Inhaltsangabe

Die Funktionen

- Die Wertübergabe und die Wertrückgabe bei Funktionen
- Lokale und globale Variablen

Arrays und Strings

- Arrays
- Die Elemente eines Arrays
- Strings
- Die Länge eines Strings ermitteln
- Das Vergleichen der Strings
- sprintf()
- sscanf()
- Einen String in einen anderen String kopieren - strcpy
- Einen String an einen anderen String anhängen - strcat
- Parsen
- In einem String nach einem String suchen - strstr
- In einem String nach einem Zeichen suchen - strrchr
- strtok
- Zusammenfassung zu den Stringfunktionen

Strukturen

- Matrix

Dateibezogene Ein- und Ausgaben

- Streams
- Dateistream öffnen
- Beschreiben einer Datei
- Eine Datei lesen
- Datei kopieren
- Öffnen der Exe-Dateien
- Öffnen einer Anwendung mit ShellExecute
- Öffnen und Schließen einer Anwendung mit CreateProcess und TerminateProcess
- Verschieben und Umbenennen einer Datei

- Löschen einer Datei
- Erstellen eines Ordners
- Verzeichnis wechseln
- Ordner löschen
- Verzeichnis auslesen

Headerdateien

- define
- Ein Programm mit Parametern öffnen
- Farbe für Fortgeschrittene(SetConsoleTextAttribute)
- Der Thread
- Die eigene Bibliothek
- Die Zeit
- Der Zufall

Audio

- Bei meinem Computer piepts wohl?!
- Lieder abspielen ohne sie zu öffnen
- Endlich Mp3
- Die Tonaufnahme
- Die Welt von Midi

Netzwerkprogrammierung

Maus, Tasten und Bildschirm

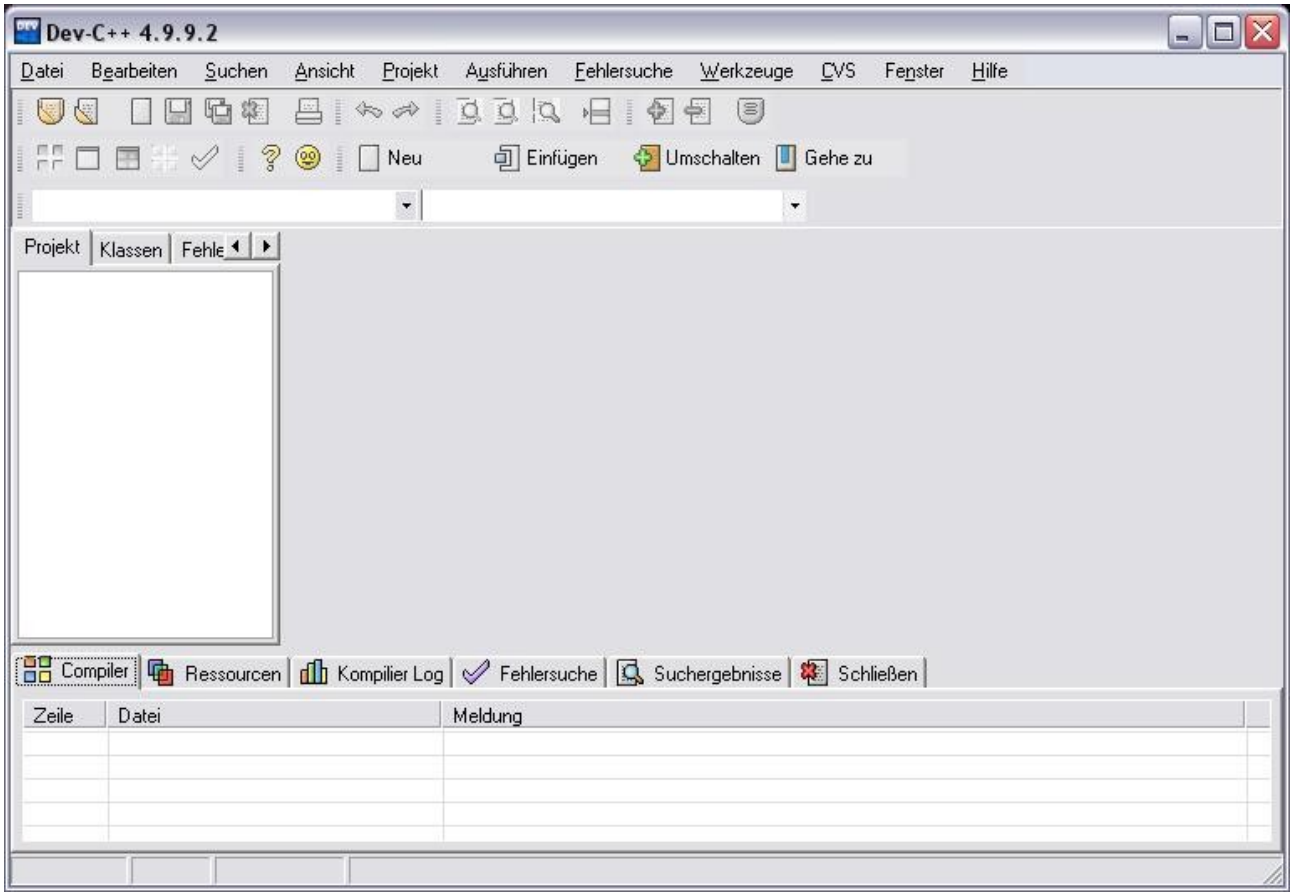
- Auflösung des Bildschirms ermitteln
- Die Maus bewegen
- Mausklick simulieren
- Wo ist die Maus?
- Tasten simulieren
- Vista und Dev-Cpp
- Der erste Schritt zur Fensterprogrammierung - MessageBox
- Literaturempfehlung

Einleitung

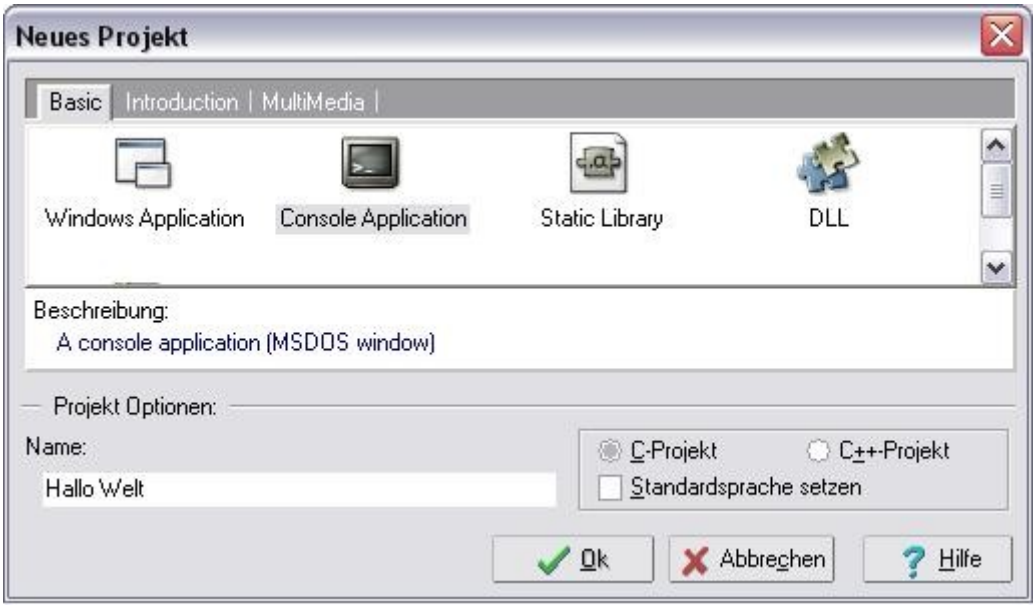
Ihr wollt C lernen? Das bedarf viel Arbeit, aber wenn man sich anstrengt, geht es ziemlich schnell. Manchmal schneller, als man glaubt. Erst einmal zu der Geschichte von C. Im Jahre 1972 entwickelte Dennis Ritchie die Programmiersprache C. Die Vorgänger dieser Sprache hießen B und keiner hätte es gedacht A. Anfangs setzte man C für das Betriebsystem Unix ein und als die ersten freien Compiler verbreitet wurden, wurde es immer bekannter und beliebter. Danach wurde C durch Bjarne Stroustrup objektorientiert gemacht. Die Erweiterung heißt also C++. Warum nicht D? Wirst du dich jetzt sicher fragen, oder? Das kommt daher, da es in C den Befehl ++ gibt. Dieser Befehl heißt nichts weiter, als +1. Es ist also ein Gag.

Dein erstes Programm

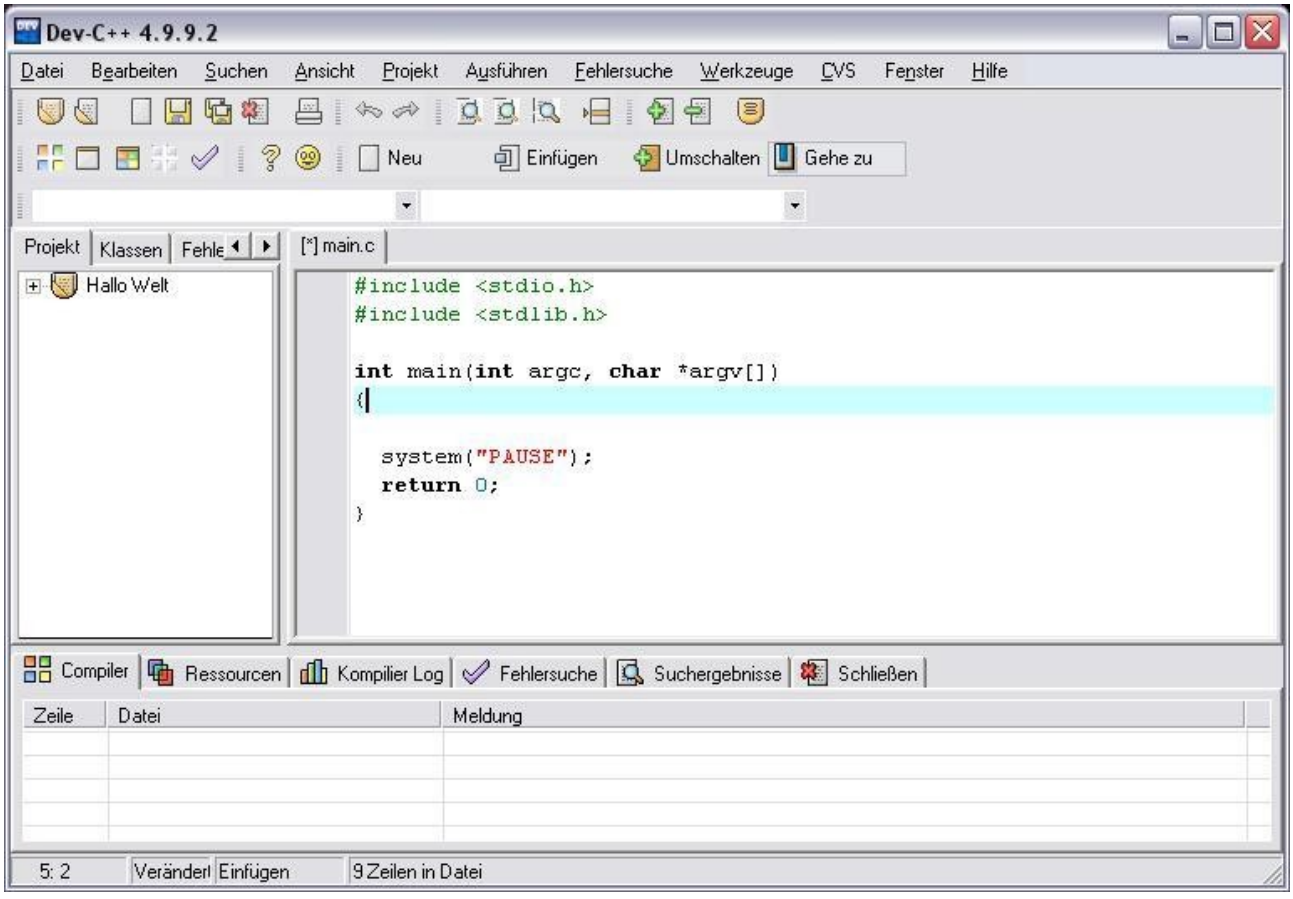
Wir wollen zuerst ein Programm schreiben, das "Hallo Welt!" ausgibt. Hierzu musst du als erstes den C-Compiler downloaden, wenn du dies noch nicht gemacht hast. Wenn du ihn gedownloadet hast, lese hier weiter. Nach der Installation siehst du folgendes Bild:



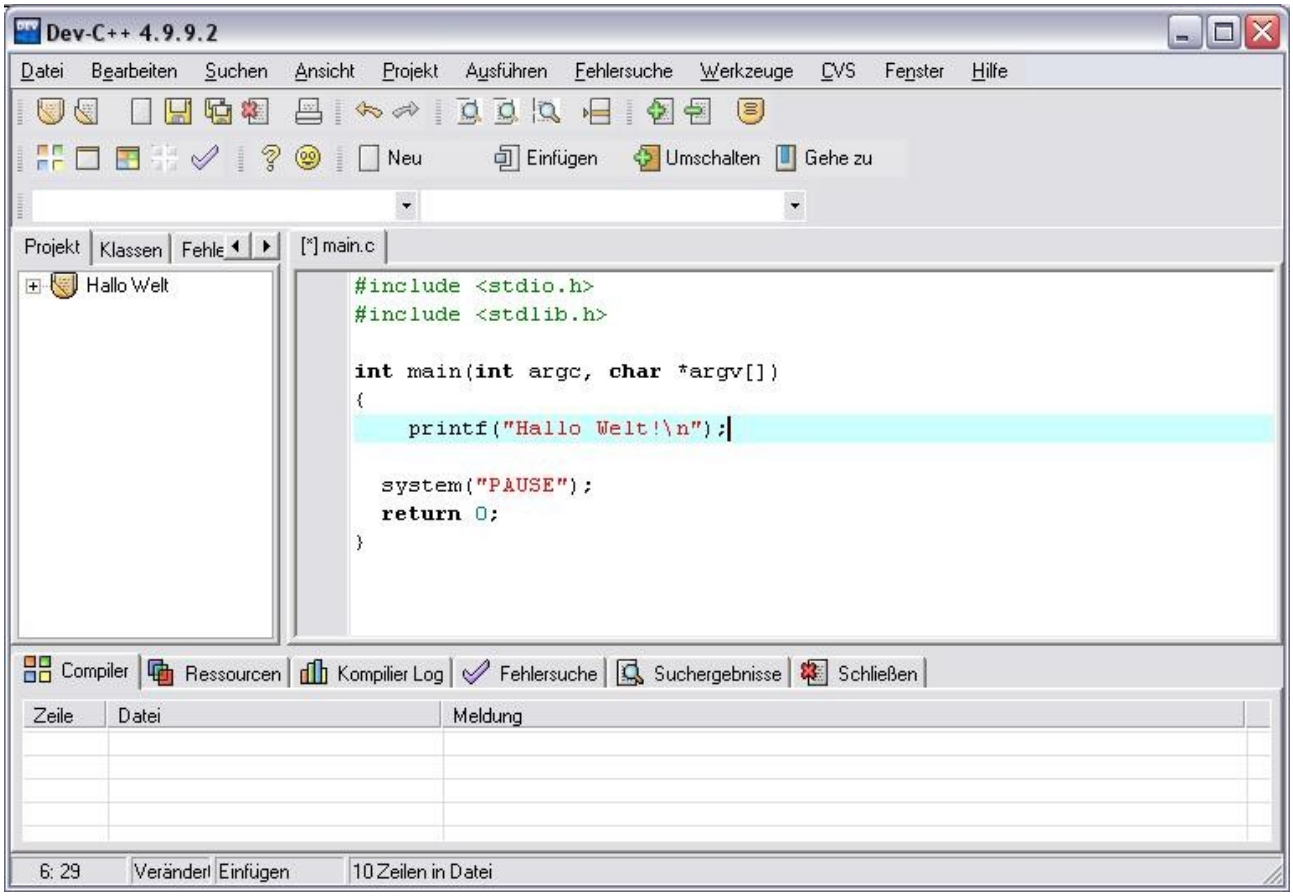
Klicke auf Datei, schließlich auf Neu, dann auf Projekt! Wenn dieses Fenster geöffnet wird, klicke auf Console Application und dass wir in C programmieren, also auf C-Projekt, und zum Schluss gibst du einen Namen deiner Wahl ein, mein Name für das Projekt ist z.B. Hallo Welt. Die angegebenen Aufgaben sind in diesem Fenster bereits ausgeführt:



Klicke anschließend auf OK und speichere das Projekt in einem Ordner deiner Wahl, aber vergiss das ".dev" hinter dem Projektnamen nicht! Schließlich solltest du dies auf deinem Bildschirm sehen:



Schreibe printf("Hallo Welt!\n"); in die Klammer, wie hier:



Zu den Befehlen : #include <stdio.h> und >#include <stdlib.h> sind Bibliotheken, die dem Compiler sagen, was beispielsweise printf bedeutet. Würdest du die beiden Bibliotheken oder auch Headerdateien weglassen, würde der Compiler einen Fehler bringen, da er printf nicht kennt. Zu dem nächsten Befehl: int main ist, wie man an dem Namen schon hört, die Hauptfunktion. Praktisch die Einleitung. Alles was in den beiden geschweiften Klammern steht, wird ausgeführt. Das was in den runden Klammern steht ist nur ein Zusatz, der eigentlich nicht benötigt wird. Man könnte ihn genauso gut weglassen. Der Befehl printf(" "); gibt das aus ,was in den runden Klammern, zwischen den Gänsefüßchen steht. Vergesse den Strichpunkt nicht, wenn du printf einsetzt. Wenn du den Text in printf() nicht mehr nur in eine Zeile schreiben willst, machst du das so:

```
printf("Text..."
"weiterer Text...");
```

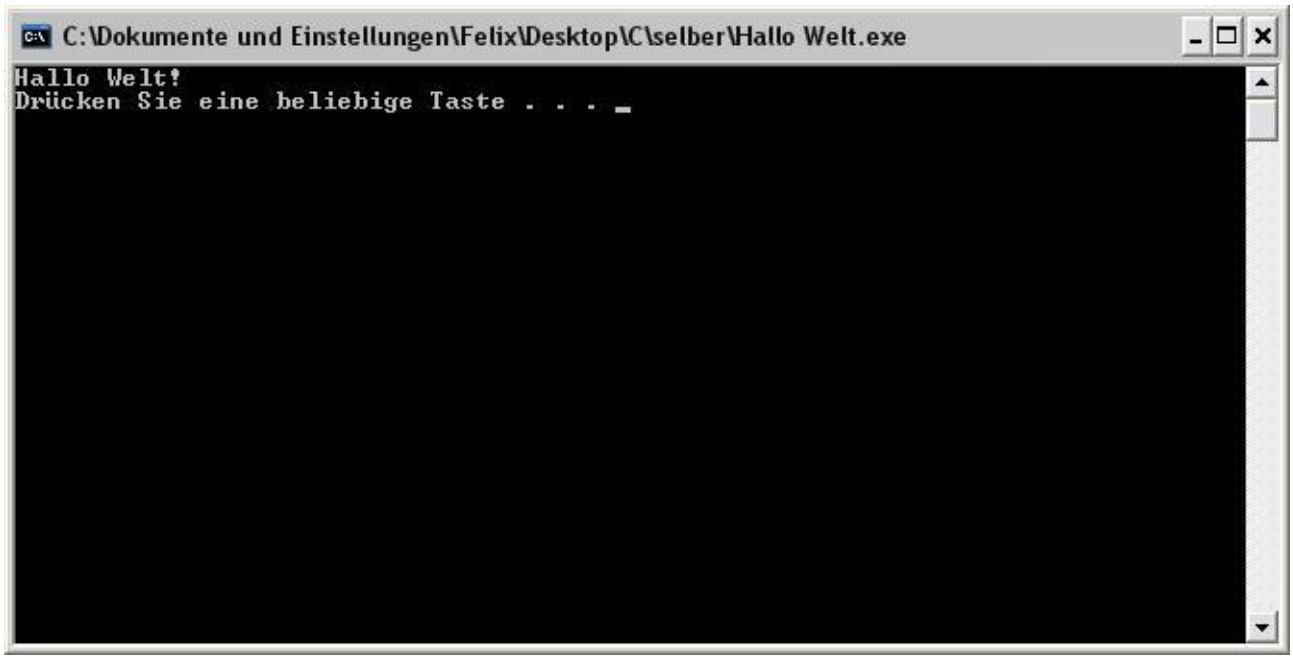
\n bewirkt einen Zeilenumbruch, falls du Html schreiben kannst, wäre das br in html. Es ist das Gleiche, wie wenn man auf Enter drückt. Ein häufig gemachter Fehler ist, dass man statt \n, /n schreibt. das wäre falsch. Also /n ist und bleibt falsch. Es gibt noch mehrere dieser Zeichen:

\a : warnender Ton
\b : Der Cursor wird um eine Position nach links versetzt.
\f : Seitenvorschub
\t : Cursor wird um 3 nach rechts verschoben

Jetzt weiter: system("Pause"); bewirkt, dass gesagt wird, drücken Sie eine beliebige Taste um zu beenden. Wenn system("Pause"); nicht stehen würde, würde das Programm sofort wieder beendet werden. Nämlich durch return 0; . Wenn du den Quelltext kommentieren willst, gibt es zwei Möglichkeiten, die ich dir an einem Beispiel zeigen möchte:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
printf("Hallo Welt!\n");
//Der Computer gibt " Hallo Welt! " aus und macht einen
Zeilenumbruch.
system("PAUSE");
return 0;/*beendet das
Programm*/
}
```

Es gibt die Möglichkeit über eine Zeile zu kommentieren, das wäre // und es gibt die Möglichkeit über mehrere Zeilen zu kommentieren, das wiederum wäre /*. Um den Kommentar zu beenden, schreibst du */ . Den Kommentar kann man im ausgeführten Programm natürlich nicht sehen. Um das Programm zu kompilieren klickst du oben auf Ausführen und danach auf Kompilieren. Wenn du kompiliert hast klicke noch einmal auf Ausführen, aber diesmal auf Ausführen. Nun fordert dich der Computer auf den Quelltext zu speichern. Vergesse .c nicht am Ende des Namen. Schließlich sollte folgendes Fenster erscheinen:



Das Programm verschönern

Wenn du die Schriftfarbe oder sonstiges ändern willst, musst du die exe Datei öffnen und schließlich mit der rechten Maustaste auf den Pfad der Datei klicken. Dort klickst du auf Eigenschaften. In diesem Fenster kannst du alles ändern, was du willst.

Das Programm mit Dos-Befehlen verschönern

Eine andere Methode die Schriftfarbe zu ändern ist der Dos-Befehl color. Ein Beispiel für diesen Befehl ist "color 0A". Der erste Parameter ist die Farbe des Hintergrundes der Dos-Anwendung. Der zweite Parameter hier A ist die Schriftfarbe. Jede Farbe hat ein Symbol. 0 ist z.B. die Farbe

schwarz. Hier die Symbole weiterer Farben:

0 = Schwarz
1 = Dunkelblau
2 = Dunkelgrün
3 = Blaugrün
4 = Dunkelrot
5 = Lila
6 = Ocker
7 = Hellgrau
8 = Dunkelgrau
9 = Blau
A = Grün
B = Zyan
C = Rot
D = Magenta
E = Gelb
F = Weiß

Jetzt ein kleines Beispielprogramm dazu:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
system("color 0A");
printf("matrix-Style\n");
system("PAUSE");

system("Cls");

system("color 0C");
printf("Error-Style\n");
system("PAUSE");

return 0;
}
```

Der Dos-Befehl Cls löscht die bisherigen Ausgaben der Anwendung.

Variablen

Ganzzahlen:

Ganzzahlen sind, wie du bestimmt schon weißt, 1,2,3,4, ... aber nicht 1/3 oder 1,42.
Es gibt folgende Typen von Ganzzahlen:

short ist im Wertebereich von -32768 bis +32767
int ist im Wertebereich von -2147483648 bis +2147483647
long ist im Wertebereich von -2147483648 bis +2147483647

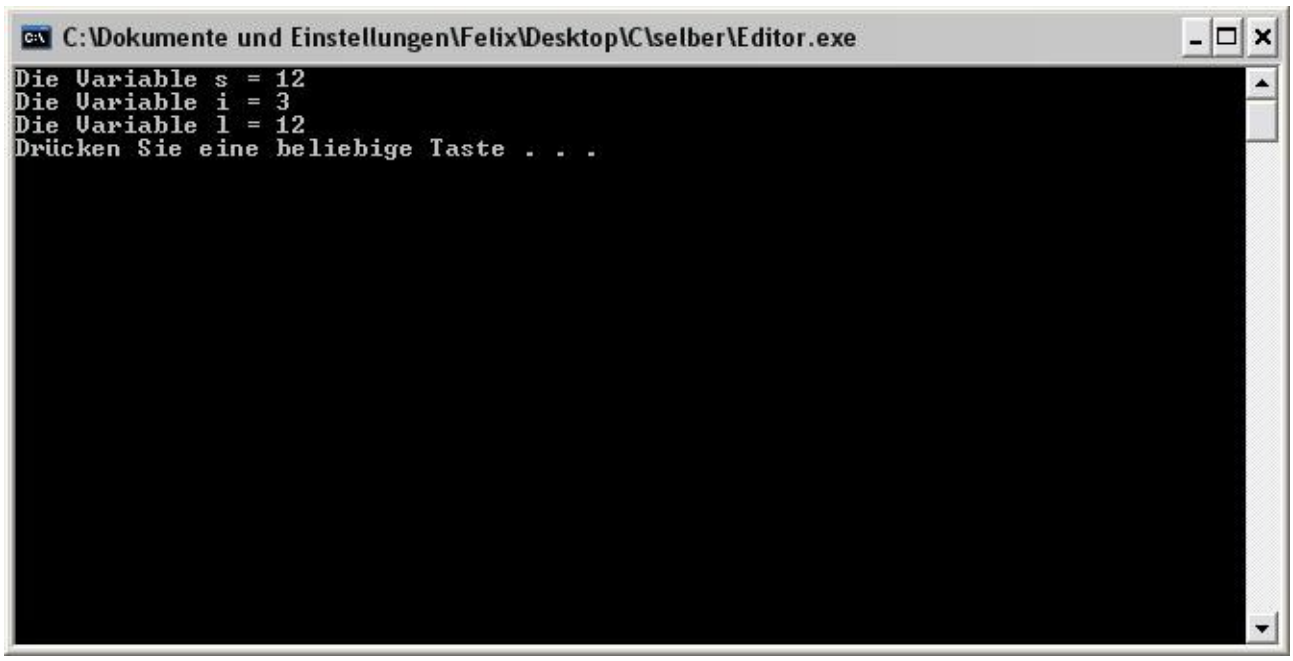
Um eine Variable in einen Quelltext einzufügen, muss man dieses beachten. Das erste Zeichen eines Variablennamens darf keine Zahl sein. Ebenfalls darf man keine Sonderzeichen, außer "_", Leerzeichen oder Umlaute einsetzen. Jetzt ein Beispiel, wie man einen Wert einer Variable einer anderen übergibt:(Wenn du nicht mehr weißt, wie man ein Projekt erstellt klicke hier.)

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
short s; //short ist der Typ und s ist der Name der Variablen
int i;
long l;
l=12; //wenn man einer Variable eine Wert gibt, muss man dies
immer mit einem ; beenden
i=3;
s=l; //l ist 12, also ist s jetzt auch 12

system("PAUSE");
return 0;
}
```

Bei dem Ausgeben einer Variable kommt ein weiteres Problem auf dich zu, wie du gleich siehst. Das Programm soll die Variablen von dem letzten Programm ausgeben:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
short s; //short ist der Typ und s ist der Name der Variablen
int i;
long l;
l=12; //wenn man einer Variable eine Wert gibt, muss man dies
immer mit einem ; beenden
i=3;
s=l; //l ist 12, also ist s jetzt auch 12

printf("Die Variable s = %d \n", s);
printf("Die Variable i = %d \n", i);
printf("Die Variable l = %ld \n", l);

system("PAUSE");
return 0;
}
```

Das %d steht übersetzt für so etwa dezimale Ganzzahl und %ld für lange dezimale Ganzzahl. So kann man es sich gut merken. Nach dem Komma steht die Variable, die ausgegeben werden soll. Das Ergebnis sollte so aussehen:



Fließkommazahlen:

Natürlich gibt es nicht nur Ganzzahlen, sondern auch Fließkommazahlen, das ist z.B. 1,42. Hier gibt es diese Typen:

float ist im Wertebereich von $+/-3.4*10^{-38}$ bis $+/-3.4*10^{38}$
double ist im Wertebereich von $+/-1.7*10^{-308}$ bis $+/-1.7*10^{308}$
long double ist im Wertebereich von $+/-3.4*10^{-4932}$ bis $+/-3.4*10^{4932}$

Ein häufig gemachter Fehler ist, dass man bei Zahlen 1,23 oder 3,4 Fehler macht. Im Englischen steht statt einem Komma ein Punkt. So heißen die Zahlen schließlich 1.23 und 3.4 . Dazu ein Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
float zahl1 = 123.456;
double zahl2 = 1234.5678;

printf("Der Wert von der Variable zahl1: %f\n",zahl1);
printf("Der Wert von der Variable zahl2: %lf\n",zahl2);

system("PAUSE");
return 0;
}
```

Rechnen:

Mit Variablen zu rechnen ist ziemlich leicht. Es gibt folgende Rechenzeichen:

= hat die Bedeutung der Zuweisung. Beispiel: Wert=9;/*Wert = 9*/
+ hat die Bedeutung der Addition. Beispiel: Wert=9+9;/*Wert = 18*/
- hat die Bedeutung der Subtraktion. Beispiel: Wert=9-3;/*Wert = 6*/
/ hat die Bedeutung der Division. Beispiel: Wert=9/3;/*Wert = 3*/
* hat die Bedeutung der Multiplikation. Beispiel: Wert=9*2;/*Wert = 18*/
% hat die Bedeutung der Modulo(Rest der Division). Beispiel: Wert=9/2;/*Wert = 1*/
(9/2=4 Rest 1, der Rest wird ausgegeben)

Jetzt ein Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
float i = 10.4;
int r = 3;
float Ergebnis;
Ergebnis=(i-r)*2; //heißt also (10.4-3)*2

printf("Das Ergebnis ist %f\n",Ergebnis);

system("PAUSE");
return 0;
}
```

Wenn man eine Ausgabe in einer bestimmten Form erhalten will, kann man dies so machen:
Schreibe die folgenden Zeilen unter printf("Das Ergebnis ist %f\n",Ergebnis); einfach weiter!

printf("Das Ergebnis im Exponentialformat ist %e\n",Ergebnis);
printf("Das Ergebnis in der kompakten Art ist %g\n",Ergebnis);

Wenn du Zahlen konvertieren willst. z.B. eine Ganzzahl zu einer Hexadezimalzahl geht dies so:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
int Zahl = 100;
printf("Dezimale Darstellung : %d \n",Zahl);
printf("Hexadezimale Darstellung : %x \n",Zahl);
printf("Oktale Darstellung : %o \n",Zahl);
printf("ASCII-Darstellung : %c \n",Zahl);

system("PAUSE");
return 0;
}
```

Man kann auch die Länge der Zahlen bestimmen. z.B. es sollen nur 4 Stellen nach dem Komma ausgegeben werden oder du willst, dass eine bestimmte Ausgabe rechtsbündig ist. Hier ein Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
int Zahl;
float Zahl2;

Zahl=12345678;
Zahl2=1.234;

printf("%14d \n",Zahl);
printf("%014d \n",Zahl);
printf("%-14d \n",Zahl);

printf("%.2f \n",Zahl2);

system("PAUSE");
return 0;
}
```

Rechnenoperatoren-Spezial(+=; -=; *=; /=)

Wenn man zwei Programme miteinander vergleicht, schaut man auch oft auf die Länge des Programms. Eine Möglichkeit, wie man sein Programm kürzer machen kann, siehst du hier:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
int a=0;
int b=0;
int c=0;

b=b+1;
a=a+1;
a=a+b;
c=a;
c=c-a;
a=a+10;
a=a*b;
c=c/b;

printf("a=%d,b=%d,c=%d\n\n",a,b,c);

system("PAUSE");
return 0;
}
```

Das ist jetzt das Programm in der langen Form. Jetzt kommt die Kurzform:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
int a,b,c;
a=b=c=0;

b++; //b=b+1
a++; //a=a+1
a+=b; //a=a+b
c=a;
c-=a; //c=c-a
a+=10; //a=a+10
a*=b; //a=a*b
c/=b; //c=c/b

printf("a=%d,b=%d,c=%d\n\n",a,b,c);

system("PAUSE");
return 0;
}
```

Unsigned

Es gibt noch einen Zusatz für die Variablen, nämlich unsigned. Dieser Zusatz erhöht den Wertebereich auf der positiven Seite, aber verkleinert ihn auf der negativen Seite. Das sieht so aus:

unsigned short hat den Wertebereich von 0 bis 65535
unsigned int hat den Wertebereich von 0 bis 4294967295
unsigned long hat den Wertebereich von 0 bis 4294967295

Bei den Fließkommazahlen gibt es diesen Zusatz nicht.

Char:

Es gibt noch einen letzten Typ, der char heißt. Man setzt ihn meistens ein, wenn man mit Buchstaben arbeitet. Dazu ein Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
char z = 'C';

printf("Wert : %d ->(ASCII-codiert)->%c\n",z,z);

system("PAUSE");
return 0;
}
```

Hier kommen gleich 2 häufig gemachte Fehler auf dich zu. Es ist sehr wichtig, dass man einen

Buchstaben immer in ' ' setzt, wenn man sie in Gänsefüßchen setzt, ist das falsch.

Umlaute

Wenn du in C Umlaute schreibst, kommen meistens irgendwelche komischen Zeichen heraus, aber nicht die Umlaute.
So kommen die Umlaute:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{

printf("Die Umlaute: \x81 \x84 \x94 \x8E \x99 \x9A \xE1");//ü ä ö
Ä Ö Ü ß
printf("\n\nJetzt in einem zusammenh\x84ngenden Text:\n\n");
printf("An einem sch\x94nen Fr\x81hlingstag bl\x81hten die
Blumen.\n\n");

system("PAUSE");
return 0;
}
```

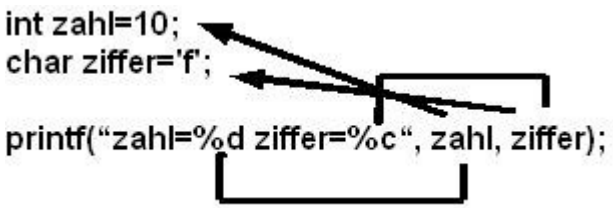
Du hast ja schon mal von der hexadezimalen Darstellung gehört. Hier werden die Umlaute einfach hexadezimal ausgegeben. Umlaute Hexadezimal:

ä : \x84 (Man sagt machmal auch einfach nur 84! \x84 ist nur die Schreibweise für hexadezimale Zahlen in C.)
ö : \x94
ü : \x81
Ä : \x8E
Ö : \x99
Ü : \x9A
ß : \xE1

Alle hexadezimalen Werte der Zahlen oder der Buchstaben und noch vieles mehr, findet ihr auf www.vias.org/mikroelektronik/ASCII.html.

Das Ausgeben mehrerer Variablen

In diesem Beispiel kann man da zwar nichts falsch machen, aber wenn zwei unterschiedliche Variablen ausgegeben werden müssen, kommt man leicht durcheinander. Hier ist ein Bild, das zeigt, wie man vorgeht:



Hier wäre es nicht so sinnvoll, dass die Zahl bei der Ziffer steht und anders herum.

Eingabeaufforderung

Wenn du jemand auffordern willst eine Zahl oder einen Buchstaben einzugeben, musst du ebenfalls mit Variablen arbeiten. Ein Beispiel:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
int Jahre;
printf("Gebe bitte ein, wie alt du bist : ");
scanf("%d",&Jahre);
fflush(stdin);
printf("\nDu bist %d Jahre alt.\n",Jahre);

system("PAUSE");
return 0;
}
```

Erst einmal zu der Funktion scanf(). In " " muss der Formatbezeichner stehen, der bei int Jahre %d ist. Nach dem Komma steht das & mit dem Namen der Variable. Vergesse das & nicht, sonst bringt der Computer an dieser Stelle eine Fehlermeldung und beendet das Programm. Das ist ebenfalls ein sehr häufig gemachter Fehler. Am Schluss der Funktion steht ein Strichpunkt. fflush(stdin); ist in diesem Programm nicht nötig, wird aber später nötig sein. Die Funktion ist dazu da im Puffer der Standarteingabe (stdin) die Zeichen zu löschen. Das ist dann wichtig, wenn man öfters scanf() einsetzt.
Wenn du Probleme hast kann dies an fflush(stdin); liegen. Setze für fflush(stdin); , getchar(); ein, das könnte das Problem lösen, da manche Compiler mit fflush(stdin) nicht zurechtkommen.

Die unsichtbare Eingabeaufforderung - getch

Um eine unsichtbare Eingabeaufforderung zu erklären, muss man zuerst einmal wissen, was eine sichtbare Eingabeaufforderung ist. Die Funktion für die sichtbare Eingabe kennst du bereits. Sie heißt scanf(). Wenn mit dieser zur Eingabe aufgefordert wird, sieht man seine Eingabe. Man kann diese verbessern oder verändern und muss sie mit Enter bestätigen. Da getch das Gegenteil von scanf ist, ist klar, dass man bei dieser Eingabeaufforderung seine Eingabe weder sieht, noch diese ändern oder bestätigen könnte. Dies hat einige Vorteile. Wenn man z.B. ein Spiel mit C programmieren möchte, würde es nerven, wenn man jeden Schritt mit Enter bestätigen müsste oder bei meinem Klavierprogramm würde es den Benutzer sicher auch störren, wenn dieser immer bestätigen müsste, welchen Ton er gerade spielen will. In diesem Vergleich Programm unten kannst du sehen, wie dieser Unterschied zwischen scanf und getch in der Praxis zur Geltung kommt:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
char Eingabe;
```

```
printf("scanf() Eingabeaufforderung:\n");
scanf("%c",&Eingabe);
getchar();
printf("Deine Eingabe wurde auf dem Bildschirm ausgegeben und \ndu
musstest Enter druecken. Deine Eingabe war: %c\n\n",Eingabe);

printf("getch() Eingabeaufforderung:\n");
Eingabe=getch();
printf("Deine Eingabe wurde nicht auf dem Bildschirm ausgegeben
und \ndu musstest nicht Enter druecken. Deine Eingabe war:
%c\n\n",Eingabe);

system("PAUSE");
return 0;
}
```

Kontrollstrukturen

Es gibt folgende Kontrollstrukturen:

Verzweigungen:
Hierbei wird im Programm eine Bedingung definiert. Je nachdem, ob diese wahr oder nicht wahr ist, wird das Programm an unterschiedlichen Stellen fortgesetzt.

Schleifen:
Sie sind dazu da, Anweisungen zu wiederholen.

Sprünge:
Die Programmausführung wird unterbrochen und an einer bestimmten Stelle, die markiert wurde, fortgesetzt.

Die if-Verzweigung

```
if(Bedingung)
{
Anweisung(en);
}
```

Erklärung: wenn(if) die Bedingung(es können auch mehrere sein)in den runden Klammern eintrifft, werden die Anweisungen in den geschweiften Klammern ausgeführt, wenn die Bedingung(en) nicht eintrifft, wird die Verzweigung übersprungen. Jetzt ein Beispiel, ähnlich dem Letzten:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
int Alter;

printf("Gebe ein, wie alt du bist: ");
scanf("%d",&Alter);
fflush(stdin);
```

```
printf("\n");
if(Alter<14)
{
printf("Du bist ein Kind\n");
}
if(Alter>=14 && Alter<18)/*Hier sind es zwei Bedingungen. Die
beiden "&" bedeuten "und", also müssen beide Bedingungen erfüllt
sein. Es gibt auch "oder". Das schreibt man "||" .*/
{
printf("Du bist ein Teenager\n");
}
if(Alter>=18)
{
printf("Du bist erwachsen\n");
}

system("PAUSE");
return 0;
}
```

Vergleichsoperatoren:

Folgende gibt es:

a < b : ist wahr, wenn a kleiner b ist
a > b : ist wahr, wenn a größer b ist
a <= b: ist wahr, wenn a kleiner oder gleich b ist
a >= b: ist wahr, wenn a größer oder gleich b ist
a == b: ist wahr, wenn a gleich b ist
a != b: ist wahr, wenn a ungleich b ist

Die else-Verzweigung

```
if(Bedingung)
{
Anweisung(en);
}
else
{
Anweisung(en);
}
```

Erklärung: Zunächst kommt genau dasselbe. Aber dann kommt else, was so viel wie sonst, andernfalls heißt. Wenn die if-Bedingung(en) unwahr sind, werden automatisch die Anweisung(en) in den geschweiften Klammern bei else ausgeführt. Das vorherige Beispiel wird jetzt ein wenig verfeinert:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
int Alter;
```

```
printf("Gebe ein, wie alt du bist: ");
scanf("%d",&Alter);
fflush(stdin);
printf("\n");
if(Alter<14)
{
printf("Du bist ein Kind\n");
}
else
{
if(Alter>=14 && Alter<18)/*Hier sind es zwei Bedingungen. Die
beiden "&" bedeuten "und", also müssen beide Bedingungen erfüllt
sein. Es gibt auch "oder". Das schreibt man "||" .*/
{
printf("Du bist ein Teenager\n");
}
else
{
printf("Du bist erwachsen\n");
}
}

system("PAUSE");
return 0;
}
```

Die else if-Verzweigung

Sie sieht ganz ähnlich aus:

```
if(Bedingung)
{
Anweisung(en);
}
else if(Bedingung)
{
Anweisung(en);
}
else
{
Anweisung(en);
}
```

Erklärung: Zunächst, wie immer dasselbe, schließlich mit dem else if das Neue: wenn die if Bedingung unwahr ist, wird die else if Bedingung ausgeführt und wenn die wiederum falsch ist, werden die Anweisungen bei else ausgeführt. Schließlich sieht das Programm so aus:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
int Alter;
```

```
printf("Gebe ein, wie alt du bist: ");
scanf("%d",&Alter);
fflush(stdin);
printf("\n");
if(Alter<14)
{
printf("Du bist ein Kind\n");
}
else if(Alter>=14 && Alter<18)/*Hier sind es zwei Bedingungen. Die
beiden "&" bedeuten "und", also müssen beide Bedingungen erfüllt
sein. Es gibt auch "oder". Das schreibt man "||" .*/
{
printf("Du bist ein Teenager\n");
}
else
{
printf("Du bist erwachsen\n");
}

system("PAUSE");
return 0;
}
```

Die switch-Verzweigung

Diesmal sieht die Verzweigung ganz anders aus, bewirkt aber ungefähr das Gleiche, wenn du schon programmieren kennst du den switch vielleicht als CASE :

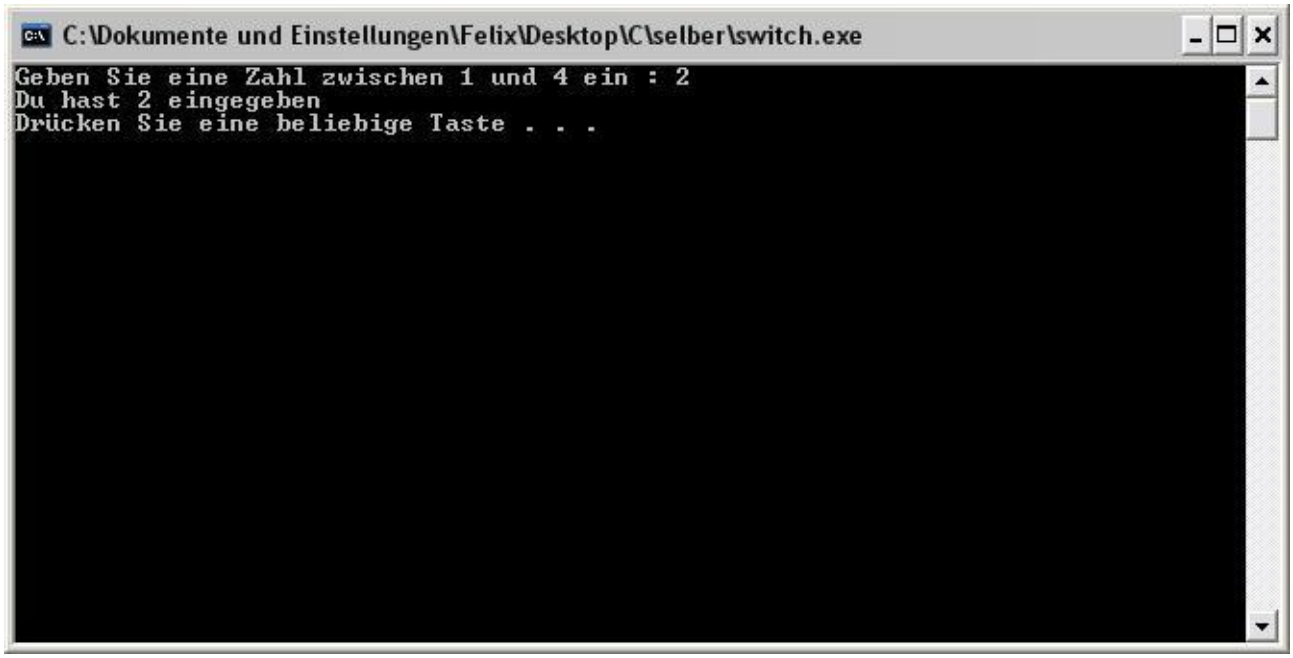
```
switch(Ausdruck)
{
case Ausdruck1 : Anweisung(en);
break;
case Ausdruck2 : Anweisung(en);
break;
case Ausdruck3 : Anweisung(en);
break;
case Ausdruck4 : Anweisung(en);
break;
case Ausdruck5 : Anweisung(en);
break;
case Ausdruck6 : Anweisung(en);
break;
...
default : Anweisung(en);
}
```

switch ist der Beginn der Verzweigung. In der Klammer steht der Name der Variable, um die es geht. Weiter geht es mit case. case heißt so viel, wie Fall. case 1 : Anweisung(en); heißt z.B. wenn du 1 eingibst, führt der Computer die Anweisung(en) bei case 1 aus. In einem Beispiel kannst du dies näher betrachten. Aber jetzt noch zu break. Mit break wird die Verzweigung beendet und der Computer macht mit den Anweisung(en) nach der Verzweigung weiter. default gleicht else : alles, was nicht mit case angegeben worden ist, ist falsch, somit werden die Anweisungen bei default ausgeführt. Wenn z.B. case 1 und 2 angegeben wurden und du 3 eingibst, werden die Anweisungen

bei default ausgeführt. Ein Beispiel dazu:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
int Eingabe;

printf("Geben Sie eine Zahl zwischen 1 und 4 ein : ");
scanf("%d",&Eingabe);
fflush(stdin);

switch(Eingabe)
{
case 1 : printf("Du hast 1 eingegeben\n");
break;
case 2 : printf("Du hast 2 eingegeben\n");
break;
case 3 : printf("Du hast 3 eingegeben\n");
break;
case 4 : printf("Du hast 4 eingegeben\n");
break;
default: printf("Unbekannte Eingabe?!\n");
}

system("PAUSE");
return 0;
}
```

Wenn dieses Fenster erscheint hast du richtig abgetippt.



Die while-Schleife

```
while(Bedingung)
{
Anweisung(en);
}
```

Erklärung: Die while-Schleife ist dazu da, Anweisungen zu wiederholen. Solange die Bedingung wahr ist, werden die Anweisungen in den geschweiften Klammern wiederholt. Wenn die Bedingung nicht erfüllt ist, wird Schleife sofort übersprungen. Du erinnerst dich bestimmt noch gut an das Alter-Beispiel. Dieses wird noch ein letztes Mal verbessert:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
int Alter;
int Ende;
while (Ende!=2){
printf("Druecke fuer die Ausfuehrung die folgenden
Tasten:\n<1>Beginn des Programms\n<2>Beendung des Programms\n");//
Diese Zeile darf keinen Zeilenumbruch haben wie hier!!
scanf("%d",&Ende);
fflush(stdin);
printf("Deine Wahl war <%d>.\n\n",Ende);
switch(Ende)
{case 1:
printf("Gebe ein, wie alt du bist: ");
scanf("%d",&Alter);
fflush(stdin);
printf("\n");
if(Alter<14)
{
printf("Du bist ein Kind\n\n");
}
if(Alter>=14 && Alter<18)
{
printf("Du bist ein Teenager\n\n");
}
else
{
printf("Du bist erwachsen\n\n");
}
case 2:break;
}
}

system("PAUSE");
return 0;
}
```

Ja jetzt können wir endlich das fertige Programm betrachten:

Wenn man die Anweisungen so und so oft wiederholen will, kann man dies so machen:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
int zahl=0;

while(zahl<5)
{
zahl++;
printf("Es wurde das %d.Mal wiederholt.\n",zahl); //diese Zeile
wird 5 Mal wiederholt
}

system("PAUSE");
return 0;
}
```

Die do while-Schleife

```
do
{
Anweisung(en);
}while(Bedingung);
```

Zuerst werden die Anweisung(en) in den geschweiften Klammern bei do ausgeführt, das geschieht immer. Wenn die Bedingung(en) bei while wahr sind, werden die Anweisung(en) innerhalb der Do Klammern wiederholt. Ein häufig gemachter Fehler ist, nach der Bedingung den Strichpunkt zu vergessen. Achte darauf! Jetzt wieder ein Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
int Eingabe;

do{
printf("Ich denke mir eine Zahl zwischen 1 und 4. Welche ist
es?\n");
printf("Eingabe: ");
scanf("%d",&Eingabe);
fflush(stdin);
}while(Eingabe!=4);
printf("Richtig!\n");

system("PAUSE");
return 0;
}
```

Natürlich ist dieses Spiel langweilig, da du weißt, welche Zahl du einsetzen musst. Aber zeig es doch deinen Freunden, die kennen den Quelltext, ja noch nicht. Man könnte für eine do while-Schleife eigentlich auch Folgendes schreiben:

```
Anweisung(en);
while(Bedingung)
{
Anweisung(en);
}
```

Da die Anweisung(en) dann aber zwei Mal geschrieben werden müssten, nimmt man eine do while-Schleife.

Die for-Schleife

```
for(Initialisierung; Bedingung; Reininitialisierung)
{
Anweisung(en);
}
```

Initialisierung bedeutet, einer Variable einen Wert zu geben. z.B. zahl=1; . Bedingung sollte klar sein. Reininitialisierung ist z.B. zahl++, dass bei jeder Wiederholung die Variable verändert(vergrößert,verkleinert) wird. In den geschweiften Klammern stehen wie immer die Anweisungen. Du hast vorhin ein Beispiel gesehen, in dem vorkam, wie man es macht, dass man mit while so und so oft die Anweisung(en) wiederholt. Jetzt kommt dasselbe Beispiel mit for:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
int zahl;
```

```
for(zahl=1; zahl<=5; zahl++)
{
printf("Es wurde das %d.Mal wiederholt.\n",zahl);
}

system("PAUSE");
return 0;
}
```

Ein paar Unterschiede gibt es schon, z.B. muss hier die Variable nicht auf 0 gesetzt werden, sondern auf 1 und die Bedingung muss ebenfalls anders heißen. Hier noch ein anderes Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
int f;

for(f=1; f<=10; f++)
{
printf("for(f=1; %d<=10; %d++)\n",f,f);
}

system("PAUSE");
return 0;
}
```

Wenn du die Initialisierung oder etwas anderes weglassen willst, schreibt man das so:

Wenn man die Initialisierung weglassen will: for(; Bedingung; Reininitialisierung)...
Wenn man die Bedingung weglassen will: for(Initialisierung; ;Reininitialisierung)...
Wenn man die Reininitialisierung weglassen will: for(Initialisierung; ;)...
Wenn man alles weglassen will: for(; ;)...

Schleifen abbrechen:

Es gibt 4 Möglichkeiten eine Schleife abbzubrechen:

break : Beendet die ganze Schleife.
continue : Bricht den aktuellen Schleifendurchlauf ab und die Schleife beginnt von vorn.
return : Beendet die komplette Funktion.
exit : Beendet das Programm.

Die Funktionen

Mit Funktionen kann man Teilprobleme auslagern. Die Lösung für eine dieser Probleme wird dann als Funktion verarbeitet. Es ist nicht sehr sinnvoll eine Funktion zu schreiben, wenn dieses Problem nur einmal im Programm vorkommt, wenn du allerdings an die Übersicht denkst ist es sinnvoll. Du kannst auch mehrere Funktionen zu einer Bibliothek zusammenfassen. Hier das Grundgerüst einer Funktion:

```
Rückgabetyp Funktionsname(Parameter)
{
Anweisung(en);
}
```

Erklärung:
Bei dem Rückgabetyp wird der Datentyp festgelegt. z.B. int, float... Wenn die Funktion keinen Wert zurückgeben soll gebe void als Rückgabetyp an.
Den Funktionsname kannst du nach den gleichen Regeln, wie bei den Variablen bestimmen. Nehme aber nicht scanf() oder andere Funktionen, die es bereits gibt.
Parameter sind Werte, die du einer Funktion als Argumente übergeben kannst. Jetzt ein kleines Beispiel:

```
#include <stdio.h>
#include <stdlib.h>

void Hallo()
{
printf("Hallo\n");
}

int main(int argc, char *argv[])
{
Hallo();//hier kommt die Funktion
printf("Willst du mich nicht begruessen, nur weil ich ein Programm
bin? Also wirklich!\n Wirds' bald?!\n"); /*hier sind wir zurück in
der main Funktion*/
system("PAUSE");
printf("Na endlich. Geht doch. Ich bin jetzt auch freundlich:\n");
Hallo(); //hier kommt die Funktion wieder
printf("Und tschuess\n"); // und das letzte Mal in der main
Funktion

system("PAUSE");
return 0;
}
```

Die Wertübergabe und die Wertrückgabe bei Funktionen:

Zuerst ein Beispiel danach die Erklärung:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double Wurzel(double d)
{
double Wurzel1;
Wurzel1= sqrt(d);

return(Wurzel1);
}
```

```
int main(int argc, char *argv[])
{
float Eingabe;
double Ergebnis;

printf("Gebe die Zahl ein, von der du die Wurzel wissen willst:
");
scanf("%f",&Eingabe);

Ergebnis=Wurzel(Eingabe);

printf("Die Wurzel von %G ist %G\n",Eingabe, Ergebnis);

system("PAUSE");
return 0;
}
```

Erklärung: Zuerst einmal zu der neuen Bibliothek math.h . Sie beinhaltet verschiedene mathematische Funktion. In dieser Einführung über C wirst du nur die Wichtigsten kennen lernen. Wenn dich diese Funktionen interessieren klicke hier.Schließlich zur eigentlichen Wertübergabe. Am Anfang wird durch scanf("%f",&Eingabe); die Zahl ermittelt, die man für die Funktion benötigt. Danach wird die Variable Eingabe zur Variable d. Das geschieht, weil Eingabe in der Klammer von der Funktion Wurzel steht und d in der Klammer steht. Jetzt wird die Wurzel berechnet und danach wird der Wert von der Variable Wurzel1 an die main Funktion übergeben. Das nennt man die Wertrückgabe.

Lokale und globale Variablen

Wenn du eine globale Variable deklarierst, kannst du mit dieser Variable in allen Funktionen arbeiten. Man erkennt eine globale Variable daran, dass sie zwischen den Bibliotheken und der ersten Funktion steht. Lokale Variablen gelten nur für die Funktionen, in denen die Variablen stehen. Eine lokale Variable ist hier Eingabe(in der main-Funktion).

Arrays und Strings

Arrays

Wenn man ein Array benutzt, multipliziert man praktisch ein Variable. Ein Array schaut so aus:

```
int zahl[5];
```

im Grunde ist das hier das Gleiche:

```
int zahl0;
int zahl1;
int zahl2;
int zahl3;
int zahl4;
```

Du fragst dich bestimmt, was die Variable zahl0 da zu suchen hat. Bei Arrays fängt man immer

zuerst mit 0 an aber dafür hört man eine Zahl früher auf. Also, wenn es heißt Variable[4], dann lautet die Reihenfolge 0.Element, 1.Element, 2.Element, 3.Element. So sieht das Grundgerüst aus:

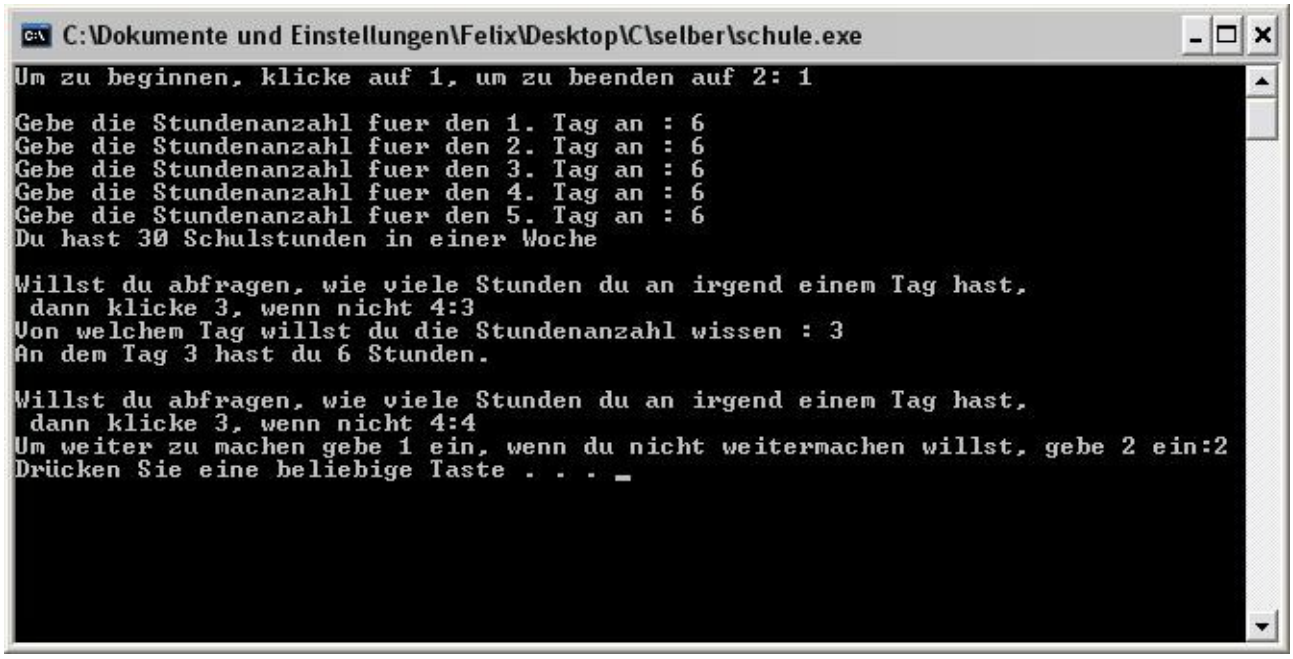
Datentyp Arrayname[Anzahl_der_Elemente];

Jetzt ein Beispiel dazu über Schulstunden *seufz*, na ja es muss sein:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
int Tage[5]; //hier der Array
int d;
int Summe=0;
int Tag;
int Ende;

printf("Um zu beginnen, klicke auf 1, um zu beenden auf 2: ");
scanf("%d",&Ende);
fflush(stdin);
printf("\n");
while(Ende!=2)
{
switch(Ende)
{
case 1 :
for(d=0; d<5; d++)
{
printf("Gebe die Stundenanzahl fuer den %d. Tag an : ",d+1);//da d
Null ist, würde die Schleife mit Tag 0 beginnen
scanf("%d",&Tage[d]);
fflush(stdin);
Summe=Summe+Tage[d];
}
printf("Du hast %d Schulstunden in einer Woche\n\n",Summe);
Summe=0;

printf("Willst du abfragen, wie viele Stunden du an irgend einem
Tag hast,\n dann klicke 3, wenn nicht 4:");
scanf("%d",&Tag);
fflush(stdin);

if(Tag == 3)
{
printf("Von welchem Tag willst du die Stundenanzahl wissen : ");
scanf("%d",&d);
fflush(stdin);
printf("An dem Tag %d hast du %d Stunden.\n\n", d, Tage[d-1]);//
da oben addiert worden ist, muss das hier rückgängig gemacht
werden
printf("Willst du abfragen, wie viele Stunden du an irgend einem
Tag hast,\n dann klicke 3, wenn nicht 4:");
```

```
scanf("%d",&Tag);
fflush(stdin);
}
if(Tag == 4)
{
printf("Um weiter zu machen gebe 1 ein, wenn du nicht weitermachen
willst, gebe 2 ein:");
scanf("%d",&Ende);
fflush(stdin);
}
break;
case 2 :break;
}
}
system("PAUSE");
return 0;
}
```

Hier das Ergebnis:



Die Elemente eines Arrays

Die Elemente eines Arrays sind die einzelnen Variablen, die ganz am Anfang als Beispiel für Arrays gezeigt worden sind. Man schreibt sie aber anders:

statt int zahl0; schreibt man int zahl[0];
statt int zahl1; schreibt man int zahl[1];
statt int zahl2; schreibt man int zahl[2];
statt int zahl3; schreibt man int zahl[3];
statt int zahl4; schreibt man int zahl[4];

Diese Variablen kannst du benutzen, wenn du einer dieser Elemente alleine z.B. ausgeben willst.

Für das erste Element würde das so aussehen:

```
printf("%d",zahl[0]);
```

Strings

Eigentlich kennst du Strings schon. Sie begegnen dir fast jedes mal, wenn du printf() benutzt. Wenn du z.B. schreibst: printf("Hallo"); //hier ist Hallo der String.
So deklarierst du einen String:

```
char wort[100];
```

Es ist dem Array ziemlich ähnlich, hat aber als Datentyp immer char. Wenn du einen String einsetzen willst, der den Wert Hallo besitzt geht das so:

```
char wort[] = { "Hallo" };
```

das selbe könntest du auch so schreiben:

```
char wort[] = {'H','a','l','l','o','\0'};
```

Das \0 ist sehr wichtig, da es angibt, dass die Werte davor zusammen einen String ergeben. Man nennt dieses Zeichen Terminierungszeichen. Oben wird es automatisch hinzugefügt. Jetzt ein Beispiel:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
char string1[100]={"Das ist der erste String und jetzt sollte
deine Eingabe folgen :"};
char Eingabe[100];
char string2[100]={"Jetzt kommt der 2.String"};

printf("Das Beispiel:\nBitte gebe ein, was dir gerade einfaellt:
");
fgets(Eingabe, 100, stdin);

strcat(string1, " ");
strcat(string1,Eingabe);

printf("%s%s\n", string1, string2);

system("PAUSE");
return 0;
}
```

Erklärung: Zuerst werden die 3 Strings, string1, Eingabe und string2 deklariert. Danach wird derjenige, der das Programm ausführt, zur Eingabe aufgefordert. Hier wird es interesant, weil hier nicht das erwartete scanf() steht. Da nicht irgendwelche Zahlen benötigt werden, sondern Buchstaben gibt es hier eine neue Funktion. Die fgets() Funktion hat folgende Struktur:

fgets(Stringname, Stringlänge, Standardstream);

Hier müsste alles klar sein, außer dem Standardstream. Der Standardstream kann entweder die Ausgabe(stdout), die Eingabe(stdin) oder die Ausgabe eines Fehlers(stderr) sein. In diesem Fall ist es die Eingabe, also stdin. Zu der Funktion: Die heißt strcat. Sie ist adzu da Strings hintereinander zu hängen. Das " " zählt als Leerstelle. Hier ist das Grundgestell:

strcat(Zielstring, Quellstring)

Das heißt so viel wie 1.String, 2.String. Wenn du die Verbindung ausgeben willst, musst du den ersten String ausgeben, wobei wir bei der nächsten Neuheit wären. Wir haben nämlich ein neues Zeichen für die Ausgabe, das du immer benötigst, wenn du mit Strings arbeitest.

Die Länge eines Strings ermitteln

Hierzu erst das Beispiel, dann die Erläuterung:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
char Eingabe[100];
int Groesse;

printf("Willst du wissen, wie viele Buchstaben du in einem
bestimmten Text geschrieben \nhast? Dann gebe diesen hier ein :
");//Diese 2 Zeile müssen in einer Zeile stehen!
fgets(Eingabe,100,stdin);

Groesse=strlen(Eingabe);
printf("Du hast %d Buchstaben eingegeben.\n",Groesse-1);

system("PAUSE");
return 0;
}
```

Dieses Programm zählt die Buchstabenanzahl. Zuerst werden zwei Variablen deklariert, Eingabe und Groesse. Da der Benutzer einen Text eingeben muss, kommt die Funktion fgets() ins Spiel, die du bereits im letzten Kapitel kennen gelernt hast. Jetzt kommt wieder etwas Neues in dieser Zeile:
Groesse=strlen(Eingabe);
Diese Zeile bewirkt, dass der Variablen Groesse die Anzahl der Ziffern, die eingegeben wurden und einer Variable angehören(in diesem Fall ist es die Variable Eingabe) übergeben wird. Du wirst dich vielleicht fragen, warum bei der Ausgabe von Groesse 1 subtrahiert wird. Die Antwort ist, dass strlen immer ein Leerzeichen hinzufügt.

Das Vergleichen der Strings

Natürlich gibt es auch eine Funktion, mit der man Strings vergleichen kann. Sie heißt strcmp(). Sie ist in einer Bibliothek deklariert, die string.h heißt. So kannst du die Funktion einsetzen:

```
strcmp(String1,String2);
```

Die beiden Strings in der Klammer werden miteinander verglichen. Dazu ein Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char Buch[] = { "Merlin" };

int main(int argc, char *argv[])
{
char Buchabfrage[6];

printf("Wie heisst dein Lieblingsbuch : ");
scanf("%s",&Buchabfrage[0]);

if((strcmp(Buch, Buchabfrage)) == 0)
printf("Ja, das ist auch mein Lieblingsbuch\n");
else
printf("Das ist auch ein gutes Buch. Aber mein Lieblingsbuch ist
Merlin\n");

system("PAUSE");
return 0;
}
```

Erklärung: Am Anfang wird die neue Bibliothek angegeben. Das schreibt man genauso, wie die anderen Bibliotheken auch, nur dass diese Bibliothek string.h heißt. Danach kommt noch einmal etwas interessantes, nämlich eine globale Variable, die hier Buch heißt. Schließlich erfolgt wieder die Eingabeaufforderung, die ich dir ja schon mehrmals erklärt habe. Dann kommt der Vergleich der beiden Strings in einer if-Verzweigung. Diese Zeile:
if((strcmp(Buch, Buchabfrage)) == 0)
heißt soviel, wie: wenn Buch = Buchabfrage, führe die folgenden Anweisungen aus. Hier musst du allerdings keine geschweiften Klammern schreiben. Das ist doch auch einmal nicht schlecht, oder?

sprintf()

sprintf() ist eine Funktion, mit der du Zahlen in einen String umwandeln kannst. sprintf() schaut einwenig aus, wie printf(). Mit printf() schreibst du ja praktisch etwas auf den Bildschirm und mit sprintf() schreibt man die Zahlen in einen String. Also haben die beiden "gewisse" Ähnlichkeiten. So sieht die Struktur aus:

```
sprintf(Stringname,"%d",Variablenname²);
```

²Diese Variable muss eine Zahl beinhalten.
Hier ein Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
```

```
{
int Zahl=12345;
char String[20];

sprintf(String,"%d",Zahl);

printf("Der Wert von der Variable String: %s\n",String);

system("PAUSE");
return 0;
}
```

Weitere Funktionen, die das selbe bewirken sind:

atoi() : für int Werte
atof() : für double Werte
strol() : für long Werte
strtod(): für double Werte

Du kannst die Werte mit diesen Funktionen aber nicht formatieren! Besser ist also sprintf.

sscanf()

sscanf() ist eigentlich genau das Gegenteil von sprintf(). Die Funktion verwandelt einen String in Zahlen. So sieht die Funktion aus:

sscanf(Stringname,"%d",&Variablenname);

Jetzt ein Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
int Zahl;
char String[]={"12345"};

sscanf(String,"%d",&Zahl);

printf("Der Wert von der Variable Zahl: %d\n",Zahl);

system("PAUSE");
return 0;
}
```

Einen String in einen anderen String kopieren - strcpy

Das Problem bei Strings ist, das man sie nicht gleichsetze kann, wie z.B. Int-Variablen. Bei den Strings braucht man eine eigene Funktion dazu:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
char Null[900];
char Text[]={"Hallo ich bin ein String."};

printf("Null-String: %s\n",Null);
printf("Text-String: %s\n\n",Text);

strcpy(Null,Text);

printf("Null-String: %s\n",Null);
printf("Text-String: %s\n\n",Text);

strcpy(Text,"");

printf("Null-String: %s\n",Null);
printf("Text-String: %s\n\n",Text);

system("PAUSE");
return 0;
}
```

Der erste Parameter von strcpy ist der String, der mit dem Text beschrieben werden soll. Der Zweite ist der String der kopiert werden soll.

Einen String an einen anderen String anhängen - strcat(Wiederholung)

Gleich ein Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
char Uhrzeit[900];
char Zeit[]={"16:23:01"};

strcpy(Uhrzeit,"Es ist ");
strcat(Uhrzeit,Zeit);
strcat(Uhrzeit," Uhr.\n\n");

printf("%s",Uhrzeit);

system("PAUSE");
return 0;
}
```

Erste String von strcat ist der String, an den der andere angehängt werden soll. Daraus ergibt sich, dass der zweite Parameter der String ist, der angehängt werden soll.

Parsen

Parsen. Das bedeutet bestimmte Informationen aus einem Text herauszuholen, zu parsen. Es gibt einige Funktionen, die sehr nützlich für das Parsen sind. Die meiste sind in der Bibliothek string.h deklariert. Es sind also String-Funktionen, anders gesagt, Funktionen, mit denen man einen Text bearbeiten kann.

In einem String nach einem String suchen - strstr

Zu strstr:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
char Text[]={"Hallo das ist die Nachricht 100"};
char Such[]={"Nachricht"};
char* Ergebnis;

Ergebnis=strstr(Text,Such);
printf("%s\n\n",Ergebnis);

system("PAUSE");
return 0;
}
```

Die Funktion strstr setzt man meistens ein, um herauszufinden, was hinter einer bestimmmten Stelle in einem Text steht. In diesem Beispiel sucht man nach dem String "Nachricht" und bekommt als Ergebnis "Nachricht 100". Dieses Ergebnis kann man durch einen Zeiger betrachten. Dieser Zeiger zeigt auf das gesuchte Textstück. Einen solchen Text-Zeiger deklariert man so:

```
char* Variablenname;
```

strstr zeigt auf das erste Auftreten des gesuchten Strings. Wenn dieser String in dem Text nicht vorkommt, ist der Zeiger "NULL".

In einem String nach einem Zeichen suchen - strrchr

strrchr ist eigentlich fast dasselbe, wie strstr, aber dort wird eben nach einem Zeichen gesucht:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
```

```
char Text[]={"16:00:01"};
char Such=':';
char* Ergebnis;

Ergebnis=strrchr(Text,Such);
printf("%s\n\n",Ergebnis);

system("PAUSE");
return 0;
}
```

Wie du vielleicht schon gemerkt hast, gibt es doch noch einen Unterschied zwischen den beiden Funktion. strrchr zeigt nämlich auf das letzte Auftreten des gesuchten Zeichens. Sonst ist das Schema aber gleich.

strtok

Beim Parsen kommt es immer wieder vor, dass Informationen zwischen Kommata, zwischen Punkten oder sonstigen Zeichen stehen. Um diese Informationen herauszufiltern gibt es die Funktion strtok:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
char Uhrzeit[]={"17:57:10"};
int o=0;

char* pToken = strtok(Uhrzeit,":");
if (pToken)
{
printf("Anzahl der Stundenzahl: %s\n",pToken);
o++;

while ( (pToken = strtok(NULL, ":")) )
{
if(o==1)
printf("Anzahl der Minutenzahl: %s\n",pToken);
if(o==2)
printf("Anzahl der Sekundenzahl: %s\n",pToken);
o++;
}
}

system("PAUSE");
return 0;
}
```

strtok zeigt immer auf den String bis zum ersten Zeichen, das nicht gewollt ist. Hier ist es so:

String am Anfang: 17:57:10
String bis zum ersten ":": 17
String bis zum zweiten ":": 57
Stringrest: 10

Beim ersten Mal muss man den String angeben, der gefiltert werden soll. Danach muss man nur noch "NULL" angeben, damit er weiter filtert.

Zusammenfassung zu den Stringfunktionen

Ich hab mir gedacht, es wäre vielleicht ganz interessant zu sehen, wie man die Ip des Standardgateways aus ipconfig herausparst. Hier eine Version dazu:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
FILE *f;

char* Ip;
char Ipchar[1024];
char Text[1024];
char Wissen[]={"Standardgateway . . . . . : "};

int Anzahl=strlen(Wissen);
int i;
int o=0;
int Ipteil[4];

system("ipconfig > ip.txt");

f=fopen("ip.txt","r");

while(fgets(Text,sizeof(Text),f)!=0)
{
if(strstr(Text,Wissen)!=NULL)
{
Ip=strstr(Text,Wissen);
strcpy(Ipchar,Ip);

for(i=0;i<Anzahl;i++)
{
Ipchar[i]=Ipchar[i+Anzahl];
}

char* pToken = strtok(Ipchar,".");
if (pToken)
{
sscanf(pToken,"%d",&Ipteil[o]);
o++;
```

```
while ( (pToken = strtok(NULL, ".")) )
{
sscanf(pToken,"%d",&Ipteil[o]);
o++;
}
}
break;
}
}
fclose(f);

printf("Standardgateway: ");
for(i=0;i<o-1;i++)
{
printf("%d.",Ipteil[i]);
}
printf("%d\n\n",Ipteil[i]);

system("PAUSE");
return 0;
}
```

Zuerst wird der Text des Dos-Befehles ipconfig in die Datei ip.txt kopiert. Das geschieht in dieser Zeile:

```
system("ipconfig > ip.txt");
```

Danach wird die Datei zeilenweise eingelesen. Durch die Funktion strstr wird überprüft, ob in dieser Zeile der Textausschnitt "Standardgateway . . . . . : " vorkommt. Wenn dieser vorkommt, weiß man, dass in dieser Zeile auch die Ip dazu steht. Danach überschreibt man "Standardgateway . . . . . : " mit dem Text, der dahinter steht, das heißt mit der Ip. Danach habe ich noch die einzelnen Teile der Ip in ein Array geschrieben. Da die Information zwischen mehreren Punkten steht, kann man hier die Funktion strtok verwenden. Jetzt hat man die Ip.

Strukturen

Mit Strukturen kann man eine Sammlung von unterschiedlichen Variablen darstellen, die unter einem gewählten Namen zusammengefasst werden. Die Deklarierung der Struktur sieht so aus:

```
struct Strukturtypname{
Variable1;
Variable2;
...
};
```

Am Anfang steht immer das struct als Einleitung. Dann der Name des Strukturtypes, den du aussuchen kannst und in den geschweiften Klammern stehen die verschiedenen Variablen. Da der Syntax dem der Funktion ähnlich sieht, vergessen manche den Strichpunkt nach der geschweiften Klammer. Das ist ein weiterer sehr häufig gemachter Fehler! Beispiel(aber nicht zu dem Fehler!):

```
#include <stdio.h>
```

```
#include <stdlib.h>

struct Adresse{
char Name[200];
char Ort[200];
int Anzahl_der_Eintraege;
char Geburtsdatum[100];
};

int main(int argc, char *argv[])
{
struct Adresse Namen;

Namen.Anzahl_der_Eintraege=1;
strcpy(Namen.Name,"Felix Neutatz");
strcpy(Namen.Ort,"...");
strcpy(Namen.Geburtsdatum,"19.9.1991");

printf("Das ist der %d.Eintrag\n", Namen.Anzahl_der_Eintraege);
printf("Name: %s\n", Namen.Name);
printf("Ort: %s\n", Namen.Ort);
printf("Geburtsdatum: %s\n",Namen.Geburtsdatum);

system("PAUSE");
return 0;
}
```

Erklärung: Zuerst kommt diesmal die Struktur. Der Strukturtyp ist Adresse. In der Struktur stehen 3 Strings und eine Integer Variable. Jetzt zur main-Funktion. Hier wird die Variable Namen mit dem Datentyp Adresse deklariert. Schließlich zur Wertübergabe. Bei Zahlen werden die Werte, wie immer mit einem = an die Variable übergeben. Aber die Schreibweise der Variable ist anders. Sie sieht so aus:

```
Name_der_Variable . Name_der_Variable_aus_der_Struktur
```

Bei Strings ist die Wertübergabe allerdings etwas anders. Man übergibt dem String einen bestimmten Wert durch die Funktion strcpy(). Im Grunde sieht diese folgender Maßen aus:

strcpy(Name_der_Variable . Name_der_Variable_aus_der_Struktur , "Wort"); Jetzt hat der String den Wert Wort. Wenn du die Strings und die Variable ausgeben willst, machst du dies mit printf(). Da kannst du vorgehen, wie du es gewohnt bist. Jetzt ein komplexeres Beispiel:

```
#include <stdio.h>
#include <stdlib.h>

struct Adresse{
char Name[200];
char Ort[200];
int Anzahl_der_Eintraege;
char Geburtsdatum[100];
};

int main(int argc, char *argv[])
```

```
{
struct Adresse Namen[100];
int f=1;
int Aktion=0;
int Wissen;
int Zahl;
char Nam[200];
char Ort[200];
char Geburt[200];
int i;

while(Aktion!=3) {
system("cls");
printf("Adressbuch:\nGebe die jeweilige Zahl ein um folgende
Aktionen auszufuehren:\n<1> Neue Adresse eingeben\n<2> Adresse
abrufen\n<3> Ende\n");
printf("Deine Wahl : < >\b\b");
scanf("%d",&Aktion);
fflush(stdin);
printf("\n");

system("cls");

switch(Aktion)
{
case 1 :
Namen[f].Anzahl_der_Eintraege=f;
printf("Das ist der %d.Eintrag.\n",f);
printf("Name: ");
fgets(Namen[f].Name,200,stdin);
printf("Wohnort: ");
fgets(Namen[f].Ort,200,stdin);
printf("Geburtsdatum : ");
fgets(Namen[f].Geburtsdatum,100,stdin);
printf("\n");
f++;
break;

case 2 :
printf("Was fuer eine Adresse benoetigst du? Was weisst du noch
von jener Adresse?\nWenn du weisst, der wievielte Eintrag die
Adresse war, druecke die 1, wenn du den Namen der Person kennst,
die 2, wenn du den Wohnort weisst die 3 und \nfuer das
Geburtsdatum die 4: ");
scanf("%d",&Wissen);
fflush(stdin);
printf("\n");
system("cls");
if(Wissen==1)
{
printf("Gebe die Zahl ein: ");
scanf("%d",&Zahl);
fflush(stdin);
```

```
system("cls");
if((Zahl >f)||(Zahl<0))
{
printf("Adresse nicht gefunden!");
}
else
{
printf("Adressnummer: %d\n", Namen[Zahl].Anzahl_der_Eintraege);
printf("Name: %s", Namen[Zahl].Name);
printf("Ort: %s", Namen[Zahl].Ort);
printf("Geburtsdatum: %s\n\n", Namen[Zahl].Geburtsdatum);
}
}
if(Wissen==2)
{
printf("Gebe den Namen ein: ");
fgets(Nam,200,stdin);
system("cls");
for(i=1;i<100;i++)
{
if((strcmp(Namen[i].Name,Nam))==0)
{
printf("Adressnummer: %d\n", i);
printf("Name: %s", Namen[i].Name);
printf("Ort: %s", Namen[i].Ort);
printf("Geburtsdatum: %s\n\n", Namen[i].Geburtsdatum);
}
}
}
if(Wissen==3)
{
printf("Gebe den Ort ein: ");
fgets(Ort,200,stdin);
system("cls");
for(i=1;i<100;i++)
{
if((strcmp(Namen[i].Ort,Ort))==0)
{
printf("Adressnummer: %d\n", i);
printf("Name: %s", Namen[i].Name);
printf("Ort: %s", Namen[i].Ort);
printf("Geburtsdatum: %s\n\n", Namen[i].Geburtsdatum);
}
}
}
if(Wissen==4)
{
printf("Gebe das Geburtsdatum ein: ");
fgets(Geburt,200,stdin);
system("cls");
for(i=1;i<100;i++)
{
if((strcmp(Namen[i].Geburtsdatum,Geburt))==0)
```

```
{
printf("Adressnummer: %d\n", i);
printf("Name: %s", Namen[i].Name);
printf("Ort: %s", Namen[i].Ort);
printf("Geburtsdatum: %s\n\n", Namen[i].Geburtsdatum);
}
}
}
system("PAUSE");
break;
}
}

return 0;
}
```

Nächste Erklärung: Dieses Programm soll eine Art Adressbuch sein, in das man Adressen eingeben und abfragen kann. Am Anfang kommt die gleiche Struktur, wie im letzten Beispiel. Aber bei der Deklaration in der Main-Funktion wird es interessant. Genau in dieser Zeile:

```
struct Adresse Namen[100];
```

Der Typ der Variable ist Adresse. Der Name ist Namen und jetzt kommt etwas, das du eigentlich schon kennst. Diese Variable ist ein Array. Du kannst also 100 Adressen(eigentlich 99, da f=1) eingeben, aber auch nicht mehr. Danach kommen noch weitere Variablen, die gebraucht werden. Schließlich wird der Benutzer aufgefordert, ob er entweder eine Adresse eingeben oder eine Adresse abzufragen oder beenden will. Davor kam noch eine while-Schleife mit der Bedingung, dass die Variable ungleich 3 ist. Nach der Eingabeaufforderung folgt eine switch-Verzweigung. Wenn du 1 eingegeben hast, willst du eine Adresse eingeben. Ich habe mir das so vorgestellt, dass ich der Adresse zuerst eine Zahl zuteile z.B. 1.Adresse, 2.Adresse... . Gleich danach gebe ich diese Zahl auf dem Bildschirm aus. Danach wird nach dem Namen, dem Wohnort und nach dem Geburtsdatum gefragt. Man hätte natürlich auch nach der Telefonnummer fragen können. Am Schluss von case 1 wird f mit 1 addiert. Somit nimmt der Computer nicht mehr Namen[1] sondern Namen[2]. Bei case 2 wird der Benutzer gefragt, was er von der bereits eingegebenen Adresse weiß. Durch mehrere if-Verzweigungen(auch eine if else Verzweigung) wird so der Rest, der Adresse ausgegeben. Die if else Verzweigung und eine der if Verzweigungen solltest du näher betrachten, wenn du dieses Programm verstehen willst.Die if else Verzweigung trifft auf dich, wenn du sagst, dass du die Adressnummer weißt. Zuerst zum if Teil. Die Bedingung dieses Teils ist: wenn die Eingabe größer als f ist oder die Eingabe kleiner als 0 ist. Diese Bedingung sollte klar sein. Kleiner als 0 geht ja nicht, da es keine solche Variable gibt: Namen[-8] und größer als f kann die Eingabe auch nicht sein, weil man die Adresse dann noch nicht geschrieben hat. Der else Teil ist auch nicht viel schwerer. Dort werden einfach alle Daten ausgegeben. Jetzt zu einer if-Verzweigung. Diese beinhaltet zuerst eine for-Schleife, die solange 1 dazu addiert bis das Array erreicht ist, mit dem der Name oder Ort oder Geburtsdatum übereinstimmt. Das System ist dem Passwort ähnlich. Man vergleicht nämlich die Eingabe mit dem String im Array. Das war das etwas komplexere Programm. Wenn du dazu Fragen hast, schick mir einfach eine Email. Ich antworte gerne.

Matrix

Eine Matrix kann man sich wie eine Tabelle vorstellen. Eine Matrix wird so deklariert:

```
int Matrix[2][2];
```

Hier würde die Tabelle so aussehen:

```
---------
| | |
---------
| | |
---------
```

Wenn man etwas in das obere linke Feld eintragen will, geschieht das so:

```
Matrix[0][0]=1;
```

Wenn man etwas in das untere rechte Feld eintragen will, kann man folgendes schreiben:

```
Matrix[1][1]=1;
```

Wichtig ist, dass man bei 0 anfängt zu zählen! Wenn man beide Anweisungen ausgeführt hat, würde die Tabelle aussehen:

```
---------
|1| |
---------
| |1|
---------
```

Jetzt eine kleine Übung zum Thema Matrix. Ein Programm soll eine Matrix[10][10] mit den Zahlen 1-100 jeweils im 10er Schritt füllen:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
int Matrix[10][10];
int x,y;

for(y=0;y<10;y++)
{
for(x=0;x<10;x++)
{
Matrix[y][x]=y*10+x+1;
}
}

for(y=0;y<10;y++)
{
for(x=0;x<10;x++)
{
printf("%3d ",Matrix[y][x]);
}
printf("\n");
}
```

```
system("PAUSE");
return 0;
}
```

Dateibezogene Ein- und Ausgaben

Streams

Die dateibezogenen Ein- und Ausgaben bezeichnet man auch als Streams(Datenströme). Diese Standardstreams kennst du bereits:

- Standardausgabe : stdout
- Standardeingabe: stdin

Neu:

- Standardfehlerausgabe : stderr

stdin kennst du von der Funktion fgets() und eigentlich benutzt du unwissend auch stdout immer, wenn du printf() einsetzt.

Dateistream öffnen

Wenn du in eine Datei schreiben oder eine Datei lesen willst, musst du sie mit einem Stream verbinden. Dieser Stream heißt FILE. FILE benötigt die Bibliothek stdio.h, da FILE ein Zeiger auf eine dort vorhandene Struktur ist. Eine Zeiger(engl. pointer) zeigt auf eine Adresse einer Variablen oder auch, wie in diesem Fall einer Datei. So deklariert man einen Zeiger:

```
Datentyp *Zeigername;
```

In unserem Fall ist der Datentyp FILE. Der Zeigername ist, wie immer beliebig. Jetzt ein einfaches Beispiel:(Für das Beispiel musst du eine Textdatei speichern. Öffne dazu den Editor(Start->Programme->Zubehör->Editor) und speichere eine Datei(mit Pfad). Du musst ihn dann aber auch im Programm umändern!)

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
FILE *f;// Diese Zeile ist der Stream oder die Verbindung

f = fopen("c:\\Mein_Ordner\\Text.txt","r");

system("PAUSE");
return 0;
}
```

Mit fopen() kann man die Datei öffnen. Wichtig bei der Pfadangabe ist, dass du die \ immer 2 Mal schreibst. So sieht das Grundgestell der Funktion aus:

```
fopen("Dateiname", "Modus");
```

Der Modus ist hier r. Hier eine Auflistung aller Modi:

r : Öffnet die Datei zum Lesen. Wenn der Pfad falsch eingegeben wurde oder Fehler vorkommen, liefert die Funktion fopen() NULL.

w : Öffnet die Datei, falls sie vorhanden ist, wird sie überschrieben, wenn sie noch nicht existiert, wird sie erstellt. Man kann die Datei aber nicht lesen, man kann sie nur beschreiben.

a : Öffnet die Datei, falls sie vorhanden ist, wird das Geschriebene angehängt, wenn sie noch nicht existiert, wird sie erstellt. Man kann aber ebenfalls nur schreiben.

b : Öffnet die Datei im binären Modus. So kann man z.B. auch exe-Dateien oder sonstige binäre Dateien auslesen und beschreiben. rb: Öffnet die Datei zum binären und normalen Lesen. wb: Öffnet die Datei zum binären und normalen Schreiben. r+ : Öffnet die Datei zum Lesen oder Schreiben. Wenn der Pfad falsch eingegeben wurde oder Fehler vorkommen, liefert die Funktion fopen() NULL.

w+ : Öffnet die Datei, falls sie vorhanden ist, wird sie überschrieben, wenn sie noch nicht existiert, wird sie erstellt. Bei diesem Modus kann man die Datei lesen oder beschreiben. Wenn der Pfad falsch eingegeben wurde oder Fehler vorkommen, liefert die Funktion fopen() NULL.

a+ : Die gleiche Bedeutung, wie bei dem Modus a aber man kann die Datei lesen oder beschreiben. Wenn der Pfad falsch eingegeben wurde oder Fehler vorkommen, liefert die Funktion fopen() NULL.

Diese ganzen Modi kann man auch hintereinander schreiben, wie du bei "wb" schon gesehen hast. Ein weiteres Beispiel wäre:

```
fopen("Datei.txt","w+rb");
```

Bei diesem Bespiel kann man im binären Modus lesen und schreiben.

Da du jetzt weißt, dass bei dem Modus r fopen() NULL liefert, falls Fehler auftreten, wird das Programm jetzt noch ein wenig verbessert:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
FILE *f;// Diese Zeile ist der Stream oder die Verbindung

f = fopen("c:\\Text.txt","r");

if(f==NULL)
{
printf("Fehler beim Oeffnen! Bitte ueberpruefe deinen Pfad!\n");
}
else
{
printf("Die Datei ist erfolgreich geoeffnet worden.\n");
}
```

```
system("PAUSE");
return 0;
}
```

Beschreiben einer Datei

Es gibt verschiedene Möglichkeiten eine Datei zu beschreiben:

Zeichenweises Beschreiben: Ein Buchstabe bzw. Zahl wird in die Datei geschrieben. Syntax:

```
fputc(Zeichen, Stream);
```

Zeilenweises Beschreiben: Ein String wird in die Datei geschrieben. Syntax:

```
fputs(String, Stream);
```

Blockweises Beschreiben: Damit kannst du mehrere Strings auf einmal in eine Datei schreiben. Syntax:

```
fwrite(Adresse, Größe, Anzahl_der_Blöcke, Stream);
```

Beispiel(Texteditor):

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
FILE *f;
char Texteingabe[600];
char file_name[255];

printf("Wie soll die Textdatei heissen:\n");
scanf("%s",&file_name);
fflush(stdin);

f = fopen(file_name,"w");

if(f==NULL)
{
printf("Fehler beim Oeffnen! Bitte ueberpruefe deinen Pfad!\n");
}
printf("Gebe bitte hier den Text ein:\n");
fgets(Texteingabe, sizeof(Texteingabe), stdin);
fputs(Texteingabe, f);

fclose(f);

system("PAUSE");
return 0;
}
```

Erklärung: Kurz nach dem Anfang wirst du aufgefordert, den Pfad einer Datei einzugeben. Schließlich wird der Pfad mit einer if-Verzweigung überprüft. Danach wird der Benutzer aufgefordert, den Text einzugeben. Mit der Funktion fgets() wird die Eingabe gelesen. Der sizeof Operator liefert die Anzahl der Bytes. Mit fputs() wird der Text in die Datei kopiert und zum Schluss wird die Datei mit der Funktion fclose() geschlossen.

Eine Datei lesen

Hier gibt es wieder die verschiedenen Möglichkeiten des Lesens:

Wenn du die Datei zeichenweise lesen willst, benötigst du die Funktion fgetc mit folgendem Syntax:
Stringname = fgetc(Streamname);

Wenn du die Datei zeilenweise lesen willst, benötigst du die Funktion fgets mit folgendem Syntax: fgets(Stringname, Streamname);

Wenn du die Datei blockweise lesen willst, benötigst du die Funktion fread mit folgendem Syntax: fread(Adresse, Größe, Anzahl_der_Blöcke, Streamname);

Dazu wieder ein Beispiel:(Das Programm soll aus der Datei den Text lesen, in die du mit unserem Texteditor deinen Text eingegeben hast.)

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
FILE *f;
char Text[600];
char file_name[255];

printf("Wie soll die Textdatei heissen, die du oeffnen
willst:\n");
scanf("%s",&file_name);
fflush(stdin);

f = fopen(file_name,"r");

if(f==NULL)
{
printf("Fehler beim Oeffnen! Bitte ueberpruefe deinen Pfad!\n");
}
printf("\nDas ist der Text der in der Datei ( \"%s\" )
steht:\n\n",file_name);
while( fgets(Text, sizeof(Text), f) !=0 )
fputs(Text, stdout);
printf("\n");

system("PAUSE");
return 0;
}
```

Interessant wird es eigentlich erst bei dieser Zeile:

```
while( fgets(Text, sizeof(Text), f) !=0 )
```

Diese Zeile bedeutet: wiederhole solange eine Zeile zu lesen, bis es keine mehr zu lesen gibt. Danach wird der gelesene Text mit fputs() ausgegeben.

Datei kopieren

Das Kopieren einer Datei ist eigentlich nichts anderes, als die Datei zu lesen und das Gelesene in eine leere Datei zu schreiben. Hier das Beispiel:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
FILE *Einlesen;
FILE *Schreiben;
char Daten[1024];

Schreiben=fopen("Neu.exe","w+b");
Einlesen=fopen("Datei.exe","rb");
while(fread(Daten,sizeof(short),1,Einlesen)!=0)
{
fwrite(Daten,sizeof(short),1,Schreiben);
}
fclose(Einlesen);
fclose(Schreiben);

return 0;
}
```

Die Datei, die in diesem Programm kopiert wird, heißt Datei.exe. Die Kopie der Datei befindet sich dann im gleichen Ordner, in dem sich das Programm befindet und heißt Neu.exe.

Es gibt aber noch einen viel schnelleren Weg und dieser heißt "CopyFile". Hier das Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

int main(int argc, char *argv[])
{
CopyFile("C:\\musicplayer.exe","C:\\Users\\Felix\\Desktop\\music.e
xe",FALSE);
return 0;
}
```

Hier wird das Programm musicplayer.exe auf meinen Desktop kopiert und heißt dann music.exe. Der dritte Parameter kann entweder FALSE oder TRUE heißen. Wenn er FALSE heißt und es

bereits eine Datei auf dem Desktop gibt, die music.exe heißt, wird diese überschrieben. Wenn der Parameter TRUE heißt wird sie nicht überschrieben.

Öffnen der Exe-Dateien

Dafür gibt es natürlich auch eine Funktion, die du auch schon sehr sehr oft benutzt hast. Es ist die Funktion system(). Dazu ein Beispiel:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
system( "\"C:\\Programme\\RobotKarol\\karol.exe\"" );

return 0;
}
```

Ich habe ein Programm geöffnet, das ich persönlich sehr sehr gut finde. Damit kann man wirklich gut programmieren lernen. Es ist für Anfänger aber auch für Profis sehr geeignet. Wenn du es downloaden willst, hier der Link: http://www.schule.bayern.de/karol/download.htm
Man hätte genau so gut ein Spiel öffnen können. Die Funktion hätte dann so ausgesehen:
system("\"C:\\WINDOWS\\system32\\spider.exe\"");

Öffnen einer Anwendung mit ShellExecute

Die 2. Methode, aus meiner Sicht eine elegantere Methode, ein Programm zu starten, ist die Funktion ShellExecute. Mit dieser Funktion kann man sogar eine Internetseite öffnen. Hierzu ein Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

int main(int argc, char *argv[])
{
ShellExecute(NULL, "open", "www.c-programmieren.com", NULL, NULL,
SW_SHOW);

return 0;
}
```

Der erste Parameter von ShellExecute ist das Fenster, in dem das Programm gestartet werden soll, aber da wir noch nichts mit Fenstern gemacht haben, brauchen wir diesen Parameter noch nicht.
Der 2. Parameter ist die Aktion, die mit der Datei durchgeführt werden soll. Wir wollen die Website öffnen, also brauchen wir hier open. Weitere Aktionen, die ausgeführt werden können, sind:

edit: öffnet die Datei im Editor
print: druckt die Datei
explore: öffnet den Explorer mit den angegebenen Pfad
find: öffnet den "Such-Explorer"

Der 3. Parameter ist die Datei, die Website, das Programm oder der Pfad, der geöffnet werden soll. Der nächste Parameter ist einfach NULL. Der 4. Parameter ist das Arbeitsverzeichnis, das angegeben werden kann. Das ist aber kein Muss. Man kann einfach NULL sagen. Als letzten Parameter muss man die Art, des Öffnens definieren. Hier ein paar Möglichkeiten.

SW_MINIMIZE = öffnet das Programm minimiert
SW_MAXIMIZE = öffnet das Programm maximiert
SW_SHOW = öffnet das Programm mit den aktuellen Positionsdaten
SW_SHOWNORMAL = öffnet das Programm "normal"
SW_HIDE = öffnet das Programm als Prozess, es kann nicht in der Taskleiste gesehen werden, nur im Taskmanager unter "Prozesse"

Jetzt noch einmal ein Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

int main(int argc, char *argv[])
{
ShellExecute(NULL, "open", "calc.exe", NULL,
"C:\\Windows\\system32", SW_SHOW);
return 0;
}
```

Dieses Programm öffnet den Taschenrechner von Microsoft, wobei ich meinen besser finde*g*.

Öffnen und Schließen einer Anwendung mit CreateProcess und TerminateProcess

Du denkst bestimmt, dass es ganz nett ist so viele Möglichkeiten für das Öffnen einer Datei zu haben, aber wo ist da der Sinn? Ich finde, dass CreateProcess im Grunde die Erweiterung von ShellExecute ist, obwohl ShellExecute auf den Befehl CreateProcess zurückgreift. Öffnen wir mal wieder eine Datei:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

int main(int argc, char *argv[])
{
STARTUPINFO si;
PROCESS_INFORMATION pi;

ZeroMemory( &si, sizeof(si) );
si.cb = sizeof(si);
ZeroMemory( π, sizeof(pi) );

CreateProcess(NULL,"notepad.exe", NULL, NULL, FALSE, 0, NULL,
NULL, &si, π);

return 0;
```

```
}
```

Zuerst begegnen dir in diesem Programm zwei Strukturen STARTUPINFO und PROCESS_INFORMATION. Sie sind sehr wichtig, weil man in diese Strukturen alle Informationen über den gestarteten Prozess abgespeichert werden. Das siehst du an den & Zeichen bei den letzten beiden Parametern von CreateProcess. Zu allen Parametern:

```
CreateProcess(
Name(des zu öffnenden Programmes),
Befehl,
Prozessattribute(Sicherheits-Attribute, bei NULL sind Standard-
Sicherheits-Attribute),
ThreadAttribute(Sicherheits-Attribute, bei NULL sind Standard-
Sicherheits-Attribute),
Flag(FALSE=erbt die Einstellung nicht,TRUE=erbt die Einstellung),
Priorität(bei NULL ist die Priorität auf normal),
Zeiger auf den Umgebungs-Block(wichtig für die Vererbung, aber
nicht für uns),
Arbeitsverzeichnis,
& STARTUPINFO-Struktur(Infos werden in die Struktur eingelesen),
& PROCESS_INFORMATION-Struktur(Infos werden in die Struktur
eingelesen),
);
```

Du wirst dich jetzt bestimmt fragen, warum ich den Namen des Programmes als zweiten Parameter angegeben habe. Alle Dateien, die man mit cmd(Start->Programme->Zubehör->Eingabeaufforderung) sofort öffnen kann, kann man auch mit diesem zweiten Parameter sofort öffnen. Man kann also manchmal beides machen:

CreateProcess(NULL,"notepad",NULL,NULL,FALSE, 0, NULL, NULL, &si, π);
CreateProcess("C:\\Windows\\system32\\notepad.exe","",NULL,NULL,FALSE, 0, NULL, NULL, &si, π);

Meistens geht aber nur die zweite Methode, da die Programme sich nicht immer in Ordnern, die in Umgebungsvariablen stehen, befinden oder sich nicht in dem Ordner befinden, in dem sich auch das Programm befindet. Um zu sehen, ob das Programm, das du öffnen willst, in solch einem Ordner gespeichert ist, öffnest du die Eingabeaufforderung und gibst "PATH" ein(natürlich ohne "").
Den Parameter mit dem Arbeitsverzeichnis hast du bereits bei ShellExecute kennengelernt.(Wenn du nicht mehr genau weißt,was dieser Parameter bewirkt, schau einfach noch einmal bei ShellExecute).
Jetzt noch zu ZeroMemory. Diese Funktion leert eine ganze Struktur. Der erste Parameter ist die Struktur mit dem & Zeichen davor und der zweite Parameter ist die Größe der Struktur, die in diesem Fall von der Funktion sizeof ermittelt wird.
In dieser Zeile:

```
si.cb = sizeof(si);
```

-->wird die Größe der Struktur an einen Parameter derselben Struktur übergeben. Die ganze Struktur STARTUPINFO ist so aufgebaut:

```
typedef struct _STARTUPINFO
{
DWORD cb;
```

```
LPTSTR lpReserved;
LPTSTR lpDesktop;
LPTSTR lpTitle;
DWORD dwX;
DWORD dwY;
DWORD dwXSize;
DWORD dwYSize;
DWORD dwXCountChars;
DWORD dwYCountChars;
DWORD dwFillAttribute;
DWORD dwFlags;
WORD wShowWindow;
WORD cbReserved2;
LPBYTE lpReserved2;
HANDLE hStdInput;
HANDLE hStdOutput;
HANDLE hStdError;
} STARTUPINFO, *LPSTARTUPINFO;
```

Genaueres erfahrt ih hier: http://msdn2.microsoft.com/en-us/library/ms686331.aspx

In der Struktur PROCESS_INFORMATION werden folgende Inhalte gespeichert:

```
typedef struct _PROCESS_INFORMATION
{
HANDLE hProcess;
HANDLE hThread;
DWORD dwProcessId;
DWORD dwThreadId;
} PROCESS_INFORMATION, *LPPROCESS_INFORMATION;
```

Es ist ein Handle zum Prozess und zum Thread gegeben. Diese braucht man z.B., wenn man den Prozess wieder beenden will. Außerdem gibt es da noch die IDs, die sind ebenfalls wichtig für den Zugriff auf den Prozess.

Jetzt aber zum Neuen an diesem Thema, das Schließen der Anwendung:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

int main(int argc, char *argv[])
{
STARTUPINFO si;
PROCESS_INFORMATION pi;

ZeroMemory( &si, sizeof(si) );
si.cb = sizeof(si);
ZeroMemory( π, sizeof(pi) );

CreateProcess(NULL,"notepad.exe", NULL, NULL, FALSE, 0, NULL,
NULL, &si, π);
sleep(2000);
```

```
TerminateProcess(pi.hProcess,0);

CloseHandle( pi.hThread );
CloseHandle( pi.hProcess );

return 0;
}
```

Die Notepad.exe wird geöffnet und nach 2 Sekunden(2000ms) wieder geschloßen. Der erste Parameter von TerminateProcess ist das Handle des Prozesses. Der Zweite die Art der Beedigung des Programmes(0=normal beeden). Am Schluss wird noch alles mit der Funktion CloseHandle aufgeräumt.

Verschieben und Umbenennen einer Datei

Dazu gleich ein Beispiel:(Ganz am Anfang solltest du eine Text-Datei erstellen, die das Programm danach gelesen hat. Diese Datei verschieben wir in diesem Beispiel zu einer HTML-Datei.)

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

int main(int argc, char *argv[])
{
MoveFile("c:\\Text.txt","c:\\MeinText.html");//Hier wird die Datei
umbenannt.
MoveFile("C:\\Text.txt","C:\\Programme\\MeinText.html");//Hier
wird sie verschoben.
system("PAUSE");
return 0;
}
```

Wie du siehst, ist die Funktion MoveFile für die Verschiebung und die Umbenennung zuständig. Der erste Dateiname ist die Datei, die verschoben oder unbenannt werden soll und der zweite Name ist die Datei, die entstehen soll. Wenn man den gleichen Pfad aber einen anderen Dateinamen angibt, wird die Datei umbenannt. Wenn man einen anderen Pfad angibt, wird sie verschoben. Es ist sehr wichtig, dass du am Anfang die Bibliothek windows.h deklarierst, sonst erkennt der Compiler die Funktion nicht.

Löschen einer Datei

Dies macht man mit der Funktion remove(). Dazu wieder ein Beispiel:(Die kopierte Datei soll wieder gelöscht werden.)

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
remove("C:\\Programme\\MeinText.html");

system("PAUSE");
```

```
return 0;
}
```

Es ist eigentlich ganz einfach. Die Datei, die in den Klammern von remove steht, wird gelöscht.

Erstellen eines Ordners

Du wirst es nicht glauben, aber da gibt es tatsächlich auch eine Funktion. Diesmal heißt sie mkdir. Hier ein sehr, ja diesmal wirklich sehr kleines Beispiel:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
mkdir("C:\\Mein_neuer_Ordner");

return 0;
}
```

Verzeichnis wechseln

Das Wechseln des Verzeichnises ist genauso leicht, wie das Erstellen. Deswegen will ich dir in diesem Zusammenhang auch noch ein paar andere Funktionen zeigen. Jetzt das Programm:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
char Verzeichnis[900];
FILE *f;

getcwd(Verzeichnis,sizeof(Verzeichnis));

mkdir("C:\\Mein Verzeichnis");
chdir("C:\\Mein Verzeichnis");

f=fopen("Datei.txt","w+");

fprintf(f,"Diese Datei befindet sich im Verzeichnis \"C:\\Mein
Verzeichnis\", weil du mit chdir in dieses Verzeichnis gewechselt
bist.");
fclose(f);

chdir(Verzeichnis);//wechselt in das Verzeichnis in dem sich
dieses Programm befindet

getcwd(Verzeichnis,sizeof(Verzeichnis));
printf("Das aktuelle Arbeitsverzeichnis ist:\n%s\n",Verzeichnis);

system("Pause");
return 0;
```

```
}
```

Die erste neue Funktion ist getcwd. Sie zeigt dir das aktuelle Arbeitsverzeichnis. Wenn das Programm in kein anderes Verzeichnis gewechselt ist, dann ist das Arbeitsverzeichnis das Verzeichnis, in dem das Programm gespeichert ist. Der erste Parameter der Funktion ist der String, der mit dem Arbeitsverzeichnis überschrieben werden soll und der zweite Parameter ist die Größe des Verzeichnisnamens. Jetzt ist die Funktion an der Reihe mit der man den Ordner wechseln kann. Sie heißt chdir. Das kommt vom Englischen change dir und heißt Verzeichnis wechseln. Als Parameter muss man nur den Namen,des Verzeichnises, in das man wechseln will, angeben, falls nötig auch den Pfad. Die letzte neue Funktion heißt fprintf. Mit dieser Funktion kann man wie printf schreiben, aber mit dem Unterschied, dass man diesmal in eine Datei schreibt. Deshalb muss als erster Parameter auch der Name des Streams angegeben werden.

Ordner löschen

Wenn dir das erstellte Verzeichnis nicht mehr gefällt, kannst du es einfach wieder löschen. Der Befehl zum Löschen des oben erstellten Verzeichnisses würde so lauten:

```
rmdir("C:\\Mein_neuer_Ordner");
```

Verzeichnis auslesen

Ein Verzeichnis auslesen, das heißt alle Dateien bzw. Ordner eines Verzeichnisses zu ermitteln. Zuerst das Beispiel, dann die Erklärung:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

int main(int argc, char *argv[])
{
HANDLE fHandle;
WIN32_FIND_DATA wfd;

int i=0;

fHandle=FindFirstFile("C:\\*",&wfd);

do
{
if (wfd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
{
printf("Ordner: %s\n",wfd.cFileName);
}
else
{
printf("Datei: %s\n",wfd.cFileName);
}
i++;
}
while (FindNextFile(fHandle,&wfd));
FindClose(fHandle);
```

```
printf("\n");

system("PAUSE");
return 0;
}
```

Ein Handle ist ein Zeiger auf etwas. In diesem Fall zeigt es auf den Inhalt des Verzeichnises. WIN32_FIND_DATA ist eine Struktur, die einige Merkmale einer Datei enthält. Mit FindFirstFile werden z.B. alle Merkmale der ersten Datei/Ordner in die diese Struktur eingelesen, wie z.B. der Name, die Größe der Datei/Ordner. Genauere Informationen über WIN32_FIND_DATA gibt es hier: msdn2.microsoft.com. Wie gesagt liest FindFirstFile die Daten der ersten Datei in die Struktur. FindNextFile macht das mit dem Rest der Dateien. Eine Kleinigkeit ist auch noch sehr wichtig. Das Sternchen nach dem Pfad des Verzeichnisses. Wenn dieses fehlt, wird gar nichts ausgegeben. Diese Bedingung:

```
if (wfd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
```

schaut, ob die Datei ein Ordner ist und zum Schluss, FindClose(fHandle) schließt den Zeiger. So ein Handle erinnert auch sehr an einen Stream. Da gibt es auch fclose, f=fopen ...

Headerdateien

Hier eine Liste aller Headerdateien, die dem ANSI-C-Standard entsprechen:

assert.h : Diese Headerdatei ist für die Fehlersuche zuständig.

ctype.h : Diese Headerdatei ist für den Zeichentest und die Konvertierung zuständig.

errno.h : Diese Headerdatei ist für die Fehler im Code zuständig.

float.h : Diese Headerdatei ist für die Eigenschaften der Fließkommazahlen zuständig.

limits.h : Diese Headerdatei ist für die Implementierungskonstanten zuständig.

locale.h : Diese Headerdatei ist für die länderspezifischen Eigenschaften zuständig.

math.h : Diese Headerdatei ist für das Mathematische zuständig.

setjmp.h : Diese Headerdatei ist für die unbedingten Sprünge zuständig.

signal.h : Diese Headerdatei ist für die Signale zuständig.

stdarg.h : Diese Headerdatei ist für die variable Parameterübergabe zuständig.

stddef.h : Diese Headerdatei ist für die standardisierten Datentypen zuständig.

stdio.h : Diese Headerdatei ist für die Standard Ein- und Ausgabe zuständig.

stdlib.h : Diese Headerdatei ist für weitere nützliche Funktionen zuständig.

string.h : Diese Headerdatei ist für Stringbearbeitung zuständig.

time.h : Diese Headerdatei ist für die Zeit zuständig.

define

Man kann define sowohl als Variable als auch als Funktion einsetzen. So sieht define in einem Programm aus:

```
#define Name Anweisung(en)/Zahl
```

Hier ein Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
#define Zahl 12

int main(int argc, char *argv[])
{
printf("Das ist define: %d\n",Zahl);

system("PAUSE");
return 0;
}
```

Hier ist define eine Variable mit dem Wert 12. Jetzt dasselbe Beispiel etwas anders:

```
#include <stdio.h>
#include <stdlib.h>
#define Zahl 12
#define Text printf("Das ist define: %d\n",Zahl);

int main(int argc, char *argv[])
{
Text;

system("PAUSE");
return 0;
}
```

#define Text ist eine Funktion, die in der main-Funktion wieder aufgerufen wird. Wenn #define eine Funktion ist, nennt man das ein Makro.

Vordefinierte Makros

Hier sind ein paar davon:

TIME : Enthält die Übersetzungszeit.
DATE : Enthält das Übersetzungsdatum.
LINE : Enthält die Zeilennummer in der Datei.
FILE : Enthält den Namen der Datei.
STDC : Ist 1, wenn ein ANSI-C-Compiler benutzt wurde.

Ein Programm mit Parametern öffnen

Hast du dich schon mal gefragt, warum ein Programm startet, wenn eine Datei aufgerufen wird? Nein. Und wie weiß dieses Programm, welche Datei es anzeigen soll? Das soll in diesem Kapitel geklärt werden. Ich habe einmal erwähnt, dass ich die Parameter von int main erst später erklären will. Jetzt ist die Zeit gekommen. Wenn eine Datei geöffnet wird, wird das Programm mit dem Namen der Datei als Parameter gestartet. Um z.B. die Datei Text.txt mit der Eingabeaufforderung(Start->Programme->Zubehör) im Editor zu öffnen. Muss man diesen Befehl eingeben:

```
notepad Text.txt
```

Jetzt zeigt der Editor die Datei Text.txt an. Um die Datei anzeigen zu können, muss der Editor aber wissen, wie sie heißt. Wie das geht, siehst du jetzt:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
char Zeile[900];
FILE*f;

if(argc==2)
{
printf("Inhalt von: %s\n\n",argv[1]);
f=fopen(argv[1],"r");
while(fgets(Zeile,sizeof(Zeile),f)!=0) printf("%s",Zeile);
fclose(f);
}
else
{
printf("Sie muessen eine Datei als Parameter angeben!\n\n");
}

system("PAUSE");
return 0;
}
```

Zuerst wirst du die Frage stellen, warum argc==2 ? Die Variable argc beinhaltet die Anzahl der Worte der Kommandozeile. Bei "notepad Text.txt" wären es 2. Diese Bedingung würde hier also

zutreffen. char *argv[] ist ein Stringarray. In argv[0] steht das Programm(z.B. notepad) und in argv[1] steht der erste Parameter. Bei uns wäre es "Text.txt". Wenn du das Programm ausgeführt hast, siehst du die Nachricht "Sie muessen eine Datei als Parameter angeben!". Wie gibt man die Datei an? Ganz einfach, klicke mit der rechten Maustaste auf irgendeine deiner Textdateien und klicke auf "Öffnen mit"->"Standardprogramm auswählen", danach auf "Durchsuchen" und öffne in diesem Fenster dein Programm. Jetzt nur noch auf Ok klicken und dein Programm erscheint mit dem Inhalt der Textdatei.

Farbe für Fortgeschrittene(SetConsoleTextAttribute)

Wie die Überschrift schon andeutet, ist diesmal SetConsoleTextAttribute der Schlüssel zur bunten Dos-Konsole. Die Funktion hat zwei Parameter. Der erste zeigt der Funktion, was bunt werden soll. Das ist unsere Ausgabe, also der OUTPUT und der zweite Parameter ist die Farbe, die man allerdings errechnen muss, wie du hier im Beispiel siehst:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

#define schwarz 0
#define blau 1
#define gruen 2
#define cyan 3
#define rot 4
#define magenta 5
#define braun 6
#define hellgrau 7
#define dunkelgrau 8
#define hellblau 9
#define hellgruen 10
#define hellcyan 11
#define hellrot 12
#define hellmagenta 13
#define gelb 14
#define weiss 15

int Farbe(int Text, int Hintergrund)
{
int Colour=Text+16*Hintergrund;
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),Colour);
}

int main(int argc, char *argv[])
{
Farbe(hellrot,schwarz);//Farbe(Schriftfarbe,Hintergrundfarbe);
printf("Roter Text mit schwarzem Hintergrund!\n");

system("PAUSE");
return 0;
}
```

Die Formel für die Farbzahl lautet:

Farbzahl=Schriftfarbe+16*Hintergrundfarbe;

Mit der Funktion GetStdHandle kann man die Konsolenausgabe ermitteln, in dem STD_OUTPUT_HANDLE als Paramter setzt.

Der Thread

Beim Programmieren stößt man manchmal auf das Problem, dass man das Programm gerne 2 oder mehr Aufgaben auf einmal machen lassen würde. Das kommt bei der Programmierung auf Dos-Ebene zwar eher selten bis nie vor, aber bei der Fensterprogrammierung ist das ein wichtiges Thema. Wenn man z.B. einen Chat programmiert, muss das Programm überprüfen, ob Nachrichten empfangen werden müssen und gleichzeitig muss überprüft werden, ob eine Nachricht verschickt werden soll. Ein Thread ist im Grunde nichts anderes als eine Funktion, die aber parallel zum Hauptprogramm ausgeführt wird. Man könnte sich einen Thread auch wie ein zweites Programm im Programm vorstellen. Was genau der Unterschied ist, siehst du in den folgenden Beispielen:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

DWORD WINAPI Thread()
{
while(1)
{
sleep(1);
printf("Ich bin der Thread!\n");
}
return 0;
}

int main(int argc, char *argv[])
{
CreateThread(NULL, 0, Thread, NULL, 0, NULL);

while(1)
{
sleep(1);
printf("Hallo, ich bin das Hauptprogramm!\n");
}

return 0;
}
```

Du siehst, der Thread ist genauso definiert und aufgebaut, wie eine Funktion, aber am Anfang der Deklaration steht "DWORD WINAPI". Man könnte eigentlich auch "int" schreiben, aber dann kommt vom Kompiler eine Warnung. Man bekommt zwar dasselbe Ergebnis, aber die Warnung nervt! Beim Aufruf des Threads gibt es allerdings eine Änderung zur Funktion. Diesmal gibt es sogar eine eigene Funktion zum Aufruf des Threads: CreateThread
Die genaue Definition zu dieser Funktion gibt es hier: msdn2.microsoft.com
Jetzt zum Ergebnis:

Ich bin der Thread!
Hallo, ich bin das Hauptprogramm!
Ich bin der Thread!
Hallo, ich bin das Hauptprogramm!

Es müsste ungefähr so aussehen. Das heißt der Thread läuft, während das Hauptprogramm ebenfalls läuft. Was ist jetzt der Unterschied zur Funktion? Ganz einfach. Eine Funktion ist nichts anderes als eine Auslagerung des Quelltextes. Man könnte also den Sourcecode der Funktion an der Stelle einfügen, an der man sie aufruft. Hier kommt das Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

DWORD WINAPI Thread()
{
while(1)
{
sleep(1);
printf("Ich bin jetzt eine Funktion!\n");
}
return 0;
}

int main(int argc, char *argv[])
{
Thread();

while(1)
{
sleep(1);
printf("Hallo, ich bin das Hauptprogramm!\n");
}

return 0;
}
```

Wenn das Programm so aussehen würde und man die Funktion als Funktion und nicht als Thread aufrufen würde, könnte man den Quelltext auch so schreiben:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

int main(int argc, char *argv[])
{
while(1)
{
sleep(1);
printf("Ich bin jetzt eine Funktion!\n");
}
```

```
while(1)
{
sleep(1);
printf("Hallo, ich bin das Hauptprogramm!\n");
}
return 0;
}
```

Wie du siehst bleibt das Programm dann in der Schleife der Funktion(die jetzt nicht mehr da ist) hängen. Beim Thread ist dies nicht der Fall, weil der Thread parallel neben dem Hauptprogramm läuft.

Die eigene Bibliothek

Ich habe dir schon erklärt, dass eine Bibliothek eine Sammlung einiger Funktionen ist. In diesem Kapitel wollen wir jetzt eine Bibliothek mit den eigenen Funktionen und Variablen erstellen. Eine Bibliothek ist eigentlich ein Programm ohne Mainfunktion. Hier ein Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

#define schwarz 0
#define blau 1
#define gruen 2
#define cyan 3
#define rot 4
#define magenta 5
#define braun 6
#define hellgrau 7
#define dunkelgrau 8
#define hellblau 9
#define hellgruen 10
#define hellcyan 11
#define hellrot 12
#define hellmagenta 13
#define gelb 14
#define weiss 15

int Farbe(int Text, int Hintergrund)
{
int Colour=Text+16*Hintergrund;
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),Colour);
}
```

Das ist schon eine Bibliothek. Sie beinhaltet mehrere Konstanten und eine Funktion. Die letzte Zeile des Quelltextes muss eine Leere sein. Diesen Sourcecode kann man in einer Datei speichern, die z.B. "Bunte_Bibliothek.h" heißt. Wenn man die Bibliothek einbinden will, macht man dies so:

```
#include "Bunte_Bibliothek.h"
```

Wenn man die Bibliothek so einbindet, muss die Bibliothek im gleichen Ordner, wie die c-Datei

sein. Allerdings hat unsere Bibliothek noch einen kleinen Schönheitsfehler. Wenn man die Bibliothek aus Versehen zwei Male einbindet, kommt es zu einem Fehler. Dagegen kann man Folgendes tun:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

#ifndef bunte_Bibliothek_h_INCLUDED
#define bunte_Bibliothek_h_INCLUDED

#define schwarz 0
#define blau 1
#define gruen 2
#define cyan 3
#define rot 4
#define magenta 5
#define braun 6
#define hellgrau 7
#define dunkelgrau 8
#define hellblau 9
#define hellgruen 10
#define hellcyan 11
#define hellrot 12
#define hellmagenta 13
#define gelb 14
#define weiss 15

int Farbe(int Text, int Hintergrund)
{
int Colour=Text+16*Hintergrund;
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),Colour);
}

#endif
```

ifndef scheint die Abkürzung für "if not defined" zu sein. Also wenn "bunte_Bibliothek_h_INCLUDED" noch nicht als define-Konstante definiert wurde, wird der Sourcecode eingebunden und "bunte_Bibliothek_h_INCLUDED" wird definiert. Wenn die Bibliothek also noch einmal eingebunden wird, erkennt die if-Verzweigung das und überspringt den Source. Jetzt viel Spaß bei der Erstellung deiner eigenen Bibliothek.

Die Zeit

Für die Thematik Zeit bietet es sich an eine Struktur zu verwenden, da sehr viele Daten gespeichert werden müssen, wie z.B. Jahr,Monat,Tag,Stunde...
So wurde es auch gemacht. Die Struktur heißt tm. Wenn du also die aktuelle Zeit wissen willst, kannst du es so machen:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

```
int main()
{
struct tm *Zeit;
long Jetzt;
time(&Jetzt);
Zeit = localtime(&Jetzt);

printf("Datum: ");
printf("%d.%d.%d\n",Zeit->tm_mday, Zeit->tm_mon + 1, Zeit->tm_year
+ 1900);

printf("Uhrzeit: ");
printf("%d:%d:%d\n",Zeit->tm_hour, Zeit->tm_min, Zeit->tm_sec);

system("PAUSE");
return 0;
}
```

Zuerst wird die Bibliothek time.h eingebunden. Dort befinden sich alle Funktionen und Strukturen, die mit der Zeit zu tun haben. Danach habe ich den Zeiger Zeit deklariert, der auf die Struktur tm zeigt. Die erste neue Funktion heißt time. Sie ermittelt die Anzahl der Sekunden seit 1.1.1970 0 Uhr und speichert sie anschließend in die Variable, die mit einem &-Zeichen in den Klammern steht. localtime wandelt jetzt die Anzahl der Sekunden in das aktuelle und lokale Datum und Uhrzeit und speichert diese Werte in die Struktur. Um diese Daten auszugeben, muss man jetzt nur noch wissen in welcher Variable(n) sich die Zeit versteckt. Folgende Variablen befinden sich in der Struktur tm:

int tm_sec; Sekunden
int tm_min; Minuten
int tm_hour; Stunden
int tm_mday; Tage
int tm_mon; Monate
int tm_year; Jahre

Es gibt zwar noch ein paar mehr, aber die sind im Moment unwichtig. Du musst also nur noch diese Variablen ausgeben. Um z.B. das aktuelle Jahr auszugeben müsstest du wie folgt vorgehen:

```
printf("%d",Zeit->tm_year + 1900);
```

Es kommt zuerst die Struktur dann ein Pfeil(->) und danach die jeweilige Variable. Aber warum +1900? Weil time die Anzahl der Sekunden seit 1970 errechnet. Dasselbe gilt für den Monat.

Der Zufall

Beim Programmieren braucht man oft eine Zufallszahl z.B. für ein Glücksspiel oder überhaupt für Spiele. Eine solche Zufallszahl generieren wir jetzt:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char *argv[])
```

```
{
int Zufallszahl;

srand(time(0));
Zufallszahl = (rand() % 10 + 1);
printf("Die Zufallszahl ist diesmal %d\n\n",Zufallszahl);

system("PAUSE");
return 0;
}
```

Ganz oben siehst, dass die Bibliothek time.h eingebunden wurde. Der Zufall ist nämlich gar kein Zufall. Die Zahl die wir generieren hängt von der Zeit ab. Für diesen Zeitfaktor ist die Funktion srand(time(0)); zuständig.
(rand() % 10 + 1) diese Zeile heißt, dass die Zufallszahl zwischen 1 und 10 liegen soll.

Audio

Bei meinem Computer piepts wohl?!

Ja in diesem Kapitel macht er das wirklich. Die Funktion dazu heißt:

```
beep(Frequenz,Länge);
```

Zu dieser Funktion gibt es eigentlich nicht mehr viel zu sagen, außer das sie die Bibliothek windows.h braucht. Wenn du mit dieser Funktion Töne mit dem Computer spielen willst, sind hier die Frequenzen der Töne:

c 264
d 297
e 330
f 352
g 396
a 440
h 495
c 528

Ich sag nur noch viel Spaß beim Piepen *g*.

Lieder abspielen ohne sie zu öffnen

An dieser Stelle will ich Bastian Schräder danken, der mich darauf aufmerksam gemacht hat, dass diese Funktion in meiner C Einführung fehlt. Die Funktionn heißt PlaySound(). Gleich ein Beispiel:

```
#include <windows.h>
#include <mmsystem.h>

int main()
{
```

```
PlaySound("Lied.wav",NULL,SND_FILENAME);

system("PAUSE");
return 0;
}
```

Damit dieses Programm funktioniert, musst du oben im Menu von Dev-C++ auf Projekt klicken --> Projekt Optionen --> oben im Fenster auf Parameter-->unter Linker auf Bibliothek/Objekt hinzuf. --> danach musst du eine Datei suchen, die libwinmm.a heißt. Diese findest du so:
Arbeitsplatz --> Lokaler Datenträger(C:) --> Dev-Cpp --> lib --> in diesem Ordner solltest du sie finden. Wenn du sie gefunden hast, klicke auf öffnen. Danach kannst du das Programm kompilieren.
Zur Funktion:

```
PlaySound("Lied.wav",NULL,SND_FILENAME);
```

In Hochkommas steht der Pfad der Datei, die abgespielt werden soll. Danach steht immer Null, es sei denn du arbeitest mit Ressourcen. Am Schluß stehen die Parameter. Folgende Parameter gibt es:

SND_FILENAME : Braucht man immer, wenn man nicht mit Ressourcen arbeitet.
SND_LOOP: Der Sound wird ewig wiederholt.
SND_ASYNC: PlaySound() kehrt unmittelbar nach dem Starten des Sounds zurück.

So würde die Funktion aussehen, wenn das Lied ewig abgespielt werden soll:

```
PlaySound("Lied.wav",NULL,SND_FILENAME | SND_ASYNC | SND_LOOP);
```

Endlich Mp3

Nach einem guten halben Jahr Suche nach einer guten Möglichkeit Mp3s abzuspielen, bin ich fündig geworden. Das Schlüsselwort heißt MCI(Media Control Interface). Jetzt geht's aber los:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

int main(int argc, char *argv[])
{
char Kommando[500];
char Dateiname[]={"C:\\Lied.mp3"};

wsprintf(Kommando,"open \"%s\" alias mp3player
shareable",Dateiname);
mciSendString(Kommando,0,0,0);
mciSendString("set mp3player time format milliseconds",0,0,0);
mciSendString("play mp3player",0,0,0);
_sleep(5000);
mciSendString("pause mp3player",0,0,0);
_sleep(2000);
mciSendString("resume mp3player",0,0,0);

system("PAUSE");
mciSendString("stop mp3player",0,0,0);
```

```
mciSendString("close mp3player",0,0,0);

return 0;
}
```

Um dieses Programm ohne Fehler kompilieren zu können musst du die libwinmm.a einbinden. Wie das geht siehst du im vorherigen Kapitel. wsprintf schreibt den Befehl an das MCI in einen String. (Hier: Kommando) Mit mciSendString wird der Befehl an das MCI gesendet. Nun zu den einzelnen Kommandos. Das erste öffnet einen Stream zur Datei, die abgespielt werden soll und startet das MCI. Das zweite legt fest, dass die Position und die Länge in Millisekunden angezeigt werden soll. Eigentlich kann man diesen Befehl weglassen, aber wir brauchen ihn noch für später. Das letzte Kommando befiehlt dem Mci das Lied abzuspielen.Weitere Befehle sind:

pause: lässt das Lied pausieren
resume: setzt das Lied nach der Pause fort
stop: stoppt das Lied
close: schließt alles

Weitere Befehle und genauere Definitionen kannst du hier nachlesen: msdn2.microsoft.com/en-us/library/ms712587.aspx
Die Funktion _sleep(5000) fügt eine 5 sekündige Pause ein. Der Parameter muss in Milisekunden eingegeben werden. (5s=5000ms)

Jetzt gibt es noch ein paar Probleme! Wie kann man herausbekommen, ob das Lied noch abgespielt wird oder nicht und wie lange wurde das Lied bereits abgespielt? Diese Probleme werden im nächsten Beispiel gelöst:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

int Ende=0;
char LaengeMp3[500];

int PlayMp3(char *Dateiname)
{
char cmd[500];

wsprintf(cmd,"open \"%s\" alias mp3player shareable",Dateiname);
mciSendString(cmd,0,0,0);
mciSendString("set mp3player time format milliseconds",0,0,0);
mciSendString("play mp3player",0,0,0);

return 1;
}
int PlayPosition(char *Pos)
{
char cmd[500];
char retval[500];
int ms;
int sek, sek2;
int min;
int hours;
```

```
int sekk;
mciSendString("status mp3player position", retval, 500, NULL);
ms=atoi(retval); // wandelt den String zu Integer um
sek=ms/1000; // rechnet in Sekunden um
min=sek/60; // rechnet in Minuten um
hours=min/60; // rechnet in Stunden um
sek2=sek-(hours*60)-(min*60); //Sekunden, die übrig bleiben
wsprintf(cmd,"%i:%i:%i",hours,min,sek2);//schreibt die Position
geordnet in den String cmd
strcpy(Pos,cmd);

mciSendString("status mp3player length",retval, 500, 0);
ms=atoi(retval); //hier passiert das Gleiche, wie oben
sekk=ms/1000;
min=sekk/60;
hours=min/60;
sek2=sekk-(hours*60)-(min*60);

sprintf(cmd,"%i:%i:%i",hours,min,sek2);
strcpy(LaengeMp3,cmd);

if(sek==sekk)/*wenn die Sekundenanzahl der aktuellen Position und
die Sekundenzahl der Länge gleich sind, ist das Lied aus*/
{
Ende=1;
}

return Pos;// aktuelle Position wird zurückgegeben
return 1;
}

int main(int argc, char *argv[])
{
char Position[500];
char Name[1024];
char alte_Position[500];

printf("Mp3-Player:\n\n");
printf("Welches Lied wollen Sie abspielen: ");
scanf("%s",&Name);
getchar();

system("Cls");
PlayMp3(Name);
while(Ende==0)
{
PlayPosition(Position);
if(strcmp(alte_Position,Position)!=0)
{
system("Cls");
printf("Das Lied:\n%s\nwird abgespielt...\n\n",Name);
printf("Aktuelle Position: %s\n",Position);
printf("Laenge des Liedes: %s\n",LaengeMp3);
```

```
strcpy(alte_Position,Position);
}
}
mciSendString("close mp3player",0,0,0);

system("PAUSE");
return 0;
}
```

Ich hab hier eine eigene Funktion erstellt, die die Lieder abspielt. Die zweite Funktion ist dazu da die Länge des Liedes und die aktuelle Position zu ermitteln. Wenn die Länge und die Position übereinstimmen, ist das Lied zu Ende. Die wichtigsten zwei Zeilen sind:

mciSendString("status mp3player position", retval, 500, NULL);
mciSendString("status mp3player length",retval, 500, 0);

Die erste Funktion schreibt die Position in Form von Millisekunden in den String retval. Die 2te Funktion schreibt die Länge ebenfalls in Millisekunden in denselben String. Mittels atoi wird der String zu einem Integer konvertiert. Das Weitere ist nur Rechnerei. Wenn du bei diesem Beispiel etwas nicht verstehst, schreib mir ne Mail.

Die Tonaufnahme

Sie baut auf dem gleichen Schema auf, wie es das Abspielen der MP3-Dateien macht, das MCI. Deswegen muss man das MCI erst wieder starten und dann kann man die Befehle an dieses schicken, damit es eine Aufnahme macht. Diese Befehle unterscheiden sich ein bisschen von den Abspielbefehlen.Hier eine Liste der neuen Befehle:(Die Befehle pause,resume, stop und close gelten immer noch!)

MCI starten:
OPEN NEW TYPE WAVEAUDIO ALIAS AUSGEDACHTER_NAME //gibt dem Stream(Handle) einen Namen
set AUSGEDACHTER_NAME time format ms//alles wird in Millisekunden angegeben
set AUSGEDACHTER_NAME channels 1 // der 1. Kanal wird benützt
set AUSGEDACHTER_NAME bitspersample 8 //Definiert die Zahl der Bits pro PCM-Sample

Aufnahme starten:
```
record AUSGEDACHTER_NAME
```

Aufnahme in einer Datei speichern:
```
char cmd[500];
wsprintf(cmd,"save AUSGEDACHTER_NAME %s",Dateiname);
mciSendString(cmd,0,0,0);
```

Jetzt alles in einem Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

int start_new_record()
{
```

```
mciSendString("OPEN NEW TYPE WAVEAUDIO ALIAS
Audiorekorder",0,0,0);
mciSendString("set Audiorekorder time format ms",0,0,0);
mciSendString("set Audiorekorder channels 1",0,0,0);
mciSendString("set Audiorekorder bitspersample 8",0,0,0);
mciSendString("record Audiorekorder",0,0,0);
}
int pause_record()
{
mciSendString("pause Audiorekorder",0,0,0);
}
int resume_record()
{
mciSendString("resume Audiorekorder",0,0,0);
}
int save_record(char *Dateiname)
{
char cmd[500];
wsprintf(cmd,"save Audiorekorder %s",Dateiname);
mciSendString(cmd,0,0,0);
}
int close_record()
{
mciSendString("stop Audiorekorder",0,0,0);
mciSendString("close Audiorekorder",0,0,0);
}

int main(int argc, char *argv[])
{

char Datei[900];
int Wahl=0;

do
{
if(Wahl==0 ||Wahl==2)
{
printf("Wie soll die wav-Datei heissen, die Sie erstellen wollen:\
n");
scanf("%s",&Datei);
getchar();
system("Cls");
}
if(Wahl==0 || Wahl==1 || Wahl==2)
{
system("Cls");
printf("Wenn Sie die Aufnahme starten(fortfuehren) moechten,
druecken Sie Enter. Wenn Sie die Aufnahme stoppen oder beenden
wollen, druecken Sie noch einmal Enter.");
getch();
system("Cls");
printf("Die Aufnahme laeuft! Druecken Sie Enter um diese zu
stoppen.");
```

```
if(Wahl==0 || Wahl==2) start_new_record();
if(Wahl==1) resume_record();
getch();
pause_record();
}
system("Cls");

printf("Sie haben folgende Moeglichkeiten:\n\n");
printf("<1>Aufnahme fortsetzen\n");
printf("<2>Aufnahme neu starten\n");
printf("<3>Aufnahme speichern\n");
printf("<4>Aufnahme beenden ohne zu speichern(Programm wird
beendet)\n");
printf("<5>Programm beenden\n\n");

printf("Ihre Wahl: ");
scanf("%d",&Wahl);
getchar();

system("Cls");

if(Wahl==2 || Wahl==4 || Wahl==5) close_record();

if(Wahl==3)
{
save_record(Datei);
system("Cls");
printf("Die Aufnahme wurde gespeichert! Druecken Sie Enter um
zurueck ins Menu zu gelangen.");
getch();
}
}while(Wahl!=4 && Wahl!=5);

return 0;
}
```

Wie im Beispiel des MP3-Players muss man hier wieder die libwinmm.a einbinden.

Eine sehr gute Seite, auf der fast alle Befehle des MCI erklärt werden, ist:
http://www.directorforum.de/showthread.php?t=21

Die Welt von Midi

Du wolltest schon immer mal dein eigenes Orchester? Dann bist du mit Midi genau an das richtige Thema gestoßen. Übersetzt heißt midi Musical Instrument Digital Interface und noch einmal übersetzt heißt es: Digitale Schnittstelle für Musikinstrumente. Diese Schnittstelle wollen wir jetzt mal ausprobieren. Als erstes fangen wir mit dem Klavier an:

```
#include <stdio.h>
#include <stdlib.h>
#include <mmsystem.h>
```

```
int main(int argc, char *argv[])
{
int Lautstaerke=100;
int Ton=60;//C1
int Kanal=1;
int Instrument=1;//Klavier
long StartMidi;
long EndMidi;
long Sound;
int Tonlaenge=1;//(in Sekunden)
int Treibernummer=0;

HMIDIOUT hMidiOut;

midiOutOpen(&hMidiOut, Treibernummer, 0, 0, 0);

for(;Ton<73;Ton++)
{
StartMidi=(65536 * Lautstaerke) + (256 * Ton) + Kanal+143;
EndMidi=(65536 * Lautstaerke) + (256 * Ton) + Kanal+127;
Sound=(256 * Instrument) + Kanal+191;

midiOutShortMsg(hMidiOut, Sound);
midiOutShortMsg(hMidiOut, StartMidi);
sleep(Tonlaenge*1000);

midiOutShortMsg(hMidiOut, EndMidi); // MIDI aufräumen
}
midiOutReset(hMidiOut);
midiOutClose(hMidiOut);

system("PAUSE");
return 0;
}
```

Damit das Programm läuft muss man mal wieder die libwinmm.a einbinden(Projekt->Projekt Optionen->Parame...->Bibliothek/Objekt hinzuf.->Arbeitsplatz->C->Dev-Cpp->lib ->libwinmm.a). Aber danach solltest du die chromatische Tonleiter von C1 bis C2 hören.(Für alle Nicht-Musiker: Der Ton wird immer einen Halbton höher C->Cis->D...) Am Anfang des Programmes werden mehrere Variablen deklariert, die brauchen wir aber erst ein bisschen später im Programm. Zuerst kommt der neue Typ einer Variable: HMIDIOUT hMidiOut. Diese Variabletyp steht für die Wiedergabe des Midi-Sounds. Man kann sie auch mit File Variablen-Typ vergleichen. Bei der FILE Variable ist es so, dass wenn man mit der Datei arbeiten(schreiben/lesen) will, das mit der FILE-Variable machen muss. Genauso ist es mit der HMIDIOUT-Variable. Wenn man etwas hören will, muss man das mit dieser Variable machen. Da man zu der Datei erst einmal Verbindung(Stream) herstellt(fopen), muss man auch erstmal eine Verbindung zum MIDI herstellen. Das geht mit der Funktion midiOutOpen(). Der erste Parameter ist das "handle"(so nennt man eine solche Variable). Bei diesem handle steht am Anfang ein &-Zeichen. Das heißt die Information, die man zum abspielen benötigt, werden in das handle hineingeschrieben.. Jetzt kommt der Grund für die viele Variablen. Man muss eine Nachricht an das MIDI schicken, welchen Ton, welches Instrument, welche Lautstärke und auf welchem Kanal den Sound abgespielt werden soll. Diese Nachricht besteht aus einer Zahl, die man zuerst berechnen muss und das geht mit dieser Formel:

```
Nachricht1=(65536 * Lautstärke) + (256 * Ton) + Kanal+143
Nachricht2=(256 * Instrument) + Kanal+191
```

Wie du siehst, sind es eigentlich 2 Formeln. Das heißt, dass auch zwei Nachrichten verschickt werden. Diese verschickt man mit der Funktion midiOutShortMsg(). Wenn man genug von dem Ton bzw. dem Sound hat, schickt man dem MIDI wieder eine Nachricht. Diesmal braucht man nur eine Formel:

```
Nachricht3=(65536 * Lautstärke) + (256 * Ton) + Kanal+127
```

Wenn man die MIDI völlig beenden will, braucht man die beiden Befehle midiOutReset und midiOutClose. Die nächste Frage, die sich stellt, ist, wie man an andere Instrumente kommt. Man muss einfach der Variable Instrument eine andere Zahl zuweisen. Welche Zahl welches Instrument ist, siehst du auf dieser Website: www.pagus.de. Eine Liste aller Noten und deren Zahlen findest du hier: www.sengpielaudio.com

Netzwerkprogrammierung

Erst einmal zu der Frage: "Was ist Netzwerkprogrammierung, oder besser was ist ein Netzwerk?" Hier die Definition: Verbindung zweier oder mehrerer Computer über elektrische Leitungen. Über diese Verbindung können Daten, aller Art (z.B. E-Mails oder Homepages) zwischen Rechnern auf der ganzen Welt übertragen werden. Diese Kommunikation zwischen den Rechnern kann man programmieren. Das nennt sich dann Netzwerkprogrammierung. Zuerst bauen wir einfach eine Verbindung zu dem Rechner in unserem Fall zum Server auf:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

int main(int argc, char *argv[])
{
int Socket;
int Laenge;
int Verbindung;
struct sockaddr_in Adressinfo;

int Port = 80;
char Ip_Adresse[80]={"213.133.106.12"};//Ip des c-
programmieren.com Servers

WSADATA wsaData;
if(WSAStartup(MAKEWORD(1,1), &wsaData) != 0)
{
printf("WSA konnte nicht initialisiert werden.\n\n");
system("PAUSE");
return 0;
}

Socket = socket(AF_INET, SOCK_STREAM, 0);
Adressinfo.sin_family = AF_INET;
Adressinfo.sin_addr.s_addr = inet_addr(Ip_Adresse);
Adressinfo.sin_port = htons(Port);
```

```
Laenge = sizeof(Adressinfo);

Verbindung = connect(Socket, (struct sockaddr *)&Adressinfo,
Laenge);
if (Verbindung == 0)
{
printf("Verbindungsaufbau erfolgreich!");
}
else
{
printf("Verbindungsaufbau ist fehlgeschlagen!");
}

close(Socket);

printf("\n\n");
WSACleanup();

system("PAUSE");
return 0;
}
```

Die Funktion WSAStartup initialisiert Winsock,damit du Sockets benutzen kannst. Gleich zur nächsten Funktion: socket(). Sie erstellt einen Socket.Der erste Parameter dieser Funktion ist das Ip-Protokoll in unserem Fall AF_INET, das heißt Internet IP-Protokoll Version 4. Der zweite Parameter ist die Übertragungsart z.B. SOCK_STREAM,TCP, UDP... Der dritte Parameter ist noch für spezielle Anwendugen, aber wenn man im Standard programmiert, heißt der dritte Parameter immer 0. Im folgenden wird der Struktur die Info der Verbindung übergeben, d.h. IP-Protokoll, Ip, Port. Danach wird die Verbindung mit der Funktion connect erstellt. Hier ist der erste Parameter, der Socket, der bereits erstellt wurde. Der zweite Parameter ist die Info über die Verbindung, die in einer Struktur steht und der dritte Parameter ist die Größe der Info bzw. der Struktur. Um eine Verbindung wieder zu trennen, braucht man die Funktion close(). Hier ist der einzige Parameter der Socket, der beendet werden soll. Danach wird Winsock mit WSACleanup() deinitialisiert. Damit dieses Programm funktiert, muss man außerdem noch libws2_32.a einbinden. (Projekt->Projekt Optionen->Parame...->Bibliothek/Objekt hinzuf.->Arbeitsplatz->C->Dev-Cpp->lib->libws2_32.a)

Jetzt kannst du eine Verbindung mit einem Server aufbauen. Jetzt kommt die Kommuniktion mit dem Server. Als Beispiel programmieren wir einen Browser.

```
#include <windows.h>
#include <winsock.h>
#include <stdio.h>

int main()
{
SOCKET Socket;
SOCKADDR_IN Adressinfo;
char Antwort[1024];
LPHOSTENT lpHostEntry;
long Verbindung;
WSADATA wsa;

char Nachricht[256];
```

```
sprintf(Nachricht, "GET /index.html HTTP/1.0\nHost: www.c-
programmieren.com\n\n" );

// Winsock starten

Verbindung=WSAStartup(MAKEWORD(2,0),&wsa);
if (Verbindung != NO_ERROR)
{
printf("WSAStartup ist fehlgeschlagen: %d\r\n", Verbindung);
}

lpHostEntry = gethostbyname("c-programmieren.com");//Ip von c-
programmieren.com wird gesucht

// Socket erstellen
Socket=socket(AF_INET,SOCK_STREAM,0);

memset(&Adressinfo,0,sizeof(SOCKADDR_IN)); // zuerst alles auf 0
setzten

Adressinfo.sin_family=AF_INET;
Adressinfo.sin_port=htons(80); // HTTP = 80
Adressinfo.sin_addr = *((LPIN_ADDR)*lpHostEntry-
>h_addr_list);//die gefundene Ip wird eingesetzt

Verbindung=connect(Socket,
(SOCKADDR*)&Adressinfo,sizeof(SOCKADDR));
if ( Verbindung == SOCKET_ERROR)
{
printf("Verbindung ist fehlgeschlagen!\n");
system("Pause");
closesocket(Socket);
WSACleanup();
}

Verbindung=send(Socket,Nachricht,strlen(Nachricht),0);
if (Verbindung == SOCKET_ERROR)
{
printf("Die Nachricht konnte nicht verschickt werden: %d\r\n",
WSAGetLastError());
system("Pause");
closesocket(Socket);
WSACleanup();
return 0;
}

<
else
{
printf("Die Nachricht wurde verschickt!\n",Nachricht);
}

while(Verbindung!=SOCKET_ERROR)
```

```
{
Verbindung=recv(Socket,Antwort,1023,0);
if(Verbindung==0)
{
printf("Server hat die Verbindung getrennt..\n");
break;
}
if(Verbindung==SOCKET_ERROR)
{
printf("Fehler bei Empfangen: %d\n",WSAGetLastError());
break;
}
Antwort[Verbindung]='\0';
printf("\nAntwort:\n\n %s\n\n\n",Antwort);
}
closesocket(Socket);
WSACleanup();
system("Pause");
return 0;
}
```

Zuerst läuft alles wie gehabt. Ich weiss nicht, ob ich schon die Funktion sprintf erklärt habe, aber mit dieser Funktion kann man in einen String schreiben. Der erste Parameter ist der Name des Strings, der zweite ist das, was in den String kommen soll.Aber spätestens bei der Funktion gethostbyname fangen die Neuerungen an. Diese Funktion wandelt einen Servernamen zur Ip-Adresse des Servers um. Aber deswegen ändert sich auch die Zuweisung der Ip-Adresse. Sie schaut dann so aus:

```
Adressinfo.sin_addr = *((LPIN_ADDR)*lpHostEntry->h_addr_list);
```

Mit der Funktion memset wird die Struktur davor aber gereinigt. Das heißt alle Werte werden auf Null gesetzt. Der erste Parameter von memset ist die Variable, mit der du arbeiten willst, die zweite welches Zeichen du in die Variable einfügen willst und die dritte ist die Länge der Variable, die du mit dem Zeichen füllen willst. Die nächste neue Funktion heißt send. Sie schickt die Nachrichten an den Server. Der erste Parameter ist der Socketname, der zweite die Nachricht und der dritte die Länge der Nachricht. Der vierte ist immer 0. Schließlich zur Schleife: Verbindung != SOCKET_ERROR bedeutet solange die Verbindung besteht, also keine Fehler vorgekommmen sind, sollen in unserem Fall die Antworten des Servers empfangen werden. Wie du siehst, ist für das Empfangen der Daten die Funktion recv zuständlich. Die Parameter entsprechen den Parametern der Funktion send. Das war's schon. Schon habt ihr den Quelltext meiner Website. Ihr könnt die Url natürlich ändern. Zum Schluss: Du musst natürlich libws2_32.a wieder einbinden.(Siehe oben)

Maus, Tasten und Bildschirm

Auflösung des Bildschirms ermitteln

Die Auflösung zu ermitteln heißt die Länge und die Breite des Bildschirms in Pixel zu suchen. Zuerst das Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <windows.h>

int main(int argc, char *argv[])
{
int Laenge = GetSystemMetrics(SM_CXSCREEN);//Größe des Bildschirms
nach links(in Pixel)
int Breite = GetSystemMetrics(SM_CYSCREEN);//Größe des Bildschirms
nach unten(in Pixel)

printf("Bildschirmlaenge: %d\n",Laenge);
printf("Bildschirmbreite: %d\n\n",Breite);

system("PAUSE");
return 0;
}
```

Wie du siehst, ist GetSystemMetrics die Lösung. Mit dieser Funktion kann man mehrere Größen des System ermitteln und wenn man den Parameter SM_CXSCREEN einsetzt, erhält man die Länge des Bildschirms in Pixel, bei SM_CYSCREEN die Breite(Höhe). Wenn du noch mehr Parameter kennenlernen willst: msdn2.microsoft.com/en-us/library/ms724385.aspx

Die Maus bewegen

Wie für vieles gibt es auch hier eine Funktion. Sie heißt SetCursorPos. Sie ist so aufgebaut:

```
SetCursorPos(x,y);
```

Das heißt, wenn x=0, dann ist die Maus ganz links auf dem Bildschirm und wenn y=0, dann ist die Maus ganz oben. Umso größer x, desto weiter rechts ist die Maus und umso größer y, desto weiter unten befindet sich die selbe. Jetzt aber das Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <time.h>
int main(int argc, char *argv[])
{
int i;

int x_Zufall;
int y_Zufall;

int x_Bildschirm = GetSystemMetrics(SM_CXSCREEN);//Groesse des
Bildschirms nach links(in Pixel)
int y_Bildschirm = GetSystemMetrics(SM_CYSCREEN);//Groesse des
Bildschirms nach unten(in Pixel)

srand(time(0));

printf("Ein kleines Scherzprogramm!\n\n");

for(i=0;i<20;i++)
```

```
{
sleep(100);
x_Zufall=(rand() % x_Bildschirm + 1);
y_Zufall=(rand() % y_Bildschirm + 1);
SetCursorPos(x_Zufall,y_Zufall);
}

system("PAUSE");
return 0;
}
```

In diesem Programm wird zuerst festgelegt in welchem Bereich sich die Maus bewegen kann. Es wird also die Bildschirmauflösung ermittelt. Danach wird die Maus 20 Mal auf einen zufälligen Punkt auf den Bildschirm gesetzt.

Mausklick simulieren

Beim Mausklick gibt es etwas besonderes. Der Mausklick besteht nicht aus einer Aktion(das Klicken), wie man eigentlich vermutet, sondern aus zwei, dem "Drauddrücken" und dem Loslassen. Jetzt wird geklickt:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

int main(int argc, char *argv[])
{
int Startbuttonx=20;//nach links
int Startbuttony=GetSystemMetrics(SM_CYSCREEN)-20;//nach unten

SetCursorPos(Startbuttonx,Startbuttony);//setzt die Position der
Maus
mouse_event(MOUSEEVENTF_LEFTDOWN,Startbuttonx,Startbuttony,0,0);//
Mausklick starten
mouse_event(MOUSEEVENTF_LEFTUP,Startbuttonx,Startbuttony,0,0);//Ma
usklick beenden

system("PAUSE");
return 0;
}
```

Die neue Funktion heißt mouse_event. Der erste Parameter ist die Aktion, wie hier im Beispiel MOUSEEVENTF_LEFTDOWN = Linksklick-gedrückt und MOUSEEVENTF_LEFTUP = Linksklick-loslassen. Man kann die Maus auch mit mouse_event bewegen. Dann heißt der Parameter MOUSEEVENTF_MOVE. Die folgenden zwei Parameter geben den Punkt an, auf dem die Aktion stattfinden soll. Die letzten zwei Parameter sind für uns unwichtig. Wenn du mehr über diese Funktion wissen willst, kannst du wieder einmal bei msdn nachschauen. Das Ergebnis des Programmes sollte es übrigens sein, dass die Maus auf den Startbutton klickt. Wenn dies nicht der Fall sein sollte, musst du die Koordinaten ein bisschen verändern.

Wo ist die Maus?

Das ist z.B. wichtig für die Simultion eines Mausklicks, damit man weiß, wie die Koordinaten z.B. für einen Klick genau heißen. Sonst kann das Probieren, wie die Koordinaten lauten könnten ziemlich nervig sein. Hier kommt meine Lösung:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

int main(int argc, char *argv[])
{
POINT Punkt;

while(1)
{
sleep(1);
system("CLS");
GetCursorPos(&Punkt);
printf("x=%d\n",Punkt.x);
printf("y=%d\n",Punkt.y);
}

return 0;
}
```

Mit der Funktion GetCursorPos kann man die Koordinaten der Maus in die Struktur Point schreiben. Sie besteht aus zwei Variablen x und y. Das war's eigentlich auch schon.

Tasten simulieren

Hier heißt das Zauberwort keybd_event. Die Funktion ist ähnlich aufgebaut wie mouse_event. Man kann mit keybd_event sogar einen Mausklick simulieren. Der erste Parameter der Funktion ist die Taste, die gedrückt werden soll. Jede Taste besitzt einen "virtual-key code". Dieser muss als Parameter angegeben werden. Der zweite Parameter kann der dazugehörige Scancode sein. Diesen muss man aber nicht angeben. Deswegen ist der zweite Parameter bei uns immer 0. Der dritte Parameter ist die Aktion. Das heißt entweder "Taste hinunterdrücken", dann ist der Wert 0 oder "Taste Loslassen", dann muss man KEYEVENTF_KEYUP einsetzen. Danach kommt noch ein Parameter, der ebenfalls 0 ist. Das erste Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

int main(int argc, char *argv[])
{
int i;
char Text[900]={"hallo"};//dieser Text wird simuliert

ShellExecute(NULL, "open", "notepad.exe", NULL,
"C:\\Windows\\system32", SW_MAXIMIZE);
sleep(1000);
```

```
for(i=0;i<strlen(Text);i++)
{
keybd_event(VkKeyScan(Text[i]),0,0,0);//Tastendruck
keybd_event(VkKeyScan(Text[i]),0,KEYEVENTF_KEYUP,0);//Loslassen
}

return 0;
}
```

Du hast dich vielleicht schon zuvor gefragt, wie man jetzt auf den "virtual-key code" einer Taste kommt. Entweder durch ausprobieren oder man geht auf die Website von msdn oder die beste Lösung zum Schluss, man setzt einfach die Funktion VkKeyScan ein um eine char Variable in virtual-key code umzuwandeln. Jetzt gibt es aber ein Problem, da z.B. a und A auf der gleichen Taste sitzen. Das heißt der virtual-key code für a ist gleich dem für A. Deswegen muss man überprüfen, ob es sich um einen kleingeschrieben Buchstaben oder um einen großen handelt. Wenn es sich um einen großen Buchstaben handelt, muss man den Tastendruck der Shifttaste(Großumschalttaste) simulieren. Wie man alle Zeichen simulieren kann, siehst du in diesem Beispiel:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

int Schreib(char Text[900])
{
int i;
int o;
int Scancode;
int fertig;

char Achtung[]={"!\"§$%&/()=?*'_:;>°ÄÜÖ"};
char Achtung2[]={"@|²³€{[]}~µ"};

for(o=0;o<strlen(Text);o++)
{
fertig=0;
Scancode=(int)Text[o];

for(i=0;i<strlen(Achtung);i++)
{
if(Text[o]==Achtung[i])
{
keybd_event(16,0,0,0);//Shifttaste wird gedrückt
keybd_event(VkKeyScan(Text[o]),0,0,0);//Tastendruck
keybd_event(VkKeyScan(Text[o]),0,KEYEVENTF_KEYUP,0);
keybd_event(16,0,KEYEVENTF_KEYUP,0);//Shifttaste wird losgelassen
fertig=1;
break;
}
}
for(i=0;i<strlen(Achtung2);i++)
{
if(Text[o]==Achtung2[i])
```

```
{
keybd_event(17,0,0,0);//Strg-Taste
keybd_event(VK_MENU,0,0,0);//Alt-Taste
keybd_event(VkKeyScan(Text[o]),0,0,0);//Tastendruck
keybd_event(VkKeyScan(Text[o]),0,KEYEVENTF_KEYUP,0);
keybd_event(17,0,KEYEVENTF_KEYUP,0);
keybd_event(VK_MENU,0,KEYEVENTF_KEYUP,0);
fertig=1;
break;
}
}

if(Scancode>64 && Scancode<91)//ABC
{
keybd_event(16,0,0,0);//Shifttaste wird gedrückt
keybd_event(VkKeyScan(Text[o]),0,0,0);//Tastendruck
keybd_event(VkKeyScan(Text[o]),0,KEYEVENTF_KEYUP,0);
keybd_event(16,0,KEYEVENTF_KEYUP,0);
fertig=1;
}

if(fertig==0) //Rest, der mit VkKeyScan alleine machbar ist
{
keybd_event(VkKeyScan(Text[o]),0,0,0);//Tastendruck
keybd_event(VkKeyScan(Text[o]),0,KEYEVENTF_KEYUP,0);
}
}

return 0;
}

int main()
{
char Text[900]={"Das ist ein Test: Hallo \"[{@}]\" Leute wie
geht's?1234 Umlaute gehen auch !: ÄÜÖäüöß\n"};//dieser Text wird
simuliert

ShellExecute(NULL, "open", "notepad.exe", NULL,
"C:\\Windows\\system32", SW_MAXIMIZE);//Editor wird geöffnet
sleep(1000);
Schreib(Text);

return 0;
}
```

In der Funktion "Schreib" wird jeder Buchstabe überprüft, ob er ein Zeichen ist, das nur mit der Shifttaste oder zusammen mit Strg und Alt erstellt werden kann. Je nachdem werden die Tasten simuliert. Du fragst dich vielleicht, was eigentlich ein Scancode ist. Jedes Zeichen hat einen Scancode. Dieses Mal unterschiedet sich der Scancode von a, von dem von A. Mit diesem Programm kannst du eine Liste machen, welche Zahl(Scancode) welchem Zeichen entspricht:

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main(int argc, char *argv[])
{
int i;

printf("Scancode: Zeichen:\n");
for(i=0;i<256;i++)
{
printf("%3d %c\n",i,i);
}
system("PAUSE");
return 0;
}
```

Jetzt weist du, warum die Zeichen von 65 bis 90 mit der Shifttaste simuliert werden müssen. Das sind nämlich die Großbuchstaben. Der Scancode ist auch der Grund dafür, dass man mit Buchstaben rechnen kann. A+B wäre dann 131.

Vista und Dev-Cpp

Als ich neulich einen neuen Computer bekommen habe, wollt ich ein ganz leichtes Programm kompilieren. Erschreckender Weise kam diese Fehlermeldung:

Makefile.win [Build Error] [Datei.o] Error 1

Nachdem ich den Dev-Cpp 2 oder 3 mal deinstalliert und wieder installiert habe und es immer noch nicht funktioniert hat, war ich geschockt. Gott sei Dank, dass ich diese Website gefunden habe, auf der alles erklärt ist, wie man den Compiler konfigurieren muss, damit er läuft. Das Forum, in dem dieser Beitrag gemacht wurde heißt www.et-forum.org. Der Beitrag war von Nigggo & Omega86 und lautete:

Um unter Windows Vista mit Dev Cpp Programme compilieren zu können, müssen nach der Installation noch ein paar Einstellungen geändert werden:

1) Systemsteuerung/System/Erweiterte Systemeinstellungen
dort auf "Umgebungsvariablen", dann unter PATH den Pfad "C:\Dev-cpp\libexec\gcc\mingw32\3.4.2" zu den vorigen Variablen hinzufügen (wenn ihr Dev Cpp im Standardverzeichnis installiert habt)

2)Um das Prog. richtig ausführen zu können, müsst ihr auf die Datei devcpp.exe rechts klicken, dann \Eigenschaften, Registerkarte "Sicherheit", dann auf bearbeiten, dann Benutzer(unter dem ihr DevCpp verwenden wollt) auswählen und Vollzugriff anklicken!

3)Im Programm Dev Cpp unter "Tools\Compiler Options\programs" für jeden Punkt den vollen Pfad angeben.(z.B.: C:\Dev-cpp\bin\gcc.exe)

nach diesen Änderungen hat Dev Cpp bei uns einwandfrei funktioniert(auf 2 PCs getestet)
Wir hoffen, dass euch diese Anleitung eine Hilfe ist.

Lg Nigggo & Omega86

An dieser Stelle möchte ich mich noch einmal herzlich bei Nigggo & Omega86 bedanken. Die Anleitung hat mir sehr weiter geholfen.

Der erste Schritt zur Fensterprogrammierung - MessageBox

Zuerst das Beispiel, dann die Arbeit:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

int main(int argc, char *argv[])
{
int Antwort= MessageBox(NULL,"Ist dieses Programm das Beste, das
du je programmiert hast?","Ein schwere Frage:",MB_ICONQUESTION |
MB_YESNO);

if(IDYES==Antwort)
{
printf("Glaub ich dir net*g*.");
}
if(IDNO==Antwort)
{
printf("Stimmt! Das Programm hab naemlich ich programmiert*g*.");
}
printf("\n\n");

system("PAUSE");
return 0;
}
```

Der Funktion MessageBox kann man 4 Parameter übergeben. Der erste ist das Fenster, aber da wir noch keines besitzen, ist dieser Parameter bei uns NULL. Der zweite Paramter ist der Text in der MessageBox und der dritte der Titel. Der vierte Parameter ist zur Bestimmung der Eigenschaften. Unsere MessageBox soll ein Fragezeichenicon besitzen = MB_ICONQUESTION und einen Ja- und einen Nein-Button = MB_YESNO. Wenn die Antwort des Benutzers Ja war, ist die Rückgabe der Funktion MessageBox gleich IDYES. Wenn er/sie auf Nein geklickt hat, ist sie gleich IDNO. Weitere Eigenschaften, Buttons und Icons der MessageBox kannst du wieder auf MSDN finden.

Literaturempfehlung:

C von Jürgen Wolf im Verlag Markt+Technik erschienen. ISBN 3-8272-4064-6 für 16,95€ für Einsteiger, wie ich einer bin oder war, ideal. Mir hat dieses Buch einen idealen Einstieg für C ermöglicht.