Im Windows Betriebssystem sind Dateien eindeutig gekennzeichnet durch ihren Pfad, Dateinamen und nach einem Punkt die Dateierweiterung.

```
z.B.: c:\testdateien\text.dat    //Datendatei
```

Mit der Dateierweiterung ist i.a. eine Verknüpfung mit einem entsprechenden Programm vorgesehen. Die Vereinbarung über die Dateierweiterung sind jedoch nicht allgemein verbindlich.

Für den Lese-/oder Schreibzugriff auf eine Datei muss die Datei grundsätzlich mit einer **`fopen()-`** Anweisung geöffnet werden.

Funktionsdeklaration der fopen()-Funktion:

```
fopen_s( &FILE* stream, const char* filename, const char* mode );
```

Der Rückgabewert der Funktion ist 0, wenn die Datei korrekt geöffnet werden konnte. Der Übergabeparameter 'stream' ist ein Zeiger auf eine Struktur **`FILE`**, welche von VC++ global definiert ist. Die Elemente der Struktur sind i.a. ohne Bedeutung. Wichtig ist nur der Zeiger, über welchen alle Zugriffe auf die Datei erfolgen.

Weitere Übergabeparameter sind der Dateiname mit kompletter Pfadangabe und der Zugriffsmode, welcher die Art des Zugriffes festlegt. Auch der Wert für **`mode`** ist als String zu übergeben.

Beispiel für den Zugriffsmode:[1]

“r”	Öffnet Datei zum Lesen. Wenn die Datei nicht existiert liefert fopen den Wert NULL
“w”	Öffnet Datei zum Schreiben. Wenn die Datei nicht existiert wird sie angelegt. Eine vorhandene Datei wird überschrieben.
“a”	Öffnet Datei zum Schreiben. Die geschriebenen Daten werden an die Datei angehängt (append). Wenn die Datei nicht existiert wird sie neu angelegt.
“r+”	Öffnet Datei zum Lesen und Schreiben. Wenn die Datei nicht existiert liefert fopen den Wert NULL
“w+”	Öffnet Datei zum Lesen und Schreiben. Wenn die Datei nicht existiert wird sie angelegt.
“a+”	Öffnet Datei zum Lesen und Schreiben. Die geschriebenen Daten werden an die Datei angehängt (append). Wenn die Datei nicht existiert wird sie neu angelegt.
“…t”	Öffnen als Textdatei, d.h. LF-Zeichen werden mit CR-Zeichen ergänzt
„…b“	Öffnen als binäre Datendatei

Um die folgenden Beispiel testen zu können muss auf Laufwerk c: ein Verzeichnis **`‚testdateien’`** eingerichtet sein! Testen Sie das Beispiel und prüfen Sie nach, ob die Datei angelegt wurde! Untersuchen Sie das Verhalten des Programms, wenn eine falsche Pfadangabe angegeben wird! Wodurch wird der Laufzeitfehler verursacht?

[1] Die Angaben erheben keinen Anspruch auf Vollständigkeit. Weitere Möglichkeiten sind dem Online-Hilfesystem von VS zu entnehmen.

Beispiel zu fopen():

```
#include <stdio.h>
#include <conio.h>
int main()
{
      FILE* stream;
      //Der frei gewählte Bezeichner stream ist ein Zeiger auf eine FILE-Struktur
      //Achtung Pfadangaben mit '\\'
      if(fopen_s(&stream,"c:\\testdateien\\text.txt", "w+" ) == 0 )
            printf( "Die Datei 'text.txt' wurde geoeffnet!\n" );
      else
            printf( "Die Datei 'text.txt' wurde nicht geoeffnet!\n\n" );
      //Dateien sollten am Programmende ordnungsgemäß geschlossen werden
      fclose(stream);
      _getch();
      return(0);
}
```

Textdatei schreiben:

Für das Schreiben und Lesen von Dateien stehen analog zu den Konsolenfunktionen **`printf(), scanf_s(), gets_s();…`** für die Ein-Ausgabe die Funktionen **`fprintf(), fscanf(),fgets(),…`** zur Verfügung. Gegenüber den Konsolenfunktionen ist als zusätzlicher Parameter ein Zeiger auf einer Datei erforderlich.

Die Funktion **`fprintf()`** schreibt Daten in einen Stream.

```
int fprintf( FILE *stream, const char *format [, argument ]... );
```

Übergabeparameter sind ein Zeiger auf eine Datei, ein Formatstring (analog zu printf()) und die zu schreibenden Datenfelder.

Das folgende Programm öffnet eine Datei zum Schreiben und speichert den eingegebenen Text in die Datei.

```
#include <stdio.h>
#include <conio.h>
int main()
{
      FILE* stream;
      char text[100];
      if ((fopen_s(&stream,"c:\\testdateien\\text.txt", "w+" )) != 0 )
            printf( "Die Datei 'text.txt' wurde nicht geoeffnet!\n" );
      else
      {
            printf( "Die Datei 'text.txt' wurde geoeffnet!\n" );
            printf("\nText: ");
            gets_s(text);
            fprintf(stream,"%s",text);
            fclose(stream);
      }
      _getch();
      return(0);
}
```

Testen Sie das Programm und lesen Sie die Datei einmal mit dem Editor des Betriebssystems und mit einem HEX-Editor (z.B.: http://hexedit.nextsoft.de/)!

Aufgaben:

- Ersetzen Sie den fopen_s()-Befehl durch `fopen_s(&stream,"c:\\testdateien\\text.txt","a")` und testen Sie das Ergebnis!

- Ersetzen Sie in obigem Beispiel die `fprintf()`-Anweisung durch `fprintf(stream,"%20s",text);` ! Betrachten Sie das Ergebnis mit dem Editor und dem HEX-Editor!

- Ändern Sie das obige Beispiel derart ab, dass der Dateiname beliebig über Tastatur eingegeben werden kann!

- Schreiben Sie ein Programm, welches die Eingabe eines beliebigen Dateinamens gestattet und überprüft, ob die Datei existiert und eine entsprechende Meldung ‚Datei existiert!' oder ‚Datei existiert nicht!' ausgibt!

Textdatei lesen:

Analog für die Konsolenfunktion **`gets()`** steht für das Lesen eines Strings aus einer Datei die Funktion **`fgets()`** zur Verfügung.

```
char *fgets( char *str, int n, FILE *stream );
```

Übergabeparameter der Funktion ist ein Zeiger auf einen String, wo der gelesene Text stehen soll, die Anzahl n der maximal zu lesenden Zeichen und ein Zeiger auf die zu lesende Datei.

Wenn das Dateiende erreicht wird oder CR/LF gelesen wird, wird automatisch ein Begrenzungszeichen 00_H angehängt.

Das folgende Programm liest einen String aus einer Datei **`text.txt`** und gibt diesen String aus.

```
#include <stdio.h>
#include <conio.h>
int main()
{
      FILE* stream;
      char text[100];
      if (fopen_s(&stream,"c:\\testdateien\\text.txt", "r" ) != 0 )
            printf( "Die Datei 'text.txt' wurde nicht geoeffnet!\n" );
      else
      {
            printf( "Die Datei 'text.txt' wurde geoeffnet!\n" );
            fgets(text,100,stream);
            printf("\nText: \n%s",text);
            fclose(stream);
      }
      _getch();
      return(0);
}
```

Schreiben Sie einen (mehrzeiligen) Text mit dem Editor des Betriebssystems und lesen Sie den Inhalt dieser Datei mit obigem Programm! Was fällt Ihnen auf?

Um mehrzeilige Texte zu lesen, muss die Datei zeichenweise bis zum Erreichen des Dateiendes gelesen werden.

Die Funktion **`fgetc()`** liest ein einzelnes Zeichen vom **`stream`** (analog **`_getch()`** für die Konsole). Die Funktion **`fgetc()`** hat als Rückgabewert den ASCII-Code des gelesenen Zeichens.

```
int fgetc( FILE *stream );
```

Das Ende einer Datei lässt sich mit der Funktion **`feof()`** ermitteln.

```
int feof( FILE *stream );
```

Die Funktion liefert einen Wert von nicht 0, wenn das Dateiende erreicht ist!

Folgende Schleife wird ausgeführt, bis das Dateiende erreicht ist:

```
while(!feof(stream))
{
. . .
}
```

Analysieren Sie das folgende Programm und testen Sie es wie oben mit mehrzeiligem Dateiinhalt!

```
#include <stdio.h>
#include <conio.h>
int main()
{
	FILE* stream;
	char text[100];
	int i=0;
	if(fopen_s(&stream,"c:\\testdateien\\text.txt", "r" ) != 0)
		printf( "Die Datei 'text.txt' wurde nicht geoeffnet!\n" );
	else
	{
		printf( "Die Datei 'text.txt' wurde geoeffnet!\n\n" );
		while(!feof(stream))
		{
		text[i]=fgetc(stream);
		i++;
		}
		text[i]=0;	//Stringende einfügen
		printf("\nText: \n%s",text);
		fclose(stream);

	}
	_getch();
	return(0);
}
```

Aufgabe:

Entwickeln Sie ein Programm, welches die Eingabe von zwei Dateinamen gestattet und beide Dateien zum Lesen öffnet.

Anschließend soll festgestellt werden, ob der Inhalt beider Dateien identisch ist. Als Ergebnis zeigt das Programm ‚Dateien sind gleich!' oder ‚Dateien sind nicht gleich!'.

Binäre Daten schreiben und lesen:

Bei der Speicherung binärer Daten werden Datenfelder mit fester Länge gespeichert. Grundlegende Datentypen wie **`int, float, double,`** … und auch **Strukturen** und **Klassen (s.u. bei C++)** haben eine feste Datenlänge. Die Länge eines Typs wird mit der Funktion **`sizeof()`** ermittelt. Zum Schreiben und Lesen binärer Daten aus einem Stream stehen die Funktionen **`fwrite()`** bzw. **`fread()`** zur Verfügung.

```
int fwrite( *buffer, int size, int count, FILE *stream );
```

*buffer	Zeiger auf die zu speichernden Daten
size	Länge eines Datenfeldes in Byte
count	Anzahl der zu speichernden Datenfelder
*stream	Zeiger auf Datei in der gespeichert wird

Der Rückgabewert ist die Anzahl der gespeicherten Datenfelder.

```
int fread( *buffer, int size, int count, FILE *stream );
```

*buffer	Zeiger auf Speicherstelle
size	Länge eines Datenfeldes in Byte
count	Max. Anzahl der zu lesenden Datenfelder
*stream	Zeiger auf Datei aus der gespeichert wird

Der Rückgabewert ist die Anzahl der tatsächlich gelesenen Datenfelder.

Bei der **`fopen()`**-Anweisung ist zu beachten, dass die Datei im Binärmodus geöffnet wird. Zu diesem Zweck muss bei dem Zugriffsmode ein Suffix **„…b“** angehängt werden.

Beispiel:

```
FILE *stream;
stream = fopen("&stream,c:\\testdateien\\binaer.dat", "w+b");
```

Bei binären Daten wird nicht unterschieden, ob es sich um numerische Daten oder um Texte handelt. Der Inhalt der Speicherplatzadressen ab *buffer wird 1:1 in die Datei übernommen. Ein Lesen der Datei mit einem Texteditor ist nicht sinnvoll. Zur Analyse betrachten Sie den Inhalt der Datei mit einem HEX-Editor!

Das folgende Programm gestattet die Eingabe von 10 Integer-Werten in ein Datenfeld, speichert die Werte in eine Datei und liest sie anschließend wieder aus.

```
#include<stdio.h>
#include<conio.h>
int main()
{
      int wert[10];
      int i;
      for (i=0;i<10;i++)
      {
            printf("\n%i. Wert: ",i+1);
            scanf_s("%i",wert+i);
      }
      FILE *stream;

/* Öffne Binäre Datei zum Schreiben */
      if (fopen_s(&stream,"c:\\testdateien\\binaer.dat", "w+b" ) != 0 )
      {
            printf( "Datei kann nicht geöffnet werden!\n" );
            return(1);
      }

/* Schreibe 10 int-Werte vom Datenfeld 'wert' in die Datei */
      fwrite(wert,sizeof(int),10,stream);
      fclose(stream);

/* Öffne Binäre Datei zum Lesen */
      if (fopen_s(&stream,"c:\\testdateien\\binaer.dat", "rb" ) != 0 )
      {
            printf( "Datei kann nicht geöffnet werden!\n" );
            return(1);
      }

/* Lese 10 int-Werte aus der Datei und lege diese im Datenfeld 'wert' ab */
      fread(wert,sizeof(int),10,stream);
      printf("\nGelesene Werte:\n");
      for (i=0;i<10;i++)
      {
            printf("%i. Wert: %i\n",i+1,wert[i]);
      }
      fclose(stream);
      _getch();
      return(0);
}
```

Das nächsten Beispiele demonstrieren das Speichern und Lesen eines Datenfeldes mit einer Struktur.

```
#include<stdio.h>
#include<conio.h>
struct artikel
{
      int nr;
      char name[28];
      double preis;
};
int main()
{
      artikel teil[5];
      artikel hilf;
      int i;
      for(i=0;i<5;i++)  //Eingabe von 5 Datensätzen
      {
            printf("\n%i.Datensatz: \n",i+1);
            printf("\nNr: ");scanf_s("%i",&hilf.nr);
            printf("\nName: ");_flushall();gets_s(hilf.name);
            printf("\nPreis: ");scanf_s("%lf",&hilf.preis);
            teil[i]=hilf;
      }
      for(i=0;i<5;i++)  //Zeige die Datensätze
      {
            printf("\n%4i %24s %8.2lf",(teil+i)->nr,(teil+i)->name,(teil+i)->preis);
      }

      FILE *stream;
      if (fopen_s(&stream,"c:\\testdateien\\structur.dat", "w+b" ) != 0 )
      {
            printf( "Datei kann nicht geöffnet werden!\n" );
            return(1);
      }
      fwrite(teil,sizeof(artikel),5,stream); //Schreibe 5 Datensätze der Struktur teil
      fclose(stream);
      printf("Daten gespeichert!\n");
      _getch();
      return(0);
}
```

Und schließlich das Beispiel zum Lesen der Datei:

```
#include<stdio.h>
#include<conio.h>
struct artikel
{
      int nr;
      char name[28];
      double preis;
};
int main()
{
      artikel teil[5];
      int i;
      FILE *stream;
      if (fopen_s(&stream,"c:\\testdateien\\structur.dat", "rb" ) != 0 )
      {
            printf( "Datei kann nicht geöffnet werden!\n" );
            return(1);
      }
      fread(teil,sizeof(artikel),5,stream);    //Lese 5 Datensätze
      for(i=0;i<5;i++)  //Zeige die Datensätze
      {
            printf("\n%4i %24s %8.2lf",(teil+i)->nr,(teil+i)->name,(teil+i)->preis);
      }
      fclose(stream);
      _getch();
      return(0);
}
```

Übung - Artikelverwaltung

Entwickeln Sie ein C-Programm für die Verwaltung einer Artikeldatei gemäß folgender Vorgaben!

Legen Sie global einen benutzerdefinierten Datentyp **`artikel`** an:

```
struct artikel
{
int nr;              //Artikelnummer
char name[24];       //Artikelbezeichnung
int anzahl;          //Artikelanzahl
double preis;        //Artikelpreis
};
```

Im Hauptprogramm (main()-Funktion) werden folgende Datentypen vereinbart:

```
...
artikel teil[100];    //Ein Datenfeld in welchem bis zu 100 Datensätze vom Typ
                      //artikel eingegeben werden können
int i;     //Anzahl der tatsächlich eingegebenen Datensätze
...
```

Die Bedienung des Programms erfolgt menügesteuert. Nach Start des Programms soll folgendes Auswahlmenü erscheinen:

```
Daten eingeben  = '1'
Daten ausgeben  = '2'
Daten speichern = '3'
Daten laden     = '4'
Daten sortieren = '5'
Daten ändern    = '6'
Daten suchen    = '7'
Daten löschen   = '8'

Ende = '0'

Bitte wählen...
```

Die Funktionalität des Programms erfolgt ausschließlich durch Aufrufen von Funktionen in Abhängigkeit von dem gewählten Menüpunkt. Für die einzelnen Menüpunkte werden folgende Funktionen deklariert:

1. int eingeben(artikel* , int);

   Übergabeparameter: Zeiger auf den Beginn des Datenfeldes, Anzahl der vorhandenen Datensätze
   Rückgabewert: Anzahl der eingegebenen Datensätze.

   Die Funktion kann zur Laufzeit des Programms öfter aufgerufen werden. Die Anzahl der Datensätze (Variable i des Hauptprogramms) wird bei jedem Aufruf um die Anzahl der eingebenen Datensätze erhöht.

2. `void ausgeben(artikel* , int);`

   Übergabeparameter: Zeiger auf den Beginn des Datenfeldes, Anzahl der vorhandenen Datensätze

   Die Funktion gibt alle Datensätze formatiert auf den Bildschirm aus. Die einzelnen Datensätze sind fortlaufend nummeriert, beginnend mit 1 (Index).

3. `void speichern(artikel* , int);`

   Übergabeparameter: Zeiger auf den Beginn des Datenfeldes, Anzahl der vorhandenen Datensätze

   Die Funktion speichert die Datei unter dem Namen **`D:\testdateiein\artikel.dat`**

4. `int laden(artikel*);`

   Übergabeparameter: Zeiger auf den Beginn des Datenfeldes
   Rückgabewert: Anzahl der gelesenen Datensätze

   Die Funktion liest die Datei (sofern vorhanden) **`D:\testdateieien\artikel.dat`** in das Datenfeld **`teil[100]`** ein und legt die Anzahl der gelesenen Datensätze in der Variablen `i` ab!

5. `void sortieren(artikel* , int);`

   Übergabeparameter: Zeiger auf den Beginn des Datenfeldes, Anzahl der vorhandenen Datensätze

   Die Funktion sortiert wahlweise nach Artikelnummer oder Artikelname. Das Datenfeld wird nur in sich sortiert. Die Ausgabe der sortierten Datei erfolgt mit dem Menüpunkt '2' (Anzeigen).

6. `void aendern(artikel* . int);`

   Übergabeparameter: Zeiger auf den Beginn des Datenfeldes, Anzahl der vorhandenen Datensätze

   Die Funktion erwartet die Eingabe des Index eines Datensatzes (vgl. Menüpunkt '2'). Dieser Datensatz wird dann angezeigt und kann neu eingegeben werden. Liegt der Index außerhalb des gültigen Bereiches soll eine Fehlermeldung erfolgen.

7. `void suchen(artikel* , int);`

   Übergabeparameter: Zeiger auf den Beginn des Datenfeldes, Anzahl der vorhandenen Datensätze

   Die Funktion erwartet wahlweise zur Eingabe eine Artikelnummer oder einen Artikelnamen. Danach werden alle Datensätze angezeigt, die dieser Eingabe (genau) entsprechen.

8. `int loeschen(artikel* , int);`

   Übergabeparameter: Zeiger auf den Beginn des Datenfeldes, Anzahl der vorhandenen Datensätze
   Rückgabewert: Anzahl der Datensätze, um 1 niedriger als die Anzahl der übergebenen Datensätze, wenn der Datensatz gelöscht wurde.

   Die Funktion erwartet die Eingabe des Index eines Datensatzes (vgl. Menüpunkt '2'). Dieser Datensatz wird aus dem Datenfeld entfernt und von den folgenden Datensätzen überschrieben, so dass sich die Anzahl der Datensätze um 1 verringert.

Arbeitshinweise:

- Erstellen Sie in einem ersten Arbeitsschritt das komplette Hauptprogramm mit allen Funktionsdeklarationen und testen Sie das Hauptprogramm.

- Bilden Sie danach die Funktionen mit ihren Initialisierungen in der Reihenfolge der Menünummern.