hinweis: Wenn eine Variable, z.B. `size_t ind0`, das Ergebnis von `erster_PKW()` ist und eine weitere Variable `size_t i` von 0 bis `autos.size() - 1` läuft, bekommen Sie mit `(ind0 + i) % autos.size()` die Indizes in der richtigen Reihenfolge.)

5. Etwas aufwändiger (diesen Teil können Sie auch erstmal zurückstellen) ist es, die Straße in einer Zeile zu drucken, wobei ein leeres Feld durch einen Punkt und ein Auto durch seinen Namen markiert wird – im Anfangszustand etwa so:

   ```
   A.B.C.D.E.F.G.H.I.J.............................
   ```

   Schreiben Sie dazu wie in Teil 4 eine Schleife über alle Autos und führen Sie einen Zähler mit, der angibt, wie viele Zeichen schon gedruckt wurden. Damit können Sie leicht herausfinden, wie viele Punkte jeweils zu drucken sind.

6. Nun noch schnell eine Methode `void strasse::bewege()`, die für jedes Auto dessen Methode `bewege` aufruft.

7. Dann können wir losfahren: Drucken Sie sie Straße im Ausgangszustand und bewegen Sie für z.B. fünf Zeitschritte die Autos weiter und drucken Sie jeweils den aktuellen Zustand der Straße.

Eine wesentliche realistischere Simulation als in Aufgabe 11.1 bekommt man, indem man die Geschwindigkeit abhängig vom Abstand zum vorausfahrenden Auto wählt. Welches das ist, wird die Klasse `strasse` bestimmen müssen – in unserer Anfangskonfiguration ist es für den PKW mit Index i der mit Index $i+1$, außer für den letzten ($i=9$), dem der erste ($i=0$) vorausfährt. In der kommenden Simulation wird das auch so bleiben, weil sich die Autos nicht überholen werden.

Problem 11.2: Eine realistischere Verkehrssimulation

1. Ergänzen Sie die Klasse PKW um eine Methode `void PKW::setzeV(PKW& vordermann)`, die die Geschwindigkeit v neu berechnet:
   (a) Zunächst soll der Abstand zum Vordermann berechnet werden (aus der Differenz `vordermann.pos - pos`, aber Vorsicht, wenn die negativ wird – z.B. in der Ausgangskonfiguration ist das für das Auto 'J' an Position 18 und seinen Vordermann 'A' an Position 0 erfüllt).
   (b) Dann darf beschleunigt werden: Die Geschwindigkeit wächst um eins.
   (c) Wenn aber die neue Geschwindigkeit größer oder gleich dem berechneten Abstand ist, wird sie auf den Wert eins kleiner als der Abstand gesetzt (wir erreichen also die aktuelle Position des Vordermanns nicht).
2. Nun bekommt die Klasse `strasse` ebenfalls eine Methode `setzeV`, aber ohne Parameter, die für jedes Auto dessen `setzeV` aufruft (wozu sie den Vordermann bestimmen muss).
3. Lassen Sie die Simulation 100 Schritte laufen (Wenn z.B. `s` Ihre Straße ist: Reihum `s.setzeV()`, `s.bewege()` und `s.drucke()` aufrufen). Was beobachten Sie?
4. Schönere Ergebnisse bekommt man, indem man in der Geschwindigkeitsberechnung aus Teil 1 nach Schritt (c) noch einen Mechanismus einbaut, der das Trödeln simuliert: Wenn die Geschwindigkeit wenigstens 1 ist, wird sie mit einer gewissen Wahrscheinlichkeit α (mit $0 \leq \alpha < 1$) um 1 reduziert.
   Dazu können wir Pseudozufallszahlen vom Typ `double` im Bereich $0, \ldots, 1$ erzeugen mittels `rand() / double(RAND_MAX)` („Pseudozufallszahlen“, weil die Folge der Zahlen in Wirklichkeit keineswegs zufällig ist – insbesondere erhalten Sie bei mehrfachem Ausführen Ihres Programms immer dieselbe Folge).
   Wenn wir diesen Wert mit α vergleichen, trifft der Vergleich gerade mit einer Wahrscheinlichkeit α zu.
   Implementieren Sie das Trödeln mit $\alpha = 0.1$!

Anmerkung: In diesem Zustand ist unser Modell schon recht ähnlich zum Nagel-Schreckenberg-Modell, mit dem auch realistische Verkehrssimulationen möglich sind.

Problem 11.3: `private`-Attribute

Bisher waren alle unsere Attribute als `public` deklariert. Dadurch haben wir uns bei unserem kleinen Programm Arbeit gespart, bei einem größeren Programm spart man sich aber viel Ärger (und letzten Endes auch Arbeit), wenn man dafür sorgt, dass auf die Attribute nur in der Weise zugegriffen wird, wie wir das vorsehen.

Dazu werden die Attribute als `private` deklariert (Breymann Kap. 3.2, S. 155-157) und Methoden zur Verfügung gestellt, um Werte der Attribute abzufragen oder ihnen neue Werte zuzuweisen.

Das probieren wir mit der Klasse `PKW` aus:

1. Setzen Sie alle Attribute der Klasse `PKW` auf `private` (die Methoden bleiben `public`).
2. Probieren Sie, das Programm zu übersetzen und sehen Sie nach, wo es nun Fehlermeldungen gibt.
   Vermutlich sind das gar nicht so viele Stellen – und auch nur lesende Zugriffe (die also den aktuellen Wert eines Attributs abfragen), alle Zuweisungen von neuen Werten sollten bereits im Konstruktor oder über die Methoden `setzeV` und `bewege` passieren.
   Daher können Sie das Programm reparieren, indem Sie für jedes Attribut `X`, das einen Fehler verursacht, eine Methode `get_X` ohne Parameter und mit passendem Rückgabetyp schreiben, die einfach den Wert des Attributes zurückliefert.
   An den Stellen, an denen sich der Übersetzer über einen Zugriff auf ein `private`-Attribut beschwert, ersetzen wir den einfach durch einen Aufruf der neuen Methode.

3. Typischerweise würde man eine solche Methode als `inline`-Methode schreiben (Breymann Kap. 3.2.2, hier nehmen wie die Variante „Definition innerhalb der Klasse“) – probieren Sie das mit wenigstens einer der Methoden aus 2 aus!

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