Dirk Praetorius,
Marcus Page,
Thomas Führer

Wintersemester 2012/13
04.12.2012

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 10

Aufgabe 10.1*. Implementieren Sie die Zugriffsfunktionen der Klasse `Bruch`, welche durch

```
class Bruch {
  long zaehler;
  unsigned long nenner;
public:
  Bruch();
  setZahler(long z);
  setNenner(unsigned long n);
  double getWert();
};
```

gegeben ist. Die Methode `getWert` soll dabei den Dezimalwert des Bruchs zurückgeben. Achten Sie bei den `set`-Funktionen auf eventuelle Zugriffskontrollen. Implementieren Sie auch den Konstruktor `Bruch()`, welcher den Zähler auf 0 und den Nenner auf 1 setzen soll. Speichern Sie den Source-Code unter `bruch.cpp` in das Verzeichnis `serie10`.

Aufgabe 10.2*. Erweitern Sie die Klasse `Bruch` aus Aufgabe 10.1 um die `public` Methode `void kuerzen()`, die die gekürzte Darstellung des Bruchs `zaehler/nenner` bestimmt. Dazu benütze man den euklidischen Algorithmus aus der Vorlesung. Weiters implementiere man eine Methode `setWert(string wert)`, welche eine beliebige Gleitpunktzahl in einen Bruch umwandelt. Die Zahl ist dabei als String gegeben. Um diese Methode zu erstellen können Sie wie folgt vorgehen. Zunächst sucht man in dem String nach dem Dezimalpunkt und zählt die Nachkommastellen. Dann löscht man den Dezimalpunkt aus dem String heraus. Den String, welcher nun eine natürliche Zahl repräsentiert, kann nun mittels der Funktion `atol` in eine `long` Variable umgewandelt werden. Diese Zahl benützt man als Zähler. Als Nenner verwende man 10^p wobei $p \in \mathbb{N}$ die Anzahl der Nachkommastellen sei. Speichern Sie den Source-Code unter `bruch2.cpp` in das Verzeichnis `serie10`.
Hinweis: Mit der Methode `find` der Klasse `string` können Sie nach einem bestimmten Zeichen im String suchen, z.B.: `int pos = wert.find('.')` gibt die Stelle des Dezimalpunkts im String `wert` an. Mit `wert.erase(pos,k)` werden ab der Stelle `pos` die `k` darauf folgenden Zeichen im String gelöscht. Die Funktion `atol` aus der Standardbibliothek `cstdlib` konvertiert einen gegebenen String (im C-Stil) in eine `long` Variable. Um die Zeichenkette eines Strings zu erhalten kann man die Methode `c_str()` der Klasse `string` benutzen.

Aufgabe 10.3*. Erstellen Sie eine Klasse `Name` welche zwei String-Variablen `vorname` und `nachname` enthält. Implemntieren Sie auch die Zugriffsfunktion `setName`, welche einen String übernimmt, diesen in Vor- und Nachname aufteilt und in die entsprechenden Variablen abspeichert. Beachten Sie auch, dass mehrere Vornamen vorhanden sein können! Schreiben Sie weiters eine Methode `printName` die Vor- und Nachname am Bildschirm ausgibt. Bei mehreren Vornamen soll, beginnend ab dem zweiten Vornamen, jeder weitere Vorname mit dem Anfangsbuchstaben und einem Punkt abgekürzt werden!

Beim Namen `Max Maxi Mustermann` soll also `Max M. Mustermann` am Bildschirm erscheinen. Speichern Sie den Source-Code unter `name.{hpp/cpp}` in das Verzeichnis `serie10`.

Aufgabe 10.4*. Implementieren Sie ein einfaches *Tic Tac Toe* Spiel. Falls Sie Ihnen nicht bekannt sind, können Sie die Regeln unter `http://de.wikipedia.org/wiki/Tic_Tac_Toe` nachlesen. Das Spiel soll mindestens die folgenden Kriterien erfüllen:

1. Es sollen zwei Spieler gegeneinander antreten können die jeweils abwechselnd am Zug sind.
2. Es soll automatisch erkannt werden, wann ein Spieler gewonnen hat.
3. Nach jedem Zug soll das aktuelle Spielfeld grafisch in der Konsole ausgegeben werden.
4. Kann sich am Ende keiner der Spieler durchsetzen, so soll `'unentschieden'` ausgegeben werden.

Verwenden Sie den Container `vector` für das Spielfeld, also etwa `vector<vector<char> >`. Jeder Spieler soll ein eigenes Zeichen verwenden, z.B.: `'x'` für Spieler 1 und `'o'` für Spieler 2. Das Spielfeld soll dabei ein $n \times n$ Feld mit $n \geq 3$ sein. Tipp: Planen Sie Ihre Datenstrukturen und den Programmverlauf bevor sie mit der tatsächlichen Implementierung beginnen.

Aufgabe 10.5. Schreiben Sie erneut eine Funktion `unique`, welche die doppelten Einträge aus einem Vektor entfernt (siehe Aufgabe 7.1). Verwenden Sie diesmal jedoch den Standardcontainer `vector`. Erklären Sie die Unterschiede zu Ihrer ersten Implementierung. Speichern Sie den Source-Code unter `unique.cpp` in das Verzeichnis `serie10`.

Aufgabe 10.6. Was gibt folgender Code am Bildschirm aus?

```
#include <iostream>
#include <string>
using namespace std;

class T1 {
  string t1;
public:
  T1(string val) {cout << "Ich bin Konstruktor von " << val << endl; t1=val;}
  T1() {cout << "Ich bin Konstruktor von default" << endl; t1="default";}
  ~T1() {cout << "Ich bin Destruktor von " << t1 << endl;}
};

int main() {
  T1 bert("bert");
  T1 bob;
  T1 def("bob");

  return 0;
}
```

Aufgabe 10.7. Erstellen Sie eine Klasse `SparKonto` mit den Variablen `kontonummer`, `guthaben` und `zinssatz`. Ferner sollen noch die `get` und `set` Funktionen für die Variablen `zinssatz` und `kontonummer` implementiert werden. Um das Guthaben zu ändern schreiben Sie die Methoden `abheben` und `einzahlen`. Beachten Sie, dass Sie bei einem Sparkonto nicht ins Minus gehen können. Der Zinssatz und die Kontonummer dürfen natürlich auch nicht negativ werden. Schließlich implementiere man noch die Methode `berechneGuthaben`. Speichern Sie den Source-Code unter `sparkonto.{hpp,cpp}` in das Verzeichnis `serie10`.

Aufgabe 10.8. *(Schachtelung von Klassen, Array von Klassen.)* Als Kunde einer Bank kann man auch mehrere Sparkonten besitzen. Erstellen Sie eine Klasse `Kunde` welche eine Liste von Sparkonten beinhaltet. Benutzen Sie dazu den Container `vector`. Weiters soll die Klasse auch ein Objekt der Klasse `Name` aus Aufgabe 10.3 enthalten. Implementieren Sie Methoden zum Hinzufügen und Löschen von Konten. Schreiben Sie dann eine Methode die das Guthaben auf allen vorhandenen Sparkonten berechnet. Überlegen Sie welche Funktionen noch sinnvoll wären. Speichern Sie den Source-Code unter `kunde.{hpp,cpp}` in das Verzeichnis `serie10`.