Kapitel 16

Datei Aus- und Eingabe

Datei Aus- und Eingabe

- Für die Ausgabe auf Dateien und die Eingabe von Dateien werden die Streams *ofstream* und *ifstream* verwendet.
- Sie sind in der Bibliothek *fstream* deklariert.
- Man unterscheidet zwischen Text- und Binärdateien.
- Textdateien enthalten ausschließlich ASCII-Zeichen.
- Sie können mit gewöhnlichen Editoren bearbeitet werden.
- C++ Quelldateien sind Textdateien.

Datei Aus- und Eingabe

- Binärdateien enthalten Daten in binärer Form.
- Diese Daten können nur mit speziellen Editoren (z.B. WinHex) sinnvoll dargestellt werden.
- Binärdateien verwendet man, um größere Datenmengen abzulegen.
- Sie benötigen hierfür nicht so viel Speicher wie Textdateien.

Ausgabe in Textdatei

- Will man Daten in eine Datei schreiben, muss zunächst ein Ausgabestream vom Typ *ofstream* definiert und mit einer Datei verbunden werden:

```
ofstream myFile("D:\\Entwicklung\\CPP\\Datei.txt");
```

- Bei der Pfadangabe ist jeder Backslash zu verdoppeln.

Ausgabe in Textdatei

- Falls die Datei noch nicht existiert, wird sie neu angelegt.
- Beim Öffnen einer Datei können Fehler auftreten.
- Diese Fehler sollten abgefangen werden.
- Zum Schreiben der Daten in diese Textdatei kann der Operator << verwendet werden.
- Für die Dateiausgabe stehen die gleichen Manipulatoren wie für *cout* zur Verfügung. (Beispielsweise: *setprecision*, *setw*)

Beispiel: Ausgabe in Textdatei

```
#include <iostream>   #include <iomanip>   #include <fstream>
using namespace std;
void main()
{
    ofstream myFile("Datei.txt");
    if(!myFile)
        cerr << "Fehler beim Öffnen der Datei";
    else
        myFile << setprecision(5) << 1.0/3.0 << endl;
    myFile.close();
}
```

Einlesen von Textdatei

- Will man Daten aus einer Datei lesen, muss zunächst ein Eingabestream vom Typ *ifstream* definiert und mit einer Datei verbunden werden:

```
ifstream myFile("Datei.txt");
```

- Die Datei muss bereits existieren.
- Zum Lesen von Daten aus dieser Textdatei kann der Operator >> verwendet werden.

Beispiel: Einlesen von Textdatei

```
#include <iostream>   #include <iomanip>   #include <fstream>
using namespace std;
void main()
{    double x;
     ifstream myFile("Datei.txt");
     if(!myFile)
          cerr << "Fehler beim Öffnen der Datei";
     else
          myFile >> x;
     myFile.close();
}
```

Die Bibliothek fstream

- Bisher haben wir die Datentypen *ifstream* und *ofstream* verwendet, die in der Bibliothek *fstream* deklariert sind.
- Diese Datentypen nennt man auch "Klassen".
- Sie stellen die benötigten Funktionen/Methoden zur Verarbeitung von Dateien zur Verfügung.

Einige Methoden von fstream

Methode	Beschreibung
eof()	Prüfung auf Dateiende
<<	Schreiben in Datei
>>	Lesen von Datei
open()	Öffnen einer Datei
close()	Schließen einer Datei
getline()	Lesen einer ganzen Zeile aus einer Datei

Erkennen des Dateiendes

- Ist nicht bekannt, wie viele Daten sich in einer Datei zum Lesen befinden, muss bis zum Dateiende gelesen werden.
- Das Erreichen des Dateiendes kann durch die Methode *eof()* geprüft werden.
- *eof()* gibt bei Erreichen des Dateiendes *true* zurück.

```
while(!myInputFile.eof())
{
    getline(myInputfile, zeile);
    cout << zeile << endl;
}
```

Öffnen von Dateien

- Um den Inhalt einer Datei lesen zu können, muss diese erst einmal geöffnet werden.
- Dies wird entweder erreicht, indem man bei der Deklaration des Eingabestreams die Datei gleich angibt:

```
ifstream myFile("Datei.txt");
```

- Oder aber man deklariert zunächst die Streamvariable und öffnet die Datei dann im nächsten Schritt:

```
ifstream myFile;
myFile.open("Datei.txt");
```

Schließen von Dateien

- Wenn die Bearbeitung einer Datei beendet ist, sollte die Datei durch *close()* geschlossen werden.

```
myFile.close();
```

- Am Ende eines Programms werden alle geöffneten Dateien automatisch geschlossen.
- Der Destruktor eines Datei-Streams schließt die Datei ebenfalls automatisch.

Lesen einer ganzen Zeile

- Durch die Methode *getline()* wird eine ganze Zeile aus der angegebenen Datei gelesen.

```
string zeile;
getline(myFile, zeile);
```

Die Bibliothek sstream

- Die Bibliothek *sstream* stellt *istringstream* und *ostringstream* zur Verfügung.
- Zur Umwandlung einer Zeichenkette in eine Zahlenvariable kann *istringstream* verwendet werden:

```
int meineZahl;
string s("123");
istringstream inputStream(s);
inputStream >> meineZahl;
```

Die Bibliothek sstream

- Soll aus einer Zahlenvariablen eine Zeichenkette werden, so kann *ostringstream* verwendet werden:

```
int meineZahl = 123;
string s;
ostringstream outputStream;
outputStream << meineZahl;
s = outputStream.str();
```

- Die Methode *str()* wandelt den Stream in einen String um und schreibt ihn in s.

Die Methode str()

- Einerseits wandelt also *str()* den Stream in einen String um:

```
string s = outputStream.str();
```

- Andererseits kann so, ein String dem Stream zugewiesen werden, um ihn dann durch >> in eine Variable zu schreiben:

```
getline(myFile, zeile);
istringstream iss.str(zeile);
iss >> name;
```

Die Methode str() - Fortsetzung

- Warum wird der String durch *getline*() nicht gleich in *name* eingelesen (statt - wie im Beispiel - in *zeile*) ?
- Man beachte, dass im eingelesen String *zeile* auch Leerzeichen enthalten sein können. Durch dem Umweg über *istringstream* ist es dadurch nun möglich, den String durch >> zu trennen und die Teilstrings verschiedenen Variablen zuzuweisen.
- Analog zu *cin*, welches bei der Eingabe auch den Gebrauch von

```
cin >> a >> b >> c;
```

- erlaubt, bei der man die Werte durch Leerzeichen getrennt hintereinander schreiben kann.

Die Methode clear()

- Die Methode *clear*() löscht den zugewiesenen String, so dass eine neue Zuweisung erfolgen kann.

```
iss.clear();
```

Vollständiges Schreiben einer Textdatei

- Im ersten Teil des folgenden Beispiels wird eine Datei *Namen.txt* zum Schreiben geöffnet, ein *string*-Feld mit Namen und Ident-Nummer angelegt und komplett element- und zeilenweise in die Datei *Namen.txt* geschrieben.
- Danach wird die Datei geschlossen.

=> siehe
"Beispiel_18_-_Datei_-_Aus_-_und_Eingabe.zip" – 1. Teil

Bsp: Vollständiges Lesen einer Textdatei

- Im zweiten Teil des Beispiels wird die Datei *Namen.txt* zum Lesen geöffnet und bis zum Erreichen des Dateiendes zeilenweise ausgelesen.
- In jeder Zeile wird die Ident-Nummer vor den Namen gestellt und das *string*-Feld wieder beschrieben und ausgegeben.
- Am Ende wird die Datei wieder geschlossen.

=> siehe
"Beispiel_18_-_Datei_-_Aus_-_und_Eingabe.zip" – 2. Teil

Einige Methoden von string

Methode	Beschreibung
find(s)	Liefert die Position, an der sich der String s befindet. Das Zählen beginnt hier – wie bei Arrays – bei 0. Die Methode liefert string::npos, wenn s nicht gefunden wurde. string::npos steht für den Wert -1
insert(n, s)	Fügt den String s an Position n ein.
erase(p, n)	Entfernt ab Position p n Zeichen.
length()	Liefert die Länge des Strings.

Einige Methoden von string - Bsp

```
string s("123");            // String s erstellen mit Zeichenkette "123"
int len = s.length();       // liefert 3; Die Länge des Strings
s.insert(2, "xy");          // s ist nun "12xy3"
s.erase(2, 2);              // s ist nun "123"
int pos = s.find("23");     // liefert die Position 1
```

Siehe dazu auch: Skript Informatik - Kapitel 12 (Strings)