Programmieren C++: Templates, STL `vector`, File-I/O, Exceptions: File kopieren mit zufälligem Umordnen der Zeilen

Klaus Kusche

Wir wollen ein Programm schreiben, das eine angegebene *Datei zeilenweise* komplett in den Speicher *liest*, dann die *Zeilen zufällig vertauscht*, und schließlich die umgeordneten Zeilen in eine andere angegebene *Datei speichert*.

Zum internen *Speichern der eingelesenen Zeilen* verwenden wir ein Objekt der vordefiniereten *STL-Container-Klasse* **`vector`** mit Elementtyp **`string`**.
Vektoren verhalten sich in vieler Hinsicht *wie Arrays* (man kann z.B. mit **`[ ]`** auf die einzelnen Elemente zugreifen), aber sie *wachsen automatisch* nach Bedarf, d.h. man muss *beim Anlegen keine Größe* angeben.
Das ist in diesem Beispiel besonders praktisch, da wir ja vorab nicht wissen, wie viele Zeilen die einzulesende Datei enthält.

a) Funktion `scramble`

Die Funktion **`scramble`** bekommt als *Parameter* einen **`vector`** mit *beliebigem Element-Typ* (die Funktion ist daher *als Template* zu programmieren).
Wie musst du diesen Vektor-Parameter übergeben, damit die Funktion die *Elemente des Original-Vektors umordnen* kann?
Die Funktion hat keinen Returnwert.

Sie soll alle Elemente dieses Vektors *zufällig umordnen*, und zwar nach dem *Fisher-Yates-Verfahren*:

- Schreib eine *Schleife*, die mit **`i`** vom Index des *letzten Elementes* des Vektors *rückwärts* bis **`1`** zählt (die Methode **`size`** liefert dir die Größe eines Vektors).
- Ermittle in der Scheife eine *zufällige Position* **`pos`** zwischen **`0`** und **`i`** (jeweils *einschließlich*!).
- *Vertausche* das Vektor-Element an Position **`pos`** mit dem Element an Position **`i`** (du darfst das vordefinierte **`swap`** aus **`utility`** zum Vertauschen verwenden).

b) Funktion `copy_scrambled`

Das *Einlesen und Ausgeben* soll in einer Funktion **`copy_scrambled`** geschehen.
Diese Funktion hat *zwei* **`string`**-*Parameter* (die Dateinamen der zu lesenden und der zu schreibenden Datei) und *keinen* Returnwert.

- *Öffne* zuerst die beiden Dateien.
  Wirf einen **`string`** mit einem *Fehlertext als Exception*, wenn das Öffnen *fehlschlägt*.
- Deklariere einen **`vector`** mit **`string`**-Elementen als lokale Variable.
- Lies in einer *Schleife* immer wieder eine Zeile in eine **`string`**-Hilfsvariable (mit der Funktion **`getline`**, nicht mit der Methode **`getline`**) und hänge die gelesene Zeile mit der Methode **`push_back`** *hinten* an den Vektor an, bis das Lesen einen Fehler liefert.
- Ruf unsere Funktion **`scramble`** für den Vektor auf.

- *Schreib* mit einer *weiteren Schleife* alle Elemente des Vektors in die Ausgabedatei (schaffst du das auch mit einer **`for ( … : … )`** -Schleife?).

c) Hauptprogramm

Unser Hauptprogramm soll mit *zwei Argumenten auf der Befehlszeile* aufgerufen werden: Dem Dateinamen der Eingabedatei und dem Dateinamen der Ausgabedatei.

- Prüfe zuerst, ob *wirklich zwei Namen* auf der Befehlszeile angegeben wurden. Wenn nicht: Fehlermeldung, Programmende.
- Sorge dafür, dass das Programm bei jedem Programmlauf *eine andere Vertauschung* der Zeilen liefert.
- Ruf **`copy_scrambled`** auf. Falls es eine *Fehlermeldung* wirft: *Fange* sie, gib sie aus, und beende das Programm.

Zusatzaufgabe

1.) **`rand()`** und **`srand(time(NULL))`** funktionieren auch in C++,
aber C++ hat *eigene Zeit- und Zufallszahlen-Funktionen*. Versuche, diese zu verwenden!

Die Idee der Zufallszahlen in C++ (Header **`<random>`**) ist,
dass man *zuerst* einmal ein *Objekt erzeugt*, das *einen Zufallszahlen-Generator repräsentiert*.
Für solche Objekte gibt es *mehrere vordefinierte Klassen*, die diverse Zufalls-Generatoren mit verschiedenen Parametern, Verteilungen usw. implementieren.

Die einfachsten Generator-Klassen sind:
`minstd_rand` (verwendet einen *ganz primitiven Generator* ähnlich wie **`rand`**)
`default_random_engine` (der *Default-Zufalls-Generator*, meist langsamer und besser)
`random_device` (verwendet intern den *Hardware-Zufalls-Generator*)

Achtung: Mit MinGW 8.x (*CodeBlocks 20.03* auf Windows) funktioniert **`random_device`** *nicht*: Es liefert *immer dieselben Zufallszahlen* (sollte mit aktuellem MinGW behoben sein).

Im *Konstruktor* dieser Klassen kann man *optional* einen *Initialisierungswert* angeben.
Gibt man *nichts* an, liefert das erzeugte Zufalls-Objekt *bei jedem Programmlauf dieselben Zufallszahlen*. Ausnahme: **`random_device`**, das *keinen* Initialisierungswert erlaubt und auch ohne Initialisierung *jedesmal andere Zahlen* erzeugt.

Wenn wir den Generator wie bei **`srand(time(NULL))`** mit der *aktuellen Tageszeit* initialisieren wollen, geben wir im Konstruktor folgenden Wert an (braucht **`<chrono>`**):

`chrono::system_clock::now().time_since_epoch().count()`

Die so angelegten Zufalls-Generator-Objekte haben alle einen **`operator()`**,
d.h. man kann so ein *Zufallszahlen-Generator-Objekt immer wieder* wie eine Funktion *aufrufen*, und es liefert jedesmal die *nächste Zufallszahl* als Ergebnis.

2.) Es gibt seit C++11 in **`<algorithm>`** bereits eine fertige Funktion **`shuffle`**
zum Permutieren eines Containers mittels Fisher-Yates.
Versuche, den Aufruf deiner Funktion durch diese Funktion zu ersetzen!

`shuffle` möchte *zwei Iteratoren* auf *Anfang* und *Ende* des umzuordnenden Bereiches (wir wollen den gesamten **`vector`** umordnen, übergib daher dessen **`begin`** und **`end`**) und ein *Zufallszahlengenerator-Objekt* (wie oben, du kannst direkt im **`shuffle`**-Aufruf ein temporäres Objekt als Parameter erzeugen und übergeben).