Arbeiten mit Dateien Teil 1: Grundlagen und Funktionsübersicht

== Vorwort ==
Wenn man mit C++ programmiert wird man früher oder später nicht Drumherum kommen irgendwelche Dateien zu lesen oder zu beschreiben.
Dafür gibt es mittlerweile viele Möglichkeiten, ich werde hier nur die von der C++- Standardbibliothek (wird ab jetzt mit **CPPLIB** abgekürzt) bereitgestellte Möglichkeit besprechen und nur dieser sollte auch verwendet werden, da diese Plattformunabhängig ist und wer damit perfekt umgehen lernt, wird bei Portierungen keine Probleme haben. Genauso sollte hier gewonnenes Wissen auf die äquivalenten C-Funktionen übertragen werden können.

Über diese Artikelreihe:
Optionale Parameter werden mit Eckigen Klammern [] versehen, also solche die man nicht zwingen Angeben muss.

== Grundlagen ==
Die wichtigen Themen werden hier nur kurz Angeschnitten und sollte gegeben falls über externe Quellen vertieft werden.

=== Was ist eigentlich eine Datei? ===
Eine Datei wird durch ihren Speicherort, ihren Namen und ihren Inhalt definiert.

Der Aufbau des Speicherorts, sowie der Name der Datei kann von Betriebssystem zu Betriebssystem variieren und sieht unter Windows z.B.: so aus:
C:\Programme\Max\Hallo.txt
-> Datenträger Buchstabenidentifikation
-> Ordnerstruktur, einzelne Unterordner werden durch ein Backslash \ getrennt (hingegen bei Linux einfach nur Slash /)
-> Dateiname
-> Datei(namens)Erweiterung, deutet oft auf den gespeicherten Inhalt hin

Der Inhalt hat eine Variable Länge und wird durch verschiedene Bytefolgen bestimmt:
[Blockierte Grafik: http://img855.imageshack.us/img855/9540/datei.jpg]
In diesem Bild sieht man eine Datei der Länge 16, im linken Teil des Fensters werden die Hexadezimal-Werte der einzelnen Bytes dargestellt und rechts die dazugehörige ASCII-Zeichen.

Siehe dazu auch [wikipedia]Datei[/wikipedia].

=== Wie ist ein Byte definiert? ===
Ein Byte hat 8 Bit und ein Bit ist entweder eine 1 oder eine 0.
In C++ gibt es verschiedene Datentypen die durch verschiedene Anzahl an Bytes repräsentiert werden, um diese Anzahl zu erhalten gibt es die **sizeof** Funktion.
Siehe dazu auch [wikipedia]Byte[/wikipedia] und [wikipedia]Bit[/wikipedia].

=== Was ist der Unterschied zwischen Binär- und Textdatei? ===
Im Prinzip gibt es keine, jedoch wird in der Programmierung oft zwischen diesen zwei Dateien, besser gesagt Modi unterschieden.
Im Binärmodus werden die Daten wie sie sind geschrieben, also Roh.
Im Textmodus werden nach Möglichkeiten alle Daten bzw. die übergebenden Datentypen in ihre ASCII-Repräsentation umgewandelt, selbe gilt in umgekehrter Richtung beim einlesen.

=== Was ist ein ASCII-Zeichen bzw. eine Tabelle? ===
Jedem Zeichen wie Buchstaben und Zahlen sind festgelegte Werte zugewiesen um in der Computerwelt damit einheitlich arbeiten zu können und damit die Darstellung sicherzustellen.
Siehe dazu auch [wikipedia]ASCII[/wikipedia].

=== Zahlensysteme ===
In der Mathematik gibt es verschiedene Zahlensysteme, das bekannteste wird wohl das Dezimalsystem sein.
Des Weiteren gibt es noch Binär, Oktadezimal, Hexadezimal und viele andere, dies sind jedoch die wichtigsten.
Siehe dazu auch [wikipedia]Dualsystem[/wikipedia], [wikipedia]Oktalsystem[/wikipedia] und

[wikipedia]Hexadezimalsystem[/wikipedia].

== Einleitung ==
In der CPPLIB werden die Dateioperationen in drei Objekte unterteilt und zwar in Ausgabeobjekte(**ofstream**), Eingabeobjekte(**ifstream**) und in Ein-/Ausgabeobjekte(**fstream**), diese sollten nach ihrem Aufgabengebiet ausgewählt werden um einmal Ressourcen zu sparen und um logische Fehler zu vermeiden.
Des Weiteren unterscheidet man zwischen **Binärdateien** und **Textdateien**, für Anfänger ist wohl letzteres interessanter und alle genannten Objekte werden standardmäßig in diesem Modus geöffnet, dabei werden die Standard Typen automatisch in Zeichenketten umgewandelt.

Nun um die oben genannte Objekte verwenden zu können muss man zuerst den dazugehörigen CPPLIB-Header einbinden:

Quellcode

```
1. #include <fstream>
2. using namespace std;
```

Wer an dieser Stelle schon den CPPLIB-Header **iostream** eingebunden hat kann die Einbindung von fstream weglassen, da diese von iostream automatisch eingebunden wird.
Der Befehl "using namespace std" muss nur einmal (pro Gültigkeitsbereich definiert) werden, er bewirkt dass man auf namespace-Objekte, wie die Streamklassen direkt zugreifen kann, wer weiteres wissen will kann sich über den *scope operator* selbstständig erkundigen.

== Funktionsübersicht ==
Die Funktionen der Objekte **ofstream**, **ifstream** und **fstream** sind nahezu identisch bis auf die Tatsache das **ofstream-Objekte** nur zur Dateieingabe, die **ifstream-Objekte** nur zur Dateiausgabe und **fstream-Objekte** für beides benutzt werden können und dementsprechend nur für das Objekt relevanten Funktionen besitzen.

=== Der Konstruktor ===
Wird verwendet um ein Stream-Objekt zu erstellen und bei Bedarf um direkt eine Datei zu öffnen:

Quellcode

```
1. fstream stream[ (const char * Dateiname [, openmode Mode = ios_base::in | ios_base::out]) ];
2. ifstream stream[ (const char * Dateiname [, openmode Mode = ios_base::in]) ];
3. ofstream stream[ (const char * Dateiname [, openmode Mode = ios_base::out]) ];
```

Parameter Dateiname: Muss ein C-String sein, also eine nullterminierte Zeichenkette sein.
Parameter Mode: Hat je nach Objekt verschiedene Standartwerte (siehe oben). Die Eigenschaften können mit einer **oder**-Verknüpfung | logisch kombiniert werden. Mögliche Werte:

- ios_base::**app** (**app**end) Setzt den Dateipositionszeiger vor jeder Ausgabe ans Ende der Datei.
- ios_base::**ate** (**at e**nd) Setzt den Dateipositionszeiger nach dem öffnen ans Ende der Datei.
- ios_base::**binary** (**binary**) Öffnet die Datei im **Binär-Modus**.
- ios_base::**in** (**in**) Öffnet die Datei im **Lese-Modus** (Eingabe).
- ios_base::**out** (**out**) Öffnet die Datei im **Schreib-Modus** (Ausgabe).
- ios_base::**trunc** (**trunc**ate) Der Inhalt der geöffneten Datei wird gelöscht.

Anmerkung:
Das **ostream-Objekt** löscht mit den Standartparametern immer den Inhalt der zu öffnenden Datei, obwohl ios_base::trunc nicht als Standartwert angeben ist. Wenn man dies nicht möchte sollte man deswegen an dieser Stelle aufpassen und explizit ios_base::app oder ios_base::in angeben.

=== Die Memberfunktion open() ===
Wird zum Öffnen von Dateien verwendet:

Quellcode

1. fstream.open (const char * Dateiname [, openmode Mode = ios_base::in | ios_base::out]);
2. ifstream.open(const char * Dateiname [, openmode Mode = ios_base::in]);
3. ofstream.open(const char * Dateiname [, openmode Mode = ios_base::out]);

Die Parameter sind dieselben wie beim Konstruktor, siehe weitere Infos **Konstruktor**.

=== Die Memberfunktion close() ===
Wird zum Schließen einer geöffneten Datei verwendet:

Quellcode

1. fstream.close ();
2. ifstream.close();
3. ofstream.close();

Sollte immer dann Aufgerufen werden wenn keine Schreib- und/oder Leseoperationen mehr vorliegen.

=== Die Memberfunktion is_open() ===
Wird zur Statusüberprüfung verwendet, genauer ob eine Datei geöffnet werden konnte oder nicht:

Quellcode

1. fstream.is_open ();
2. ifstream.is_open();
3. ofstream.is_open();

Die Funktion gibt **true** zurück wenn eine Datei offen ist und **false** wenn keine Datei offen ist bzw. nicht geöffnet werden konnte.

=== Positionierung ===
Die Positionierungsfunktionen verwendeten einen dynamischen **integralen Datentypen streampos**, der Einfachheit halber kann hier **[unsigned] int** angenommen werden.

==== Die Memberfunktion tellp() und tellg() ====
Werden verwendet um die aktuelle **Schreibposition (tellput)** bzw. die aktuelle **Leseposition (tellget)** zu **ermitteln**:

Quellcode

1. fstream.tellg();
2. fstream.tellp();
3. ifstream.tellg();
4. ofstream.tellp();

Rückgabe ist die aktuelle Position vom Typ streampos.

==== Die Memberfunktion seekp() und seekg() ====
Werden verwendet um die aktuelle **Schreibposition (seekput)** bzw. die aktuelle **Leseposition (seekget)** zu **verschieben** bzw. zu **positionieren**:

Quellcode

1. ifstream.seekg(streampos Position)
2. ifstream.seekg(streampos Offset, seekdir Start);
3. ofstream.seekp(streampos Position)
4. ofstream.seekp(streampos Offset, seekdir Start);

Diese Funktionen sind auch beim **fstream**-Objekt verfügbar.
Parameter Position: Gibt die absolute Position, vom Anfang der Datei an.
Parameter Offset: Relative Positionsverschiebung, von Start abhängig. (Negative Werte erlaubt)
Parameter Start: Gibt die Startposition an. Gültige Werte, können nicht kombiniert werden:

- ios_base::**beg** (**beg**in) Vom Anfang der Datei.
- ios_base::**cur** (**cur**rent) Von der aktuellen Position, siehe *tellp/tellg*.
- ios_base::**end** (**end**) Vom Ende der Datei.

=== Schreiben ===
==== Die Memberfunktion put() ====
Wird verwendet um einzelne Bytes an die aktuelle Schreibposition zu schreiben, dabei wird, falls vorhanden, das Byte überschrieben und die **Schreibposition** um eins verschoben:

Quellcode

1. fstream.put (char Byte);
2. ofstream.put(char Byte);

Parameter Byte: Das zu schreibende Byte.

==== Die Memberfunktion write() ====
Wird verwendet um Datenblöcke an die aktuelle Schreibposition zu schreiben, dabei wird, falls vorhanden, Daten überschrieben und die **Schreibposition** um die Datenblocklänge verschoben:

Quellcode

1. fstream.write (const char *Daten, streamsize Laenge);
2. ofstream.write(const char *Daten, streamsize Laenge);

Parameter Daten: Ein Zeiger im Speicher auf die zu schreibenden Daten.
Parameter Länge: Die Länge der zu schreibenden Daten. Dynamischer Integraler Datentyp kann der Einfachheit halber als **int** angesehen werden.

==== Der Schreiboperator ====
Der **Schreib-Operator <<** ist ein mächtiges Werkzeug um alle gängigen Typen in eine Datei zu schreiben sowie benutzerdefinierte Typen elegant zu schrieben, auf letzteres wird in den folgenden Artikel noch genauer eingegangen.
Die Daten werden an die aktuelle Schreibposition geschrieben, vorhandene Daten werden gegebenenfalls überschrieben und die **Schreibposition** wird um die Datengröße automatisch verschoben.

Quellcode

1. fstream << bool(true) << int(8) << float(3.14) << (void*)(0xFFFF) << char('Z') << "Hello World" << endl;
2. ofstream << bool(true) << int(8) << float(3.14) << (void*)(0xFFFF) << char('Z') << "Hello World" << endl;

Wie man sieht kommt der Schreiboperator dem der Konsole (**cout**) nahe und man kann größtenteils auch die Befehle der Konsole verwenden.

=== Lesen ===

==== Die Memberfunktion get() ====
Wird verwendet um ein Byte von der aktuellen Position einzulesen, dabei wird die **Leseposition** um eins verschoben:

Quellcode

1. fstream.get ();
2. ifstream.get();

Rückgabe ist eine **int**-Repräsentation des eingelesenen Bytes.

Alternativer Funktionsaufruf:

Quellcode

1. fstream.get (char &Byte);
2. ifstream.get(char &Byte);

Parameter Byte: Variable in die das eingelesene Byte gespeichert werden soll.

==== Die Memberfunktion getline() ====
Wird größtenteils bei Textdateien verwendet um eine Zeile, beginnend von der aktuelle Leseposition, einzulesen, dabei wird die **Leseposition** automatisch um die Anzahl der gelesenen Bytes verschoben:

Quellcode

1. fstream.getline (char* Daten, streamsize Laenge [, char Trennzeichen]);
2. ifstream.getline(char* Daten, streamsize Laenge [, char Trennzeichen]);

Parameter Daten: Ein Zeiger im Speicher an dem die gelassenen Daten abgelegt werden sollen.
Parameter Länge: Größe des Speicherbereichs der Daten. Dynamischer Integraler Datentyp kann der Einfachheit halber als **int** angesehen werden.
Parameter Trennzeichen: Optional, Zeichen bis zu welchem eingelesen werden soll. Falls nicht angegeben wird das betriebssystemabhängige Zeichen für Zeilenumbruch verwendet.

==== Die Memberfunktion read() ====
Wird verwendet um Datenblöcke von der aktuellen Leseposition zu lesen, dabei wird die **Leseposition** um die Anzahl der gelesenen Bytes verschoben:

Quellcode

1. fstream.read (char *Daten, streamsize Leange);
2. ifstream.read(char *Daten, streamsize Leange);

Parameter Daten: Ein Zeiger im Speicher an dem die gelesenen Daten abgelegt werden sollen, muss größer gleich der Größe der zu lesenden Datengröße sein.
Parameter Länge: Die Länge der zu lesenden Daten. Dynamischer Integraler Datentyp kann der Einfachheit halber als **int** angesehen werden.

==== Der Leseoperator ====
Der **Lese-Operator >>** ist ein mächtiges Werkzeug um alle gängigen Typen aus einer Datei zu lesen sowie benutzerdefinierte Typen elegant zu lesen, auf letzteres wird in den folgenden Artikel noch genauer eingegangen.
Die Daten werden von der aktuellen Leseposition gelesen und die **Leseposition** wird um die Datengröße automatisch verschoben.

Quellcode

1. fstream >> Bool >> Int >> Char;
2. ifstream >> Bool >> Int >> Char;

Wie man sieht kommt der Leseoperator dem der Konsole (**cout**) nahe und man kann diesen größtenteils genauso handhaben.

== Weitere Artikel ==

- Arbeiten mit Dateien Teil 2: Textdateien (demnächst)