Übersichtsblatt:
Dateizugriff

Oft kommt es vor, dass man in C++ Programmen auch Daten speichern muss. Um Daten in Dateien zu speichern oder sie von dort zu lesen, verwendet man in C++ die so genannten File-Streames. Um diese nutzen zu können muss am Programmanfang die Bibliothek "fstream" includiert werden.

```
#include<fstream>
```

Ist diese Bibliothek eingebunden, so kann man sich jederzeit entscheiden, ob man Zeichenweise oder mit ganzen Byte-Blöcken auf Dateien zugreifen möchte.Der zugriff auf die Datei erfolgt in dem man einen entsprechenden Stream öffnet.

Stream zum schreiben in Dateien:

```
std::ofstream out("datei.txt", <Flags>);
```

Stream zum lesen aus Dateien:

```
std::ifstream in("datei.txt", <Flags>);
```

Häufig verwendete Flags zum Öffnen von Dateien std::ios::	
`in`	lesen (Datei muss existieren)
`out`	leeren und schreiben (ggfls. Erzeugen)
`out | trunc`	leeren und schreiben (ggfls. Erzeugen)
`out | app`	anhängen (ggfls. Erzeugen)
`in | out`	lesen und schreiben (Datei muss existieren)
`in | out | trunc`	leeren, lesen und Schreiben (ggfls. Erzeugen)
`binary`	bearbeiten im Binärmodus
`ate`	nach öffnen wird ans Dateiende gegangen

Flags können angegeben werden müssen aber nicht, wenn man keine Angibt, so wird im default-Modus gearbeitet. Wenn der entsprechende Stream geöffnet ist, so überprüft man am besten zunächst ob auch alles geklappt hat. Wenn dies nicht der fall ist, kann man eine Fehlermeldung ausgeben und mit `exit(1)` abbrechen. Ansonsten kann man jetzt endlich beginnen, auf die Datei zuzugreifen.

Zeichenweise:

Schreiben:

Mit `out.put(char)` wird ein übergebenes Zeichen in die Datei geschrieben. Will man mehre Zeichen hintereinander eingeben, so empfiehlt sich die Verwendung einer Schleife.

Lesen:

Mit `in.get(char)` wird eine einzelnes Zeichen in eine übergebene Variable vom Typ `char` gelesen. Die Formulierung `while(in.get(char))` macht es möglich, so lange einzelne Zeichen zu lesen, biss man am ende der Datei angekommen ist.

Byte-Block-Weise:

Schreiben:

Dies ist interessant wenn man Datenstrukturen, wie z.B. einen Punkt mit x und y Koordinate speichern will. Dies tut man mit den folgenden Zeilen:

```
Punkt p(1,0);
out.write(reinterpret_cast<const char*>(&p), sizeof(p));
```

Die Datenstruktur Punkt wird also als String uminterpretiert und in die Datei geschrieben.

Lesen:

Nun wollen wir unseren gerade gespeicherten Punkt wieder aus der Datei laden. Dazu gehen wir ähnlich vor wie beim Schreibvorgang:

```
Punkt q;
in.read(reinterpretet_cast<char*>(&q), sizeof(q);
```

So wird aus dem gespeicherten String wieder ein Punkt.

Nach dem Schreib- bzw. Lesevorgang muss der entsprechende Stream nun nur noch geschlossen werden. (`in.close() bzw. out.close()`)

© Marco Rancinger

eTeachers.de
Lernen leicht gemacht