Praktikum Programmierung INF
Tutorial zum Praktikum WS2008/09

Martin Heinzerling - tud@mheinzerling.de

18. Januar 2009

Technische Universität Dresden

Zusammenfassung

Das hier Angebote Material ist kein Originalmaterial des Professors. Es wurde nach besten Wissen und Gewissen durch meine Person zusammengestellt. Aus offensichtlichen Gründen kann ich aber dennoch keine Garantie auf Vollständigkeit oder Korrektheit geben. Die Verwendung erfolgt auf eigene Gefahr.
Das Script darf ausschließlich privat genutzt werden. Öffentliche Präsentationen außerhalb dieses Studiengangs an der TUD sind strikt untersagt.
Zuwiderhandlung werden zur Anzeige gebracht.

Ich rate dringend davon ab, die Lösungen bloß abzuschreiben. Ein Verständnis der Thematik ist absolut notwendig für die Prüfung. Viele unterschätzen AuD und Programmieren und machen die Prüfung zweimal. Also immer alles selber machen und hier nur kontrollieren!!

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Allgemeines

Die aktuellen Aufgaben finden Sie immer unter:

`http://www.orchid.inf.tu-dresden.de/gdp/lehre/Praktikum/`

Dieses Dokument sowie weitere Hinweise zum Praktikum finden Sie unter:

`http://www.mheinzerling.de`

Bei Probleme, Fragen und Verbesserungsvorschläge erreichen Sie mich über:

`tud@mheinzerling.de` oder `#ICQ:141840546`

Im Text sind **Schaltflächen**, *Menüeinträge*, `Texteingabe`, Tasten auf der TASTATUR sowie andere *Bezeichner* gesondert gekennzeichnet.

Bilder habe ich auf eine mäßige Qualität reduziert um die Dateigröße gering zu halten. Aber man sollte das nötige erkennen.

Wer die Beispiele mit dem Visual Studio nachvollziehen will, bekommt dieses als Student der TU Dresden unter

`http://msdn40.e-academy.com/elms/Storefront/Storefront.aspx?campus=tu_dresden_inf`

Über die Seite können Sie ihr Login erfragen. Wer es weniger groß mag, dem empfehle ich

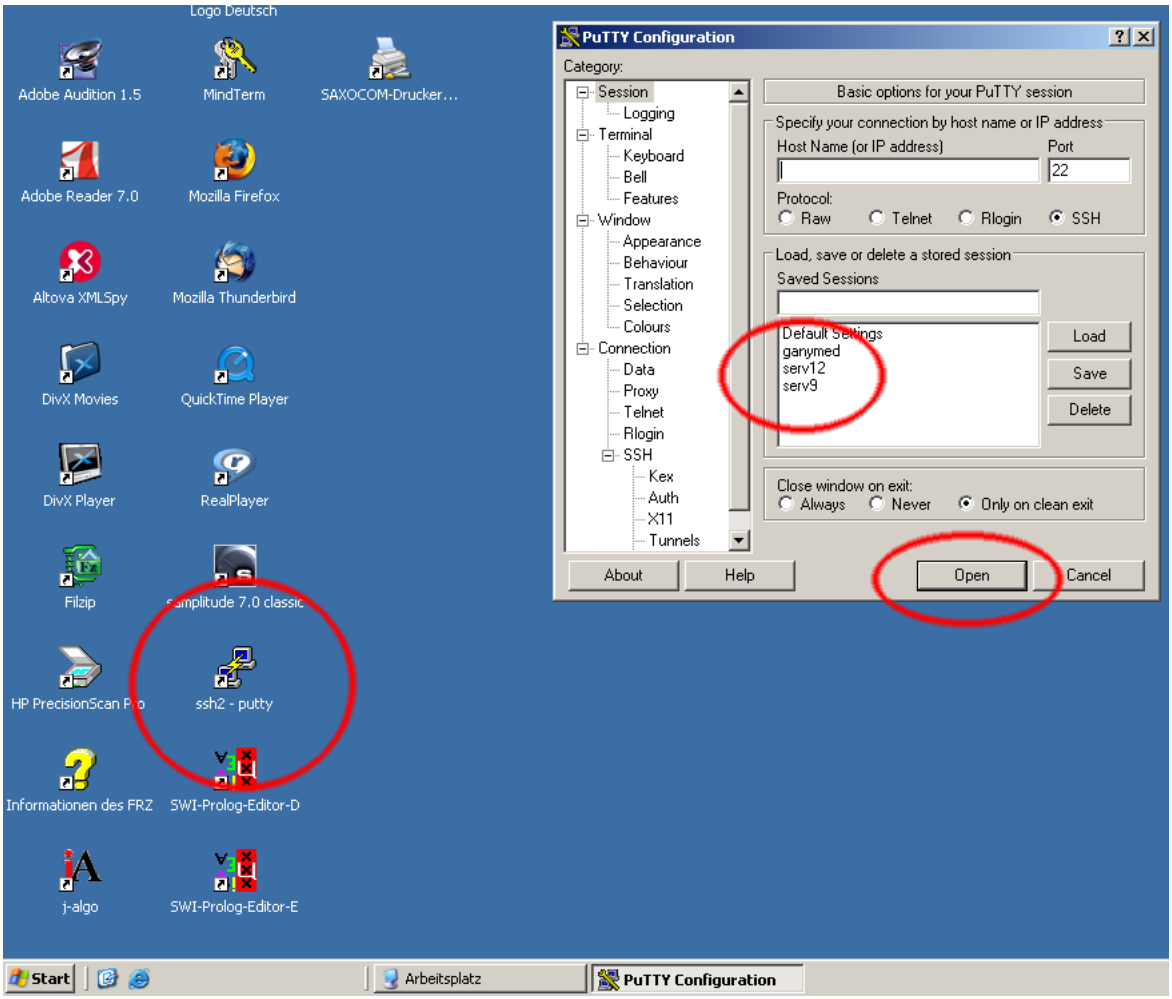
`http://www.bloodshed.net/devcpp.html`

P.S.: Das ich zwischen „Sie“, „ihr“ und „wir“ wechsle hat keine tiefere Semantik ;)

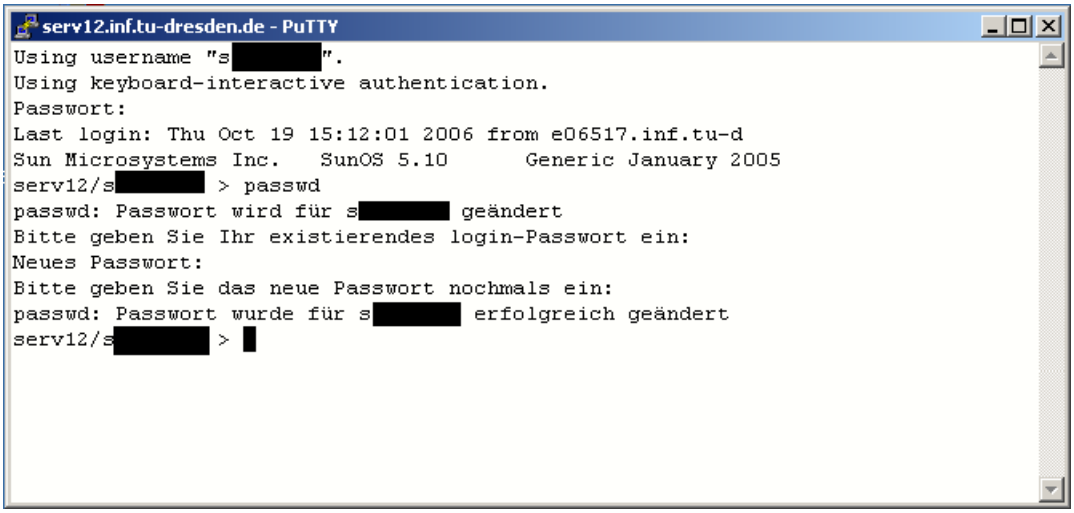
1.2 Passwort ändern

Bevor wir beginnen sollten Sie ihr Standardpasswort ändern. ACHTUNG das neue Passwort gilt nur im FRZ, im URZ bleibt das altere weiterhin gültig.

Um das Passwort zu ändern klicken Sie doppelt auf die Verknüpfung **ssh2-putty** auf ihrem *Desktop*. Danach erscheint der Dialog *PuTTy Configuration*. In diesem wählen Sie in der Sessionliste *serv12* oder *serv9* aus, und klicken anschließend auf **Open**.



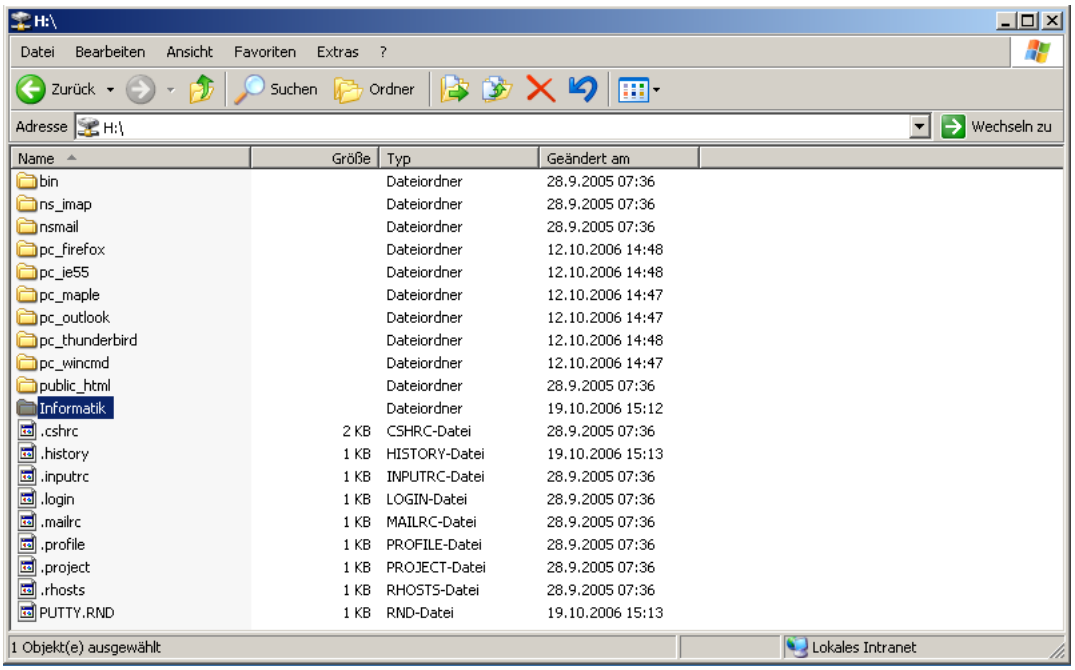
Darauf hin erscheint eine Konsole in der Sie aufgefordert werden ihr aktuelle Passwort einzugeben. Es erscheinen dabei keine Sterne wie unter Windows. Das Passwort wird blind eingegeben. Nach erfolgreicher Anmeldung können Sie mit dem Befehl `passwd` und dem anschließendem drücken der ENTERTASTE ihr Passwort ändern. Dazu werden Sie erneut zur Eingabe ihres Passwortes aufgefordert, sowie zur zweimaligen Eingabe ihres neuen Passwortes, diese bestätigen Sie jeweils mit ENTER.



Nach Wiedererscheinen der Eingabeaufforderung können Sie das Fenster mit `exit` und Enter schließen.

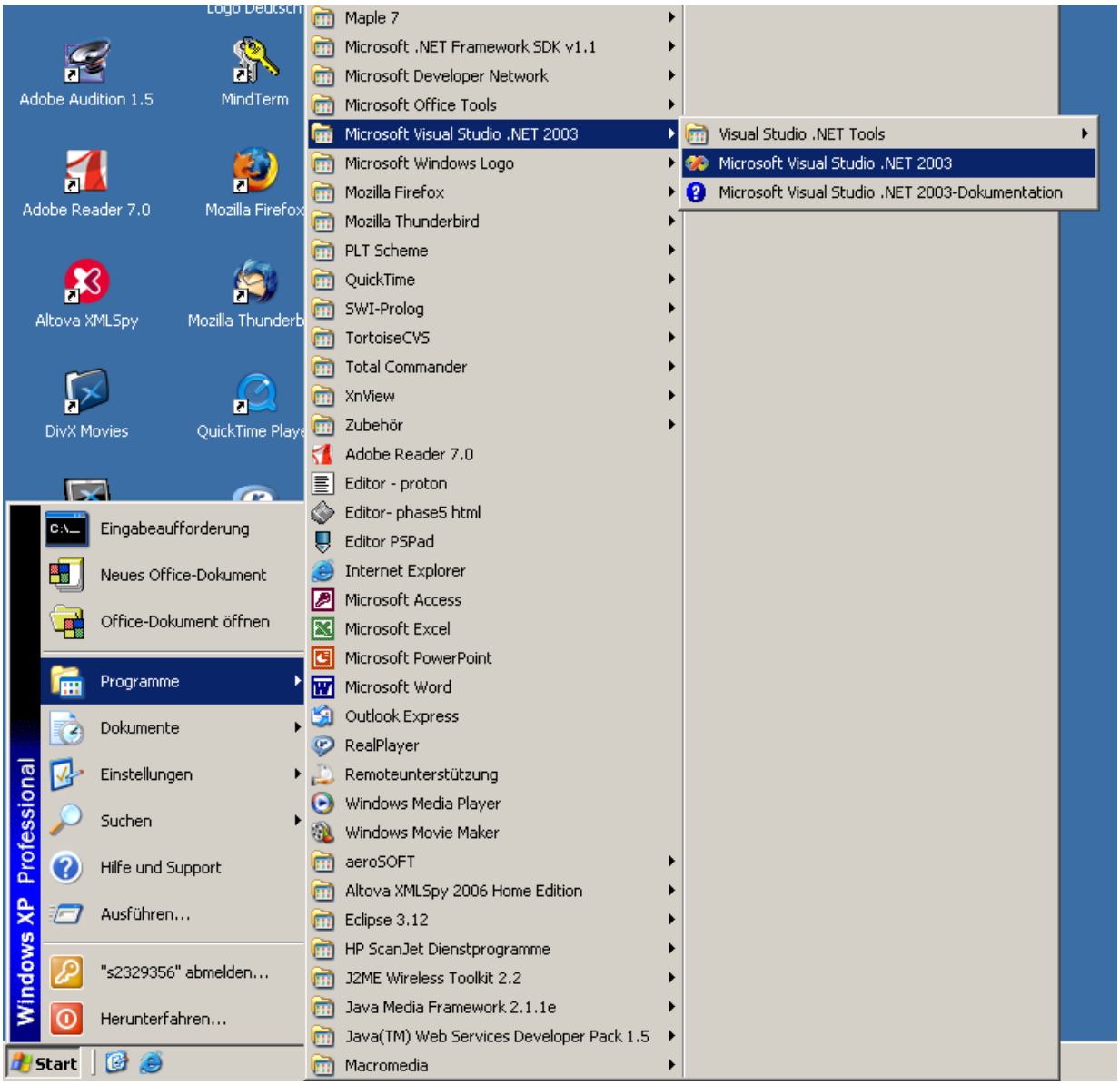
1.3 Benutzerverzeichnis

Mit der Anmeldung im FRZ wurde Ihnen auch ein eigenes Laufwerk *H:* für ihre Daten erstellt. Dieses können Sie mit dem *Arbeitsplatz* oder *Windows Explorer* öffnen. In *H:* erstellen Sie dann einen Ordner *Informatik* (*Datei* → *Neu* → *Ordner*) in dem alle zukünftigen Projekte erstellt werden.



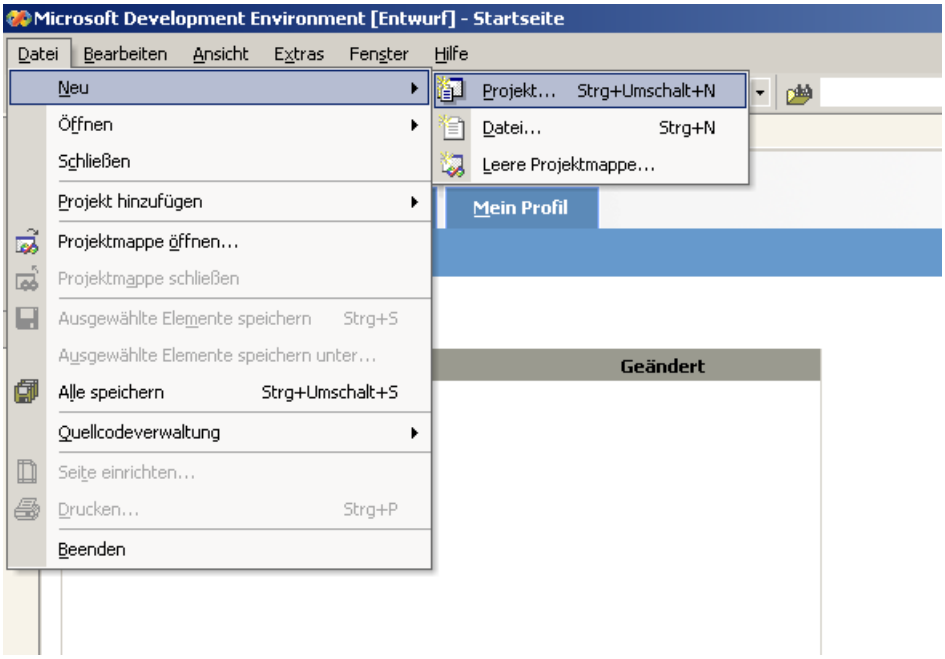
1.4 „Hello World“

Beginnen wir mit einem einfachen und dem Standard-Beispiel. Öffnen Sie zunächst das *Microsoft Visual Studio .NET 2003* indem Sie auf **Start** in der Taskleiste klicken. Wählen Sie dann *Programme*, *Microsoft Visual Studio .NET 2003* und nochmals *Microsoft Visual Studio .NET 2003*. [1]

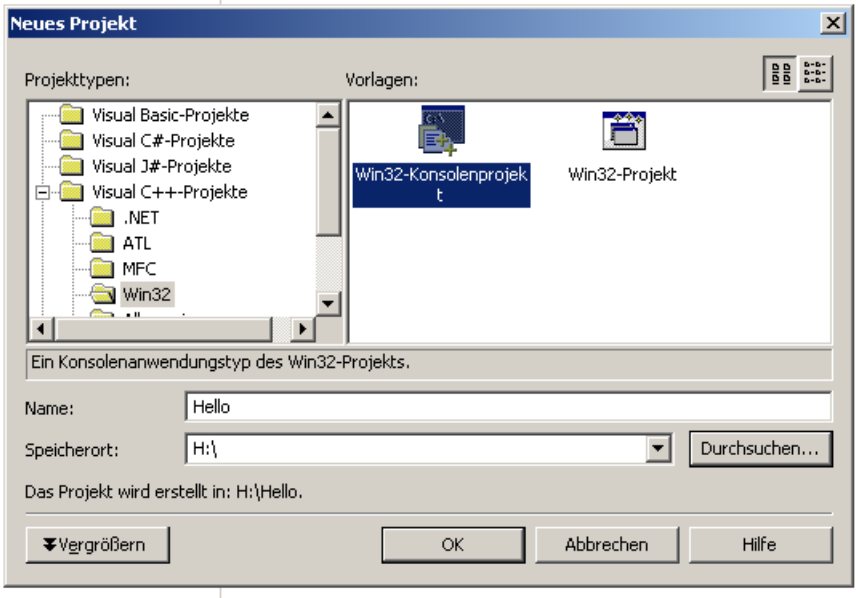


Sobald die Entwicklungsumgebung gestartet ist, erstellen Sie über *Datei*, *Neu* und *Projekt...* ein neues Projekt.

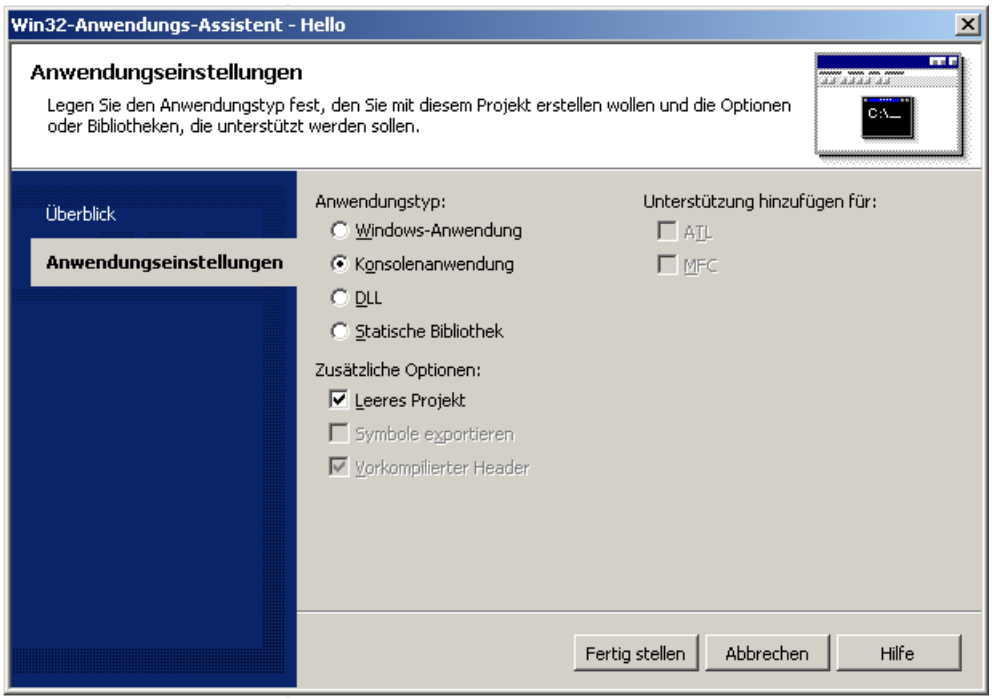
[1] Bei MS Visual Studio 2005 sind die Schritte ähnlich.



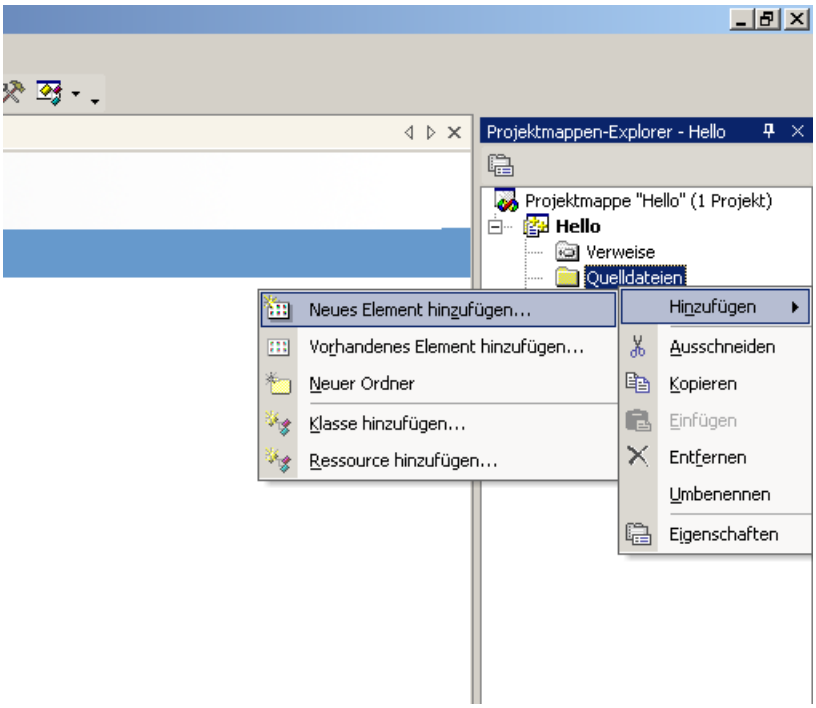
Im nächsten Dialog wählen Sie im Feld *Projekttypen* **Visual C++-Projekte** und **Win32** aus. Darauf hin ändert sich die Auswahl der *Vorlagenliste* und dort wählen Sie dann **Win32-Konsolenprojekt**. Im Feld *Name* geben Sie `Hallo` ein, und wählen als Speicherort den zuvor erstellten Ordner `H:\Informatik`. Abschließen klicken Sie auf **OK** und kommen zu einem weiteren Dialog.



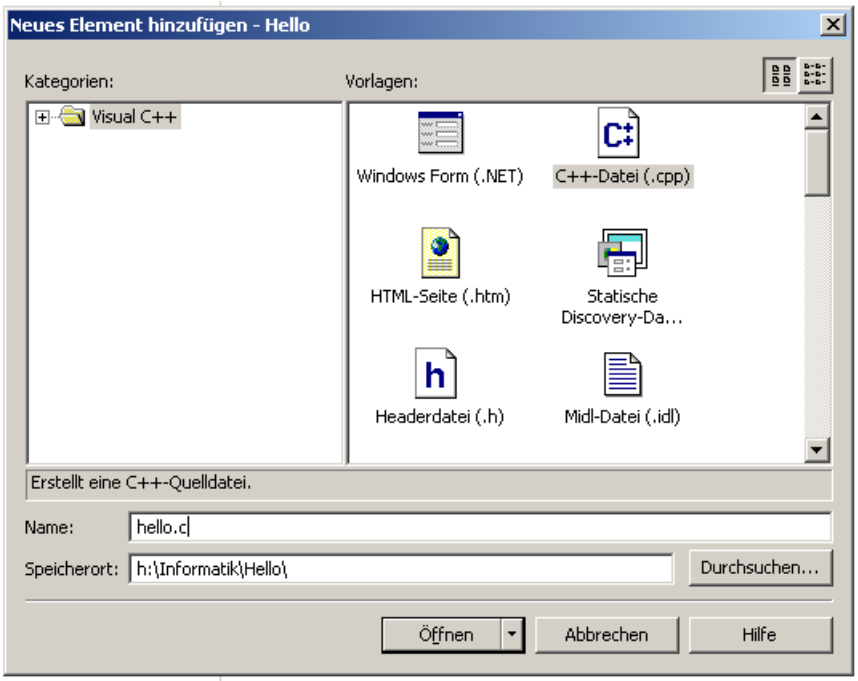
Dort wählen Sie in der linken Liste *Anwendungseinstellungen* aus. Setzen Sie nun noch einen Haken in *Leeres Projekt* und klicken dann auf **Fertigstellen**.



Um nun auch arbeiten zu können benötigen wir noch ein Quelldatei in unserem Projekt. Klicken Sie im *Projektmappen-Explorer* rechts auf *Quelldateien*. Wählen Sie dann *Hinzufügen* und *Neues Element hinzufügen*.(Sollte der *Projektmappen-Explorer* nicht sichtbar sein, können Sie diesen über *Ansicht* und *Projektmappen-Explorer* sichtbar machen.)



Im nächsten Dialog wählen Sie in den Vorlagen **C++-Datei(.cpp)** aus und geben als Namen `hello.c` ein. (Da wir nur eine C-Anwendung erstellen wollen und keine C++, erstellen wir also eine Datei mit der Endung .c statt .cpp). Danach wieder mit **Öffnen** bestätigen.

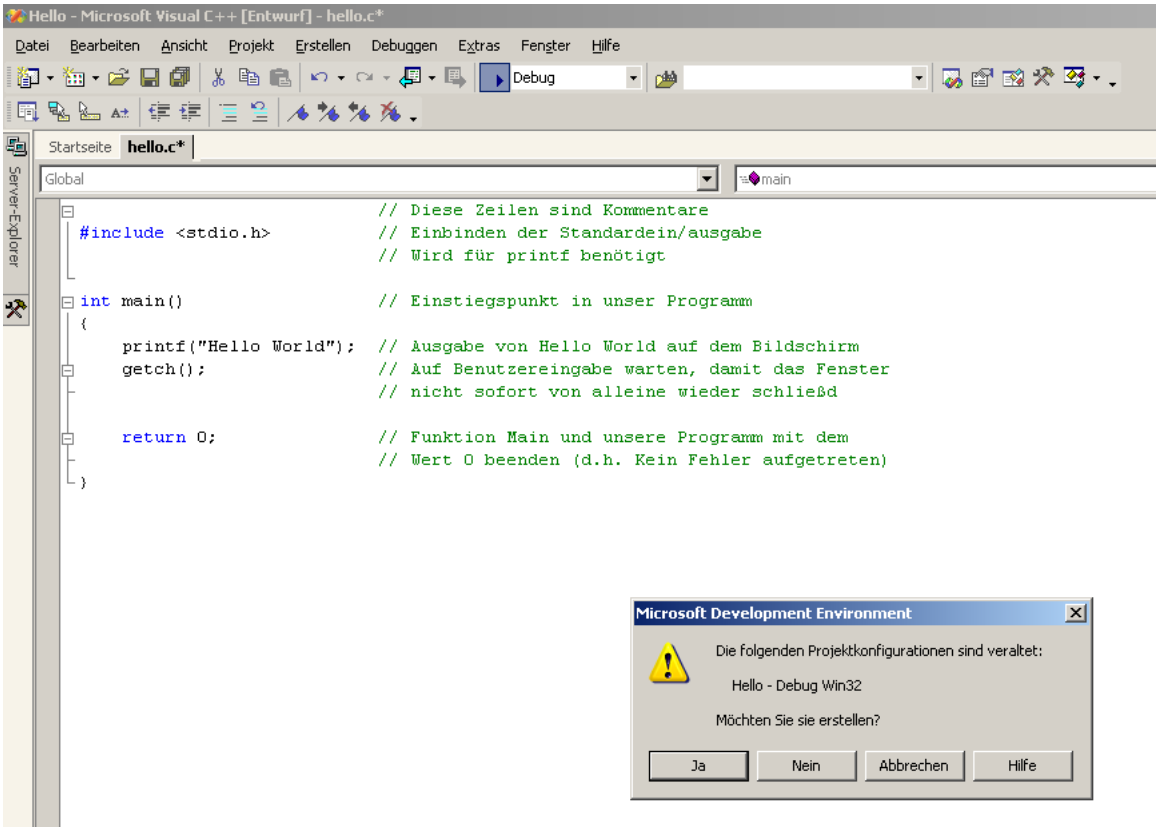


Sie haben nun eine leere Quelldatei erstellt in die Sie nun folgendes schreiben:

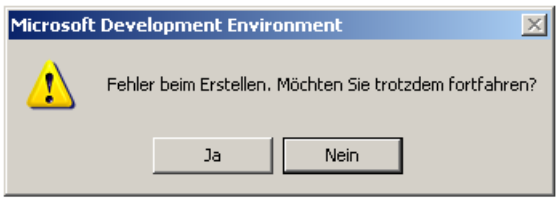
```
#include <stdio.h>          //Standard Ein/Ausgabe einbinden

int main()                  //Hauptfunktion die vom Betriebssystem
                            //aufgerufen wird
{
  printf("Hello World");    //Ausgabe auf dem Bildschirm
  getch();                  //Warten auf Benutzereingabe, damit das
                            //Fenster nicht automatisch schließt
  return 0;                 //Programm mit 0, keine Fehler, beenden
}
```

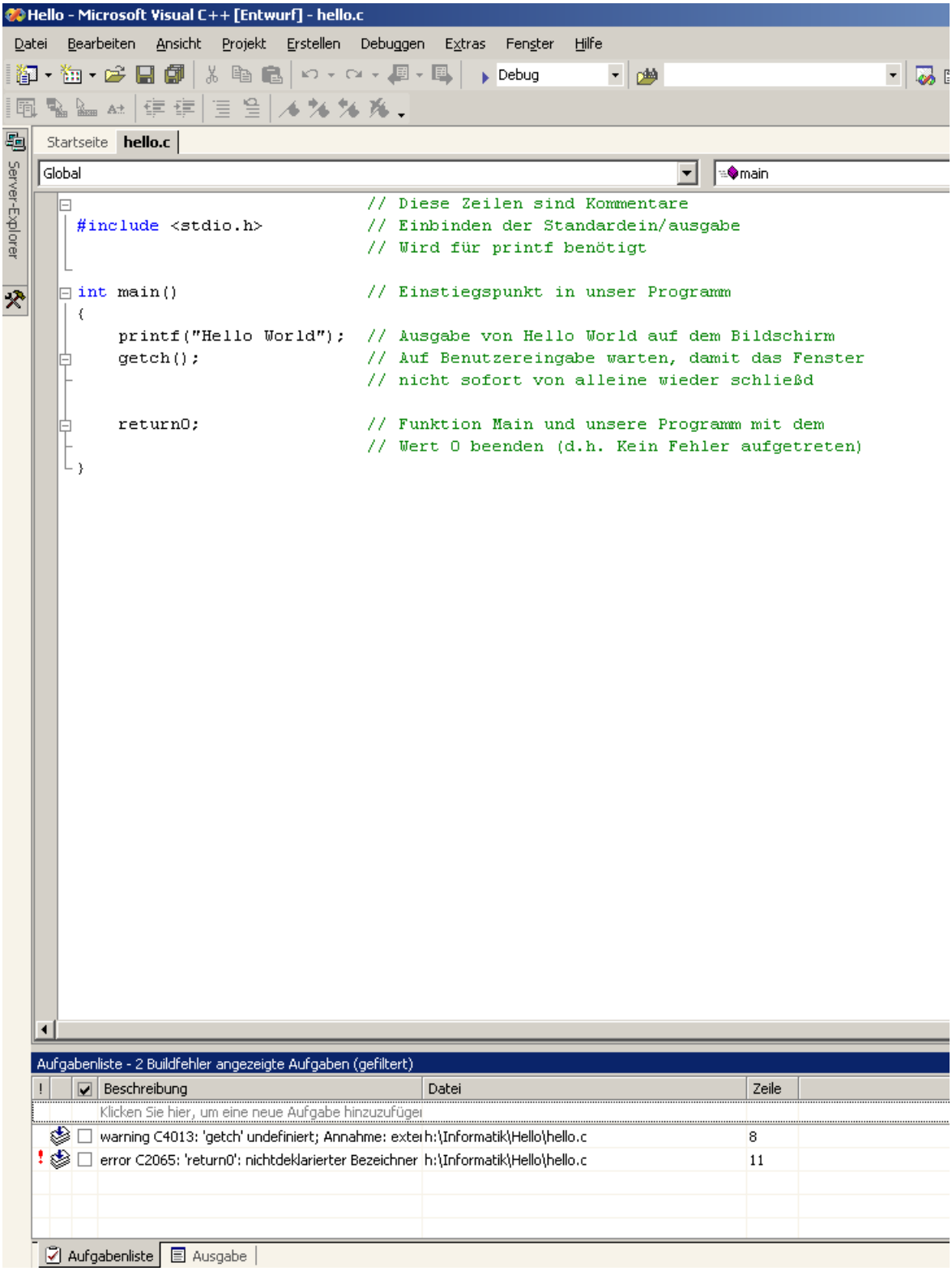
Anschließend drücken Sie F5 und bestätigen das nächste Fenster einfach mit **Ja**.



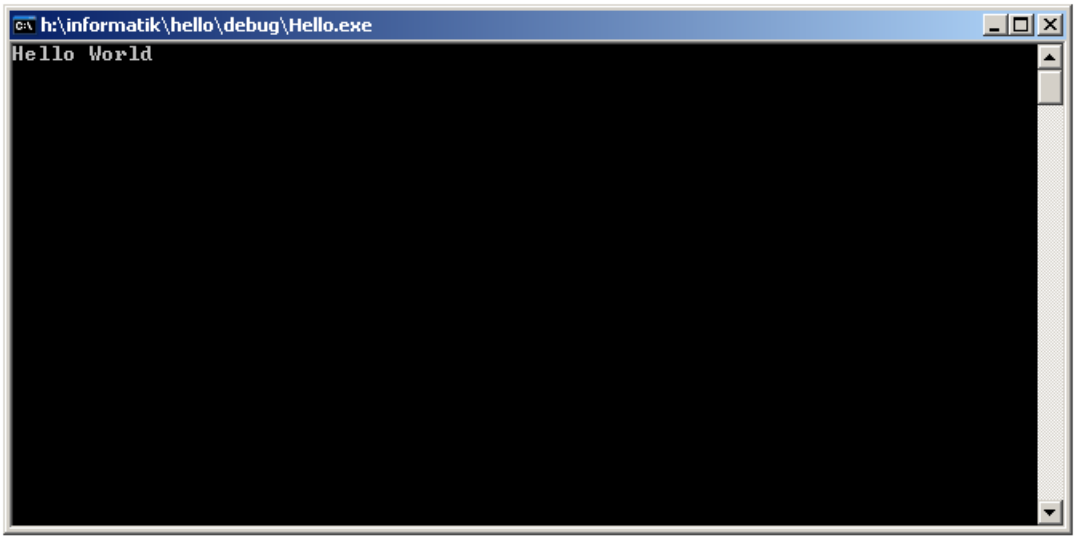
Wer jetzt kein schwarzes Fenster mit dem Text Hello World sieht, hat erstmal was falsch gemacht ;).
Darauf wird man auch durch den Compiler hingewiesen, der dann fragt ob das Projekt weiter erstellt werden soll. Hier klicken Sie dann auf **Nein**.



Am unteren Rand des Fensters sehen Sie nun eine Reihe von Warnungen und Fehlern. In meinem Fall zeigt der Compiler z.B. an das `return0` kein gültiger Ausdruck ist.



In diesem einfachen Beispiel können Sie eigentlich nur Fehler beim abschreiben gemacht haben. Korrigieren Sie diese und starten Sie ihr Programm erneut mit F5.



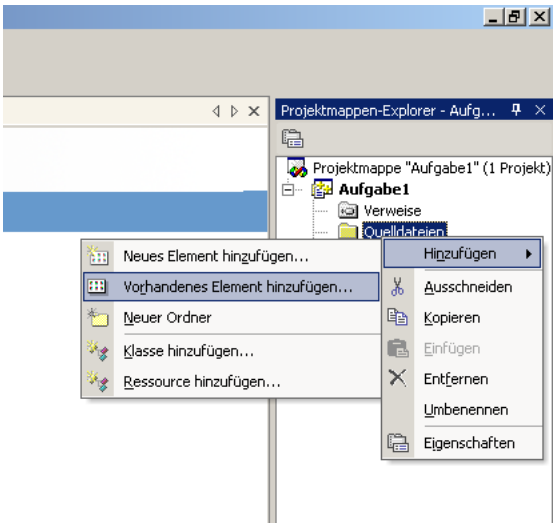
Glückwunsch, damit haben Sie nun ihr erstes Programm geschrieben. Nicht sonderlich beeindruckend, aber irgendwo muss man ja mal anfangen.

Kapitel 2

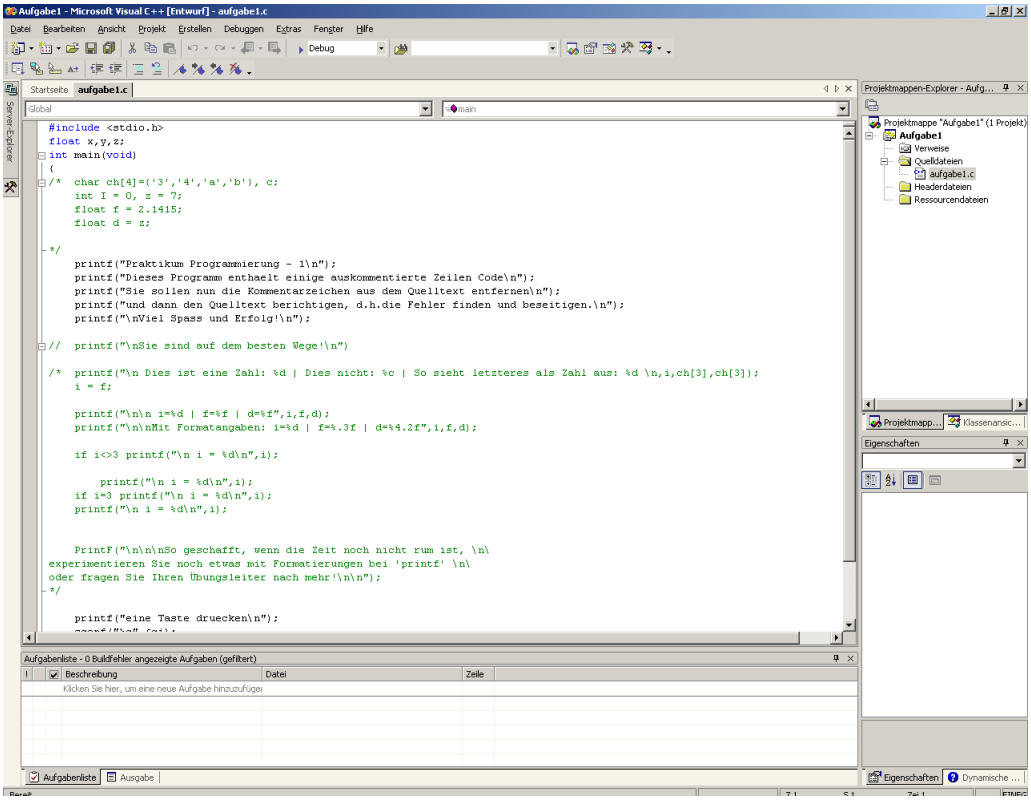
1. Aufgabe - Fehler finden

2.1 Projekt erstellen

Erstellen Sie zunächst ein neues leeres Projekt mit dem Namen `Aufgabe1` im Visual Studio, analog zum Hello World Beispiel bis zum Erstellen einer Datei.
Öffnen Sie dann
`http://www.orchid.inf.tu-dresden.de/gdp/lehre/Praktikum/` in einem Browser ihrer Wahl und speichern Sie die Datei *aufgabe1.c* mit Rechtsklick auf den Link und *Ziel speichern unter* in dem Verzeichnis `H:\Informatik\Aufgabe1\`. Klicken Sie nun wieder im Visual Studio im *Projektmappen-Explorer* rechts auf *Quelldateien*, *Hinzufügen* und *Vorhandenes Element hinzufügen* und wählen Sie nun *aufgabe1.c* aus und klicken auf **Öffnen**.

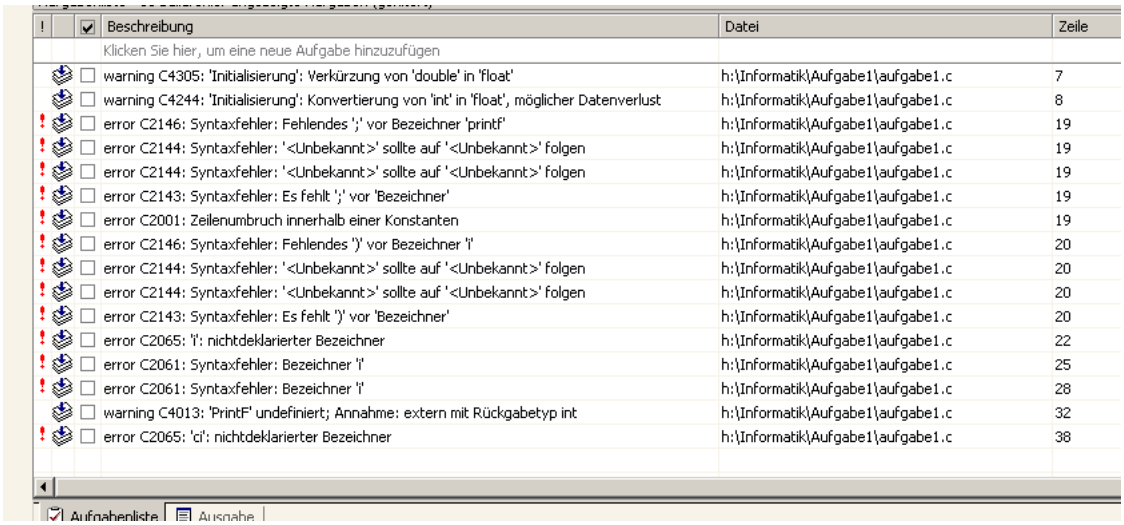


Mit einem Doppelklick auf *aufgabe1.c* im *Projektmappen-Exploren* öffnen Sie diese nun zum Bearbeiten.



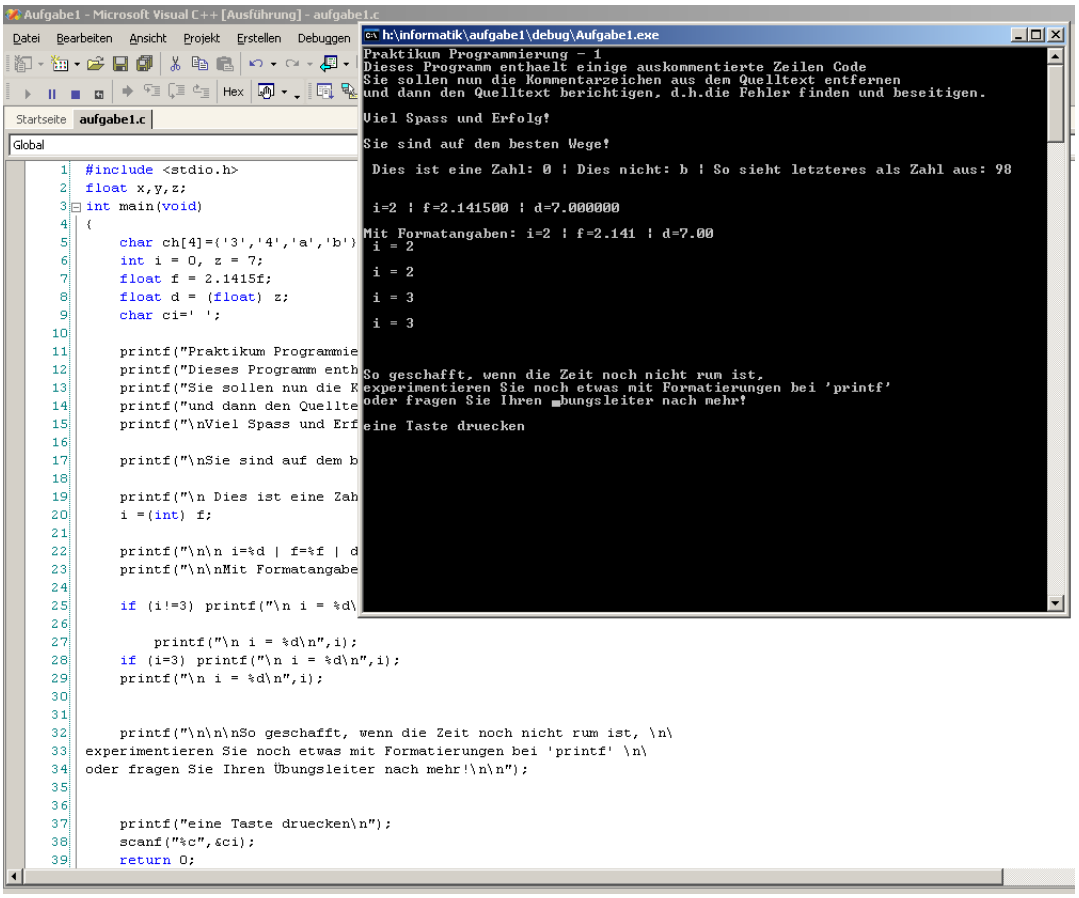
2.2 Die Aufgabe

Lesen Sie sich nun erst einmal die Aufgabe in der Quelldatei durch. Versuchen Sie die Fehler zu finden. Dazu können Sie das Programm mit F5 starten (Denn Dialog wieder mit **Ja** Bestätigen) und sich Fehlermeldungen ausgeben lassen. Gehen Sie dabei der Reihe nach von Oben nach Unten um alle Fehler zu finden und starten Sie immer mal ihr Programm dazwischen um zu gucken ob Meldungen hinzugekommen sind oder welche verschwunden sind. (Die Lösung ist hinter dem nächsten Bild. Probieren Sie es erstmal selbst bevor Sie weiter scrollen!!!)



Es sind 8 Fehler und 3 Warnungen enthalten

Art	Zeile		
Warnung	07	float f=2.1415f;	Float wird mit f gekennzeichnet
Warnung	08	float d= (float) z;	Damit wird die Konvertierung bewusst erzwungen und der Compiler weiß damit, dass es kein Fehler ist
Fehler	17	; fehlt am Ende	Jede Programmzeile in C endet mit einem Semikolon
Fehler	19	Schießendes „ fehlt vor ‚i	Strings werden mit „ geöffnet und auch wieder geschloßen
Fehler	06	I → i	C unterscheidet Groß- und Kleinschreibung. Im nachfolgenden wird immer ein kleines i verwenden, groß wäre aber auch möglich
Warnung	20	i=(int) f;	Wieder erzwungene Konvertierung
Fehler	25	if (..)	If Bedingung gehören in Klammern
Fehler	25	<> → !=	Ungleich wird durch != ausgedrückt
Fehler	28	if ( )	Wieder Klammern, ggf.. auch == wenn man vergleichen will, hier wird nur die Zuweisung überprüft
Fehler	32	printf	Standardfunktionen sind alle klein geschrieben
Fehler	09	char ci=’ ’; für Zeile 38	Nicht definierte char, ci → c



Damit hätten Sie die erste Aufgabe geschafft.

Kapitel 3

2. Aufgabe - Sortieren

3.1 Einführung

Um die Aufgabe bearbeiten zu können, müssen noch folgende Elemente eingeführt werden: Symbolische Konstanten, Arrays, while-Schleifen, for-Schleifen und der Zufallszahlengenerator.

In der Aufgabe wird gleich zu Beginn eine symbolische Konstante definiert. Diese wird durch ein `#define`, gefolgt von dem Bezeichner und dem Wert definiert. Solche Konstanten werden beim Kompilieren der Datei sofort an allen Stellen im Quelltext ersetzt, d.h. zur Laufzeit existiert diese Konstante schon nicht mehr. Sinnvolle Verwendung für symbolische Konstanten sind wie in unserem Beispiel die Grenzen von Datenfelder oder auch andere konstante Zahlen oder Wörter im Quelltext, die häufiger vorkommen und vielleicht mal geändert werden müssten. In so einem Fall ist es ja angenehmer eine Zeile am Anfang der Datei zu ändern, als alle Vorkommen im Quelltext zu suchen.

Die Konstanten bringen uns gleich zum zweiten „neuen“ Element in unserem Quelltext zu den Datenfeldern oder auch *Arrays*. Arrays können im Prinzip von jedem Datentyp (hier: int) erstellt werden. Dabei wird wie bei jeder Deklaration erst der Datentyp und der Name angegeben, und gleich daran die Länge bzw. Dimension des Arrays.

```
int arr[4][2][5]; //Dimension ist dann 3 mit insgesamt
                  //40 Zellen vom Typ int
```

Wir verwenden hier 1- und 2-dimensionale Arrays um unser Zahlenfeld zu verwalten. Die Vorteile der Verwendung von Arrays dürften offensichtlich sein, man benötigt nur eine Variable und nicht 40 um bei unserem Beispiel zu bleiben. Eine wichtige Besonderheit muss bei Arrays aber immer berücksichtigt werden, sie beginnen immer bei 0 mit der Indizierung und nicht bei 1 wie in der Mathematik üblich. D.h. ein Array *int arr[3]* hat die Elemente *arr[0]*, *arr[1]* und *arr[2]*, und diese können jetzt wie „normale“ Variablen verwendet werden. Vorerst wollen wir erstmal davon ausgehen das die Länge von Array immer konstant ist und nicht verändert werden kann. Gibt man einen Index außerhalb der Grenzen an, bekommt man einen Fehler.

Um die Aufgabe nun lösen zu können, müssen wir nun nur noch Schleifen einführen. Schleifen gibt es in allen Programmiersprachen recht viele, wir wollen zunächst die *while*-Schleife anschauen.

```
while(Bedingung) //Boolesche Ausdruck
{
  //Code der Schleife
}
```

Eine while-Schleife prüft vor Eintritt in den Schleifenrumpf und vor jedem neuen Durchlauf ob die Bedingung noch wahr ist, ist dies nicht mehr der Fall wird der Code nach dem Schleifenkontrukt fortgesetzt.

Die *for*-Schleife ist am besten als Zählschleife einzusetzen. Im Gegensatz zur while-Schleife geben wir der for-Schleife nicht einen, sondern drei durch Semikolon getrennte Parameter. Der Erste ist das was vor der Schleife ausgeführt wird, der Zweite ist die Laufbedingung (Gegenteil von Abbruchbedingung), d.h. solange diese Wahr liefert läuft die Schleife, und der dritte Parameter wird mit jedem Schleifendurchlauf ausgeführt, i.A zählt man hier eine Variable hoch.

```
int i;
//...
for(i=0; i<10;i++)
{
// i=0..9 -> 10 Durchläufe
}
//Hier ist i=10!!
```

Um Zufallszahlen in C erstellen zu lassen, benötigen wir die Bibliothek *time.h* und *stdlib.h*. In unserer Aufgabe sind diese bereits eingebunden. Um nun eine Zufallszahl zu erhalten, müssen wir den Generator erst mit `srand((unsigned int) time(NULL));` initiieren. Und mit `int a=rand();` können wir ganz einfach Zufallszahlen erhalten.

3.2 Die Aufgabe

Die Aufgabe kann wieder über die bekannte Adresse bezogen werden.

Wir wollen uns hier nicht zu viel mit den Ausnahmesituationen beschäftigen, und gehen davon aus das der User immer korrekte Eingaben macht. Für das Selbststudium würde es sich aber anbieten, die Eingabe mal „Idiotensicher“ zu machen.

Um einen Einstieg in die Aufgabe zu bekommen, sollten Sie zunächst die beiden Eingaben, und die Ausgaben implementieren. Dem Sortieren widmen wir uns dann anschließen. (Wer schon Ideen dafür hat, sollte nicht weiter lesen und diese versuchen umzusetzen.)

Tastatureingabe

Im einfachsten Fall müssen wir hier ja bloß mit *scanf(...)* 25 Zahlen einlesen und diese unserem Feld mit dem richtigen Index zuweisen. Da unsere 25 eine willkürliche Grenze ist, und auch 2.5 Million heißen könnte, wäre es natürlich nicht sinnvoll alle diese Anweisung untereinander aufzuschreiben. Für so etwas haben wir die for-Schleife eingeführt. (ACHTUNG: Arrays beginnen bei 0!!)

Nun sollte einem eigentlich der Gedanke kommen „Was mach ich wenn ich bloß 5 Zahlen sortieren will?“. Um diese Frage zu beantworten gibt es 2 Lösungsansätze, der erste, vom Lehrstuhl für dieses Bsp. bevorzugte, wäre vorher noch zu Fragen wie viel Zahlen man eingeben möchte. Die Lösung gefällt mir persönlich nicht so und deshalb möchte ich eine Alternative dazu anbieten. Wir bieten dem Benutzer die Möglichkeit die Eingabe von Zahlen, durch z.B. die Zahl 0 zu beenden. D.h. wir müssen noch überprüfen ob der Benutzer eine 0 eingegeben hat, und dann ggf. die Schleife vorzeitig verlassen. Dazu wird der `break;` Befehl verwendet. Bei dieser Lösung müssen wir aber darauf achten das wir mitzählen wie viele Zahlen es sind.

```
printf("\nEingabe der Elemente von der Tastatur \n
          0 beendet die Eingabe\n");

for (i=0;i<MAX; i++)
{
  printf ("%0d: ",i+1); //Noch was für's Layout
  scanf("%d",&help);
```

```
  if (help==0)  break;
  else feld[i]=help;
  anzahl++;
}
```

Zufallszahlen

Diesmal kommen wir leider nicht umhin zu fragen wir viele Elemente es sein sollen. Dabei muss die Eingabe natürlich kleiner als MAX sein. Bevor wir aber unsere Schleife für die Zufallszahlen bauen, müssen wir noch den Zufallszahlengenerator starten.

```
scanf("%i",&anzahl);
while (anzahl>MAX)
{
  printf("Zu viele Elemente, max. %i zulässig: ",MAX);
  scanf("%i",&anzahl);
}

srand((unsigned int) time(NULL));   //Zufallsgenerator starten

printf("\nErzeugung der Werte ueber Zufallszahlen \n");
for (i=0;i<anzahl;i++) feld[i]=rand();
```

Ausgabe

Was nun noch fehlt ist die Ausgabe. Die bauen wir natürlich nur einmal und kopieren sie dann noch nach unten. Wer sich schon mit Funktionen beschäftigt hat, könnten natürlich auch eine Funktion daraus machen.

Da unsere Ausgabe nach der Aufgabenstellung „ansprechend“ aussehen soll, müssen wir auf einige Tricks zurück greifen. Wir wissen z.B. das unsere Konsole 80 Zeichen breit ist, und unsere Zahlen nicht mehr als 11 Zeichen haben (4Byte). Mit etwas Mathematik bleibt also nur die Möglichkeit 4 Zahlen nebeneinander darzustellen, die trennen wir durch ein Leerzeichen und es bleiben 19 Zeichen für die Zahl. Und da hilft uns nun `printf(...)` die Ausgabe zu formatieren.

```
for (i=0;i<anzahl;i++)
{
  printf("%19d ",feld[i]);
}
```

Sortieren

Nach dem die Vorbereitung nun abgeschloßen ist, können wir mit dem Schwerpunkt der Aufgaben anfangen ;). Um Zahlen zu sortieren gibt es unzählige gute und weniger gute Algorithmen. Wir wollen hier einen der weniger guten versuchen zu implementieren, und zwar Bubblesort. Bubblesort beginnt am Anfang des Feldes und vergleicht die ersten beiden Elemente, ist das Zweite kleiner werden die Beiden getauscht. Dann wird das Zweite mit dem Dritten verglichen, und ggf. getauscht usw. Wenn man am Ende (bzw. n-1) angekommen, geht man wieder nach vorne und macht das Ganze nochmal. Dies wiederholt man auch max. (n-1)-mal. Danach ist das Feld sortiert. Wir haben also hier einen Aufwand von $(n-1)^2$ und deshalb ist das eher ein schlechter Algorithmus. Wer dies etwas optimieren will könnte z.B. Quicksort nehmen (Google hilft da). Aber kommen wir zu einen Bubblesort-Beispiel:

5	2	7	1	Start; 5 und 2 vergleichen und tauschen
2	5	7	1	5 und 7 vergleichen und nicht tauschen
2	5	7	1	7 und 1 vergleichen und tauschen
2	5	1	7	7 ist fertig
2	5	1	7	Zurück; 2 und 5 vergleichen und nicht tauschen
2	5	1	7	5 und 1 vergleichen und tauschen
2	1	5	7	5 ist auch fertig
2	1	5	7	Zurück; 2 und 1 vergleichen und tauschen
1	2	5	7	2 ist fertig, und damit auch 1

Und hier nun noch die Implementierung.

```
for (j=0;j<anzahl-1;j++)
{
  for (i=0;i<anzahl-1;i++)
  {
    if (feld[i]>feld[i+1])
    {
      help=feld[i];  //Tausch über Hilfsvariable
      feld[i]=feld[i+1];
      feld[i+1]=help;
    }
  }
}
```

3.3 Eine Lösung

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

#define MAX 25

int main(void)
{
  int feld[MAX];
  int anzahl=0,i,j,help;
  char ch;
//--------- Eingabe ----------------------
  printf("Eingabe der Werte\n");

  printf("\nSollen die Elemente von der Tastatur eingegeben werden (j/n) ?");
  do ch = toupper(getch()); while ((ch!='J')&&(ch!='N'));
  if (ch == 'J') //Tastatur
  {
    printf("\nEingabe der Elemente von der Tastatur \n0 beendet die Eingabe\n
    for (i=0;i<MAX;i++)
    {
        printf ("%0d: ",i+1);
        scanf("%d",&help);
        if (help==0)  break;
```

```
            else feld[i]=help;
            anzahl++;
        }
    }
    else //Zufall
    {
        printf("\nWieviele Zahlen sollen generiert werden? ") ;
        scanf("%i",&anzahl);
        while (anzahl>MAX)
        {
            printf("Zu viele Elemente, max. %i zulässig: ",MAX);
            scanf("%i",&anzahl);
        }
        srand((unsigned int) time(NULL));   //Zufallsgenerator starten

        printf("\nErzeugung der Werte ueber Zufallszahlen \n");
        for (i=0;i<anzahl;i++) feld[i]=rand();
     }
//--------- Ausgabe unsortiertes Feld ----------------------
    printf("\n\nDas unsortierte Feld lautet:\n");
    for (i=0;i<anzahl;i++) printf("%19d ",feld[i]);
//--------- Sortieren ----------
    printf("\n\nJetzt wird sortiert, bitte gedulden Sie sich einen Moment!\n");

    for (j=0;j<anzahl-1;j++)
    {
        for (i=0;i<anzahl-1;i++)
        {
            if (feld[i]>feld[i+1])
            {
                help=feld[i];
                feld[i]=feld[i+1];
                feld[i+1]=help;
            }
        }
    }
//--------- Ausgabe sortiertes Feld ----------------------
    printf("\n\nDas sortierte Feld lautet:\n");

    for (i=0;i<anzahl;i++) printf("%19d ",feld[i]);

    printf("\n\nBeenden mit Taste");
    getch();
    return 0;
}
```

An der Aufgabe könnte man jetzt noch eine ganze Weile weiter basteln, die Eingabe sicher machen oder das Sortieren beschleunigen, aber das ist eher eine schöne Hausaufgabe. Von mir an dieser Stelle nur noch die Ausgabe.


F:\Scripte\Tutorial\PP\Aufgaben\INF02\sort.exe
Eingabe der Werte
Sollen die Elemente von der Tastatur eingegeben werden (j/n) ?
Wieviele Zahlen sollen generiert werden? 10
Erzeugung der Werte ueber Zufallszahlen
Das unsortierte Feld lautet:
16527 29055 18738 19455
2459 5045 11721 29392
29860 7265
Jetzt wird sortiert, bitte gedulden Sie sich einen Moment!
Das sortierte Feld lautet:
2459 5045 7265 11721
16527 18738 19455 29055
29392 29860
Beenden mit Taste

Kapitel 4

3. Aufgabe - Sortieren II

4.1 Einführung

Ziel der heutigen Übung ist den Umgang mit Funktionen, Parametern und Rückgabewerten zu üben. Eine Funktion kennen wir dabei bereits.

```
int main()    //Rückgabewert int; kein Paramter
{
  return 0;   //Rückgabe 0
}
```

Wir erkennen also, dass eine Funktion genau wie eine Variable einen Typ haben muss, der dem Rückgabewert entspricht. Und diesen Rückgabewert können wir mit `return` einen Wert geben. Möchten wir nichts zurückgeben, erhält die Funktion den Typ `void`.

```
void mach_was()
{
  //return ; optionales return um die Funktion vorzeitig zu verlassen
}
```

Neben einer Rückgabe von einer Funktion möchten wir der Funktion oft auch eine Eingabe mitgeben. Die Parameter für die Eingabe werden durch Komma getrennt, mit dem Typ versehen in die runden Klammer nach dem Funktionsnamen geschrieben. Z.B. um eine Summe zu berechnen:

```
int summe(int zahl1, int zahl2, int zahl3)
{
  return zahl1+zahl2+zahl3;
}
//Aufruf
int z=1,r=0;
r=summe (2,4,z);
```

Der Aufruf unsere Funktionen erfolgt wie der Aufruf der „Systemfunktionen“ die wir bereits kennen.

In unserem Beispiel übergeben wir der Funktionen die Zahlen 2, 4 und den **Inhalt** der Variable z. Warum hab ich jetzt Inhalt so betont? Ganz einfach, die Variable die wir so übergeben werden von der Funktion nicht verändert. D.h. z ist vor dem Aufruf 1 und danach immer noch, auch wenn wir zahl3 ändern würden. Möchten wir nun aber unsere Variable verändern dürfen wir diese nicht als Wert übergeben, sonder müssen diese als Referenz übergeben. Das beste Beispiel dafür ist scanf, welches ja als zweiten Paramter die Variable enthält in die die Eingabe gespeichert werden soll. Und da haben wir ja gelernt das wir dies beim Aufruf mit

dem Referenzoperator & machen. Auf der anderen Seite, also der Funktion müssen wir dies ebenfalls durch den Derefenzierungsoperator * kennzeichen. Hier wieder ein Beispiel.

```
int a=3,b=6;  //globale Varibalen
void swap_n(int x,int y)
{
  int h=0; //lokale Varibalen
  h=x;
  x=y;
  y=h;
}

void swap_r(int *x,int *y)
{
  int h=0;
  h=*x;
  *x=*y;
  *y=h;
}
//Aufruf
//a=3 b=6
swap_n(a,b);
//a=3 b=6
 swap_r(&a,&b);
//a=6 b=3
```

Wenn man Array übergeben will, verwendet man am einfachsten [ ] hinter dem Namen des Parameters. Ganz nebenbei habe ich jetzt noch die Begriffe der globalen und lokalen Variablen eingeführt. Variablen gelten, sofern sie in einer Funktion definiert wurden, nur innerhalb dieser. Sollte eine Globale den selben Namen haben wie eine Lokale haben, dann wird die Globale temporär von der Lokalen in der Funktion verdeckt.

4.2 Die Aufgabe

Es geht mit dem Programm vom letzten Mal weiter. Alle Teilschritte sollen in Funktionen ausgelagert werden und Sie sollen sich mit anderen Suchalgorithmen vertraut machen. Suchalgorithmen finde Sie Unmengen mit Google, daher werde ich darauf nicht näher eingehen. Außerdem werden Sie in der Zukunft noch öfter selbstständig Wissen erarbeiten müssen, und da schadet ein gezielter, fähiger Umgang mit Suchmaschinen nicht ;). Aber auch zu den Funktionen kann ich nicht viel sagen, ohne das es schon ein Lösung ist. Machen Sie sich klar was die Funktionen brauchen und was sie zurück geben. Wie immer gibt es mehrere Lösungen.

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

#define MAX 25

int main(void)
{
  int feld[MAX];
  int anzahl=0,i,j,help;
```

```
  char ch;
//-------- Eingabe ----------------------
  printf("Eingabe der Werte\n");

  printf("\nSollen die Elemente von der Tastatur eingegeben werden (j/n) ?");
  do ch = toupper(getch()); while ((ch!='J')&&(ch!='N'));
  if (ch == 'J') //Tastatur
  {
    printf("\nEingabe der Elemente von der Tastatur \n0 beendet die Eingabe\n
    for (i=0;i<MAX;i++)
    {
        printf ("%0d: ",i+1);
        scanf("%d",&help);
        if (help==0) break;
        else feld[i]=help;
        anzahl++;
    }
  }
  else //Zufall
  {
    printf("\nWieviele Zahlen sollen generiert werden? ") ;
    scanf("%i",&anzahl);
    while (anzahl>MAX)
    {
      printf("Zu viele Elemente, max. %i zulässig: ",MAX);
      scanf("%i",&anzahl);
    }
    srand((unsigned int) time(NULL));  //Zufallsgenerator starten

    printf("\nErzeugung der Werte ueber Zufallszahlen \n");
    for (i=0;i<anzahl;i++) feld[i]=rand();
   }
//-------- Ausgabe unsortiertes Feld ----------------------
  printf("\n\nDas unsortierte Feld lautet:\n");
  for (i=0;i<anzahl;i++) printf("%19d ",feld[i]);
//-------- Sortieren ----------
  printf("\n\nJetzt wird sortiert, bitte gedulden Sie sich einen Moment!\n");

  for (j=0;j<anzahl-1;j++)
  {
    for (i=0;i<anzahl-1;i++)
    {
      if (feld[i]>feld[i+1])
      {
        help=feld[i];
        feld[i]=feld[i+1];
        feld[i+1]=help;
      }
    }
  }
//-------- Ausgabe sortiertes Feld ----------------------
  printf("\n\nDas sortierte Feld lautet:\n");
```

```
  for (i=0;i<anzahl;i++) printf("%19d ",feld[i]);

  printf("\n\nBeenden mit Taste");
  getch();
  return 0;
}
```

4.3 Eine Lösung

Hier nun wieder eine Lösung. Bei manchen könnte man mehr in die Funktion einlagern oder auch weniger, oder die Parameter oder den Rückgabewert ändern. Aber das liegt alles im ermessen des Programmierers...

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

#define MAX 25

void ausgabe(int f[],int a)
{
  int i=0;
  for (i=0;i<a;i++) printf("%19d ",f[i]);
}
void swap(int *a, int *b)
{
  int h=0;
  h=*a;
  *a=*b;
  *b=h;
}
void zufall(int f[],int a)
{
  int i=0;
  srand((unsigned int) time(NULL));   //Zufallsgenerator starten
  for (i=0;i<a;i++) f[i]=rand();
}
int eingabe(int f[])
{
  int i=0,a=0,h=0;
  for (i=0;i<MAX;i++)
  {
    printf ("%0d: ",i+1);
    scanf("%d",&h);
    if (h==0)  break;
    else f[i]=h;
    a++;
  }
  return a; //Anzahl zurück
}
```

```
void sort(int f[],int a)
{
  int i=0,j=0;
  for (j=0;j<a-1;j++)
  {
    for (i=0;i<a-1;i++)
    {
      if (f[i]>f[i+1])
      {
        swap(&f[i],&f[i+1]);
      }
    }
  }
}

int main(void)
{
  int feld[MAX];
  int anzahl=0;
  char ch;
//--------- Eingabe ----------------------
  printf("Eingabe der Werte\n");

  printf("\nSollen die Elemente von der Tastatur eingegeben werden (j/n) ?
  do ch = toupper(getch()); while ((ch!='J')&&(ch!='N'));
  if (ch == 'J') //Tastatur
  {
    printf("\nEingabe der Elemente von der Tastatur \n0 beendet die Eingabe\n
    anzahl=eingabe(feld);
  }
  else //Zufall
  {
    printf("\nWieviele Zahlen sollen generiert werden? ") ;
    scanf("%i",&anzahl);
    while (anzahl>MAX)
    {
      printf("Zu viele Elemente, max. %i zulässig: ",MAX);
      scanf("%i",&anzahl);
    }
    printf("\nErzeugung der Werte ueber Zufallszahlen \n");
    zufall(feld,anzahl);
   }
//--------- Ausgabe unsortiertes Feld ----------------------
  printf("\n\nDas unsortierte Feld lautet:\n");
   ausgabe(feld,anzahl);
//--------- Sortieren ----------
  printf("\n\nJetzt wird sortiert, bitte gedulden Sie sich einen Moment!\n");
  sort(feld,anzahl);

//--------- Ausgabe sortiertes Feld ----------------------
  printf("\n\nDas sortierte Feld lautet:\n");
```

```
  ausgabe(feld,anzahl);

  printf("\n\nBeenden mit Taste");
  getch();
  return 0;
}
```

Kapitel 5

4. Aufgabe - Einfach verkettete Liste

5.1 Einführung

In der vierten Aufgabe wollen wir uns mit erweiterten Datentypen, hier speziell den Einfach-Verketteten-Listen, beschäftigen. Aus der Vorlesung und Übung AuD sollte die Grundlagen schon vorhanden sein, daher möchte ich darauf nicht im Detail oder mit bunten Bildchen eingehen. Wer das ganze jedoch nochmal grafisch nachvollziehen will, und dazu rate ich eigentlich jedem, findet eine detaillierte und bebilderte Erklärung im AuD Script ab Seite 67ff.

Schauen wir uns zunächst die Datenstruktur an.

```
typedef struct elem *IntList;
typedef struct elem { int wert;
                      IntList next;
                    } elemtype;
```

Es muss also ein neuer Datentyp mit `typedef` definiert werden, und dieser soll eine Struktur mit dem Namen „elem“ von je einem Integer und einen Zeiger auf eine baugleiche Struktur sein. Da C es verbietet eine Struktur durch sich selbst zu definieren, müssen wir zuvor noch einen Zeiger auf unsere Struktur definieren, diesen nennen wir „IntList“, und können ihn dann in der Struktur wieder verwenden. Diesen Zeigertyp brauchen wir auch für einige Funktion. Mit „elemtype“ geben wir der Struktur noch einen Namen(Brauchen wir auch später).

Wollen wir nun eine neues Element unserer Liste erzeugen, müssen wir dynamisch zur Laufzeit Speicher zuweisen, diese machen wir mit `malloc()` wie folgt:

```
IntList neu;                                  //Zeiger auf neues Element
neu = (IntList)malloc(sizeof(elemtype));      //Element Speicher zuweisen
neu->wert = n;                                //Einen Wert eintragen
neu->next=NULL;                               //Zeiger auf das nächste
                                              //Element erstmal auf NULL
```

Mit `sizeof()` ermitteln wir die Größe unserer Struktur, und geben dies dann an malloc weiter, um den Speicherplatz zu bekommen. Würden wir sizeof() auf eine Integer aufrufen, würden wir z.B. 4 (Byte) als Rückgabe erhalten.

5.2 Die Aufgabe

Nach Durchlesen des bereits fertigen Quelltext in der Aufgabenstellung, wissen wir nun das wir die Funktionen für das Anhängen von Elementen an das Ende der Liste, das Abspalten des Kopfelements, das Ermitteln der Länge von Listen, das Ausgeben vo Listen, sowie das sortierte Einfügen in Listen implementieren sollen. In der main-Funktionen sollen wir dann noch das Aufteilen der Zufallsliste in zwei sortierte Listen(gerade und ungerade Zahlen) hinzufügen.

Mein Empfehlung zum Herangehen an die Aufgabe: Erstmal noch „Anhängen von Elementen„ lesen um ein Gefühl für die Thematik zu bekommen. Anschließend die Länge und die Ausgabe in jedem Fall versuchen alleine zu lösen. Das Abtrennen des Kopfelements und das sortierte Einfügen auch erstmal allein probieren und nur im Notfall oder zur Korrektur hier weiter lesen. Wenn de Funktionen alle stehen, dann dürfe das Aufteilen auch kein Problem mehr sein.

Anhängen von Elementen

Wie dem AuD Script, und den Skizzen darin, zu entnehmen, müssen wir jetzt zunächst ein Element erzeugen und mit einem Wert und dem NULL-Zeiger füllen. Um das Element nun anzuhängen, sollten wir natürlich erstmal überprüfen ob unsere Liste überhaupt schon vorhanden ist. Ist die nicht der Fall lassen wir den Anfang der Liste auf unser neues Element zeigen. Der aufwändigere Fall, das es schon eine Liste gibt, zwingt uns zunächst die Liste bis zum Ende durchzulaufen, und dann den letzten Next-Zeiger auf unser neues Element zu setzen. Wichtig ist dabei das wir unseren Listen-Anfangs-Zeiger nicht verlieren. Ist der erstmal weg, sind die Daten auf immer im Speicher verschollen!!!

```
void Append(IntList *l, int n)
{ IntList neu,help;

  neu = (IntList)malloc(sizeof(elemtype));    //Element erzeugen
  neu->wert = n;
  neu->next=NULL;

  if (*l==NULL) *l = neu;                     //Liste noch leer
  else                                        //Ende suchen
  {
    help = *l;                              //Nicht auf dem Orginal machen!
    while (help->next != NULL) help=help->next;
    help->next=neu;
  }
}
```

Wie gesagt, nun erstmal durchdenken und dann mit Länge und Ausgabe selbst probieren.

Länge ermitteln

Die Funktion möchte ich nutzen um neben der iterativen Lösung (also durch Schleifen), auch ein rekursive zu zeigen. I.A. bietet es sich an auf einer rekursiven Datenstruktur alle Funktionen rekursiv zu gestalten, meist sind aber die iterativen intuitiver und einfacher nach zu vollziehen. Wir wissen das unsere Liste die Länge 0 hat, wenn der Zeiger auf den Anfang NULL ist. Und wir wissen auch, wenn dies nicht der Fall ist, ist die Liste um 1 länger als Liste die an dem Next-Zeiger des Elements hängt. Daraus folgt dann:

```
int GetLaenge(IntList l)
{
  if (l==NULL) return 0;
  return 1+GetLaenge(l->next);
}
```

Ausgabe

Die Ausgabe ist eigentlich identisch zur Länge, bloß statt die Länge zu zählen, geben wir etwas mit printf() aus. Hier halte ich mich dann wieder an die Musterlösung und gebe die iterative

Funktion an.

```
void ListeAusgeben(IntList l)
{
  if(l==NULL) printf("\nDie Liste ist leer!\n");
  else
  {
    printf("\n");
    while(l!=NULL)                    //solange ein Element existiert
          {
            printf("%8d",l->wert);    //dieses ausgeben
            l=l->next;                //naechstes vorbereiten
          }
          printf("\n");
  }
}
```

Kopfelement abtrennen

Für die Umsetzung müssen wir nur den Anfangszeiger der Liste auf das nächste Element setzen, aber vorher den Kopfzeiger sichern um ihn zurück geben zu können.

```
IntList GetEle(IntList *l)
{
  IntList Ele = *l;        //Kopfelement festhalten

  *l = (*l)->next;         //neuer Listenanfang
  Ele->next = NULL;        //(altes) Kopfelement von Liste abtrennen
  return Ele;              //Adresse zurueckgeben
}
```

Sortiert Einfügen

Etwas schwieriger ist das sortierte Einfügen in ein Liste. Zuerst muss wieder geprüft werden ob die Liste leer ist, oder ob das einzufügende Element kleiner als das erste Element der Liste ist. Alternativ müssen wir sonst wieder die Liste solange durchlaufen bis wir die richtige Stelle gefunden haben.

```
void InsertEle(IntList *l, IntList Ele)
{
  IntList help;

  if ((*l==NULL) || ((*l)->wert >= Ele->wert))    //vorn einfuegen
  {
    Ele->next = *l;
    *l = Ele;
  }
  else                                            //hinter 'help' einfuegen
  {
    help = *l;
    while ((help->next != NULL)&&(help->next->wert < Ele->wert))
    {
      help=help->next;
```

```
    }
        Ele->next=help->next;
    help->next=Ele;
  }
}
```

Aufteilen der Listen

An der entsprechend markierten Stelle im Quellcode soll nun abschließend noch das Aufteilen in zwei Listen implementiert werden. Mit Hilfe der bereits definierten Funktionen ist dies aber nur noch Formsache. Wir entfernen jeweils das Kopfelement, überprüfen ob der Wert gerade bzw. ungerade ist, und fügen das Element in die entsprechende Liste ein.

```
while (Liste != NULL)
{
  if (Liste->wert%2) InsertEle(&UListe, GetEle(&Liste));
  else InsertEle(&GListe, GetEle(&Liste));
}
```

5.3 Eine Lösung

Die hier implementierten Funktionen sind beliebte Prüfungsaufgaben. Alle Baum- und Listenaufgaben sind prüfungsrelevant!

Nun nochmal die Lösung fast am Stück.

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

typedef struct elem *IntList;
typedef struct elem { int wert;
                      IntList next;
                    } elemtype;

//HIER DIE FUNKTIONSDEFINITIONEN EINFÜGEN

// 0 soll in der Liste als wert nicht vorkommen !!
void ZufallsListe(IntList *l)
{ int i, anzahl;

  srand((unsigned) time(NULL));
  anzahl = 10 + rand()%20;
  for(i=0;i<anzahl;i++) Append(l,1+rand());
        //haenge anzahl elemente an eine bisher leere liste an
}

void NeueListe(IntList *l)
{ int Zahl;

  printf("\nEingabe der Listenelemente (0 = Ende)\n\n");
  do { printf("Zahl: ");
```

```
        scanf("%i",&Zahl);
        if (Zahl != 0)  Append(l,Zahl);
      } while (Zahl != 0);
}



int main(void)
{ IntList Liste = NULL,      // Gesamtliste
          UListe = NULL,     // Liste mit ungeraden Werten
          GListe = NULL;     // Liste mit geraden Werten
  char ch;

  printf("\nSollen die Elemente von der Tastatur eingegeben werden (j/n)? ");
  do ch = toupper(getch()); while ((ch!='J')&&(ch!='N'));
  if (ch == 'J') NeueListe(&Liste);
     else  ZufallsListe(&Liste);

//--- Ausgeben der Gesamtliste ------------------------

  printf("\nGesamtliste mit %d Elementen: \n", GetLaenge(Liste));
  ListeAusgeben(Liste);

  printf("\nJetzt wird geordnet\n");

//--- Aufloesen der Gesamtliste und erzeugen zweier geordneter Listen

//--- mit geraden bzw. ungeraden Werten

  while (Liste != NULL)
    if (Liste->wert%2) InsertEle(&UListe, GetEle(&Liste));
       else InsertEle(&GListe, GetEle(&Liste));

//--- Ausgeben aller Listen ---------------------------

  printf("\nGesamtliste mit %d Elementen: \n", GetLaenge(Liste));
  ListeAusgeben(Liste);      // muss leer sein !!

  printf("\n\nListe mit %d ungeraden Elementen: \n", GetLaenge(UListe));
  ListeAusgeben(UListe);

  printf("\n\nListe mit %d geraden Elementen: \n", GetLaenge(GListe));
  ListeAusgeben(GListe);

  printf("\n\n");
  return 0;
}
```

Kapitel 6

5. Aufgabe - Warteschlange

6.1 Einführung

Mit der fünften Aufgabe wolle wir unsere Kenntnisse über Listen noch etwas vertiefen. Doch zuvor noch ein paar Worte zum Modularisierungskonzept in C (Detailliert im Script Kapitel 7). Diesmal liegen uns drei Datei vor. Einmal die *warte.c*, unser Hauptmodul mit der main-Funktion und dann die *queues.h* die die Schnittstelle zur *queues.c* anbietet. In dieser sind alle (öffentlichen) Funktionsköpfe der queues.c aufgelistet. Dazu noch einige Präprozessoranweisungen die sich selbst erklären und das doppelte Einbinden der Header-Datei verhindert. Beim Einbinden selbst ist darauf zu achten das die include-Anweisung bei eigenen Header-Dateien im Quellverzeichnis immer in doppelte Anführungszeichen statt <> gesetzt werden.

6.2 Die Aufgabe

Unser Aufgabe ist es nun die in der queues.h definierten Funktionen in der queues.c zu implementieren. (D.h. wir verändern diesmal bloß die queues.c. Alles andere ist korrekt und fertig). Am Ende soll dabei eine Modul für eine Warteschlange heraus kommen, an die man Werte anhängen kann und auch wieder welche abfragen kann. Eine Warteschlange wird i.A. nach dem FIFO-Prinzip(First In First Out) - also wer zuerst da ist, wird als erstes bearbeitet - implementiert.

Bevor Sie sich aber jetzt wie wild auf das Programmieren stürzen, weil es sich so anhört als wäre es das selbe wie das letzte Mal, noch ein paar theoretische Betrachtungen. Nehmen wir mal an man zieht nun eine einfach verkettete Liste zur Lösung des Problem heran. Welche neuen Probleme treten dann auf? Wie verändert sich das Zeitverhalten beim Aus- und Einketten mit der Länge der Liste? Letztes mal haben wir ja gesehen, dass wir entweder zum Ein- oder Ausketten die Liste einmal komplett durchlaufen müssen, und das dauert um so länger, je länger die Liste ist.

Was liegt also näher als das Ende unserer Liste wieder an den Anfang zu „knoten“, um Anfang und Ende bei einander zu haben. D.h. wir machen aus unserer Liste einen Ring, setzen den Zeiger aber nicht auf den Anfang sondern auf das Ende. Nun haben wir unabhängig von der Länge immer den Anfang durch `Ende->next`.

Soweit so gut, bei diesem Konzept habe ich es auch in einigen Gruppe belassen. Aber eigentlich sind wir ja noch nicht fertig mit der Optimierung, wenn man bedenkt das wir vor jeden Einfügen die Funktion `voll();` aufrufen und diese ja dann wieder durch die ganze Liste geht um zu zählen ob die Schlange voll ist. Es zwingt sich nun also förmlich auf, irgendwo die aktuellen Anzahl der Elemente im Ring zu speichern. Und dazu führen wir ein zusätzliches Element ein, den Dummy, der die Anzahl speichert. Die beste Position ist hier natürlich die zwischen Ende und Anfang weil man dort wieder recht resourcenschonend ran kommt.

Aber egal für welche Möglichkeit man sich nun entscheidet(Liste, Ring oder Ring mit Dum-

my), muss man immer die jeweiligen Sonderfälle im Auge behalten. Es bietet sich auch in jedem Fall an sich dazu kleine Bildchen zu malen um den Sachverhalt immer korrekt zu erfassen. Damit können wir nun endlich zur Praxis kommen, und das heißt hier die Implementierung mit dem Ring mit Dummy.

Neue Warteschlange

Wie sieht eine leere Warteschlange als Ring mit Dummy aus? Und genau diese soll unsere Funktion zurück geben.

```
queue createqueue()
{
  queue dummy = (queue)malloc(sizeof(q_ele));    //Dummy erzeugen
  dummy->wert = 0;                               //Zähler ist 0
  dummy->next=dummy;                  //Dummy zeigt auf sich selbst.
  return dummy;                       //Zeiger auf den Dummy zurück geben
}
```

Voll oder Leer - das ist hier die Frage

Nachdem wir uns ja zu beginnt vielleicht überlegt haben, sind diese Funktionen nun recht einfach zu implementieren, denn die Anzahl der Elemente steht ja im Dummy. Und der ist immer `q->next`.

```
int leer(queue q)
{
  if (q->next->wert==0) return 1;   //q->next ist der
                                    //Dummy mit der Anzahl
  return 0;
}

int voll(queue q)
{
  if (q->next->wert>=MAXLEN) return 1;
  return 0;
}
```

Wert einfügen

Auch hier können wir wieder von unseren anfänglichen Überlegungen profitieren, da unser Ring mit Dummy beim Einfügen keine Sonderfälle hat. Es muss also zuerst wieder ein neues Element erzeugt werden und dann in den bestehenden Ring eingefügt werden[Vor dem Dummy]. (Sollte man unbedingt mal aufzeichnen!!). Wir dürfen auch nicht vergessen unseren Wert in Dummy um Eins zu erhöhen.

```
void appendqueue(queue *q, int i)
{
  queue neu = (queue)malloc(sizeof(q_ele));    //Element erzeugen
  neu->wert = i;                               //Wert rein

  ((*q)->next->wert)++;                 //Anzahl im Dummy erhöhen

  neu->next=(*q)->next;                 //neu auf Dummy zeigen lassen
```

```
  (*q)->next=neu;                    //q auf neu zeigen lassen
  *q = neu;                          //*q auf neu verschieben
}
```

Wert holen

Beim Holen des Wertes soll immer der Älteste ausgegeben werden und das ist hier immer der nach dem Dummy. Durch das Entfernen aus dem Ring wird natürlich ein Element frei. Um den Speicher dazu frei zu geben, verwenden wir die Funktion `free();`. Weiterhin müssen wir hier den Sonderfall des Entfernen des letzten Element extra behandeln.

```
void getqueue(queue *q, int *i)
{
  queue alt=(*q)->next->next;        //Zeiger auf auszukettendes Element
  ((*q)->next->wert)--;              //Anzahl in Dummy verringern
  *i=alt->wert;                      //Wert für den Referenzparameter
                                     //sichern

  if (*q!=alt)                       //Wenn q(=neu)!=alt
  {
    (*q)->next->next=alt->next;        //einfach Zeiger umlegen
  }
  else                               //sonst
  {
    *q=(*q)->next;                     //*q auf Dummy verschieben
    (*q)->next=*q;                     //und einen Ring mit sich selbst
                                       //bauen
  }
  free(alt);                         //Speicher freigeben
}
```

6.3 Zwei Lösungen

Wie gesagt können Sie die Warteschlange auf verschiedene Weisen implementieren. Hier nun noch mal die gezeigte Lösung komplett und darunter eine Lösung mit einem normalen Ring ohne Dummy.

Ring mit Dummy

```
#include <stdlib.h>
#include "queues.h"

typedef struct ele
{
  int wert;
  queue next;
} q_ele;

queue createqueue()
{
  queue dummy = (queue)malloc(sizeof(q_ele));    //Dummy erzeugen
  dummy->wert = 0;
  dummy->next=dummy;
```

```
  return dummy;
}

int leer(queue q)
{
  if (q->next->wert==0) return 1;  //q->next ist der Dummy mir der Anzahl
  return 0;
}

int voll(queue q)
{
  if (q->next->wert>=MAXLEN)  return 1;
  return 0;
}

void appendqueue(queue *q, int i)
{
  queue neu;

  neu = (queue)malloc(sizeof(q_ele));    //Element erzeugen
  neu->wert = i;
  ((*q)->next->wert)++;  //Anzahl erhöhen


  neu->next=(*q)->next;
  (*q)->next=neu;
  *q = neu;
}

void getqueue(queue *q, int *i)
{
  queue alt=(*q)->next->next;
  ((*q)->next->wert)--;  //Anzahl verringern
  *i=alt->wert;
  if (*q!=alt)  (*q)->next->next=alt->next;
  else
  {
    *q=(*q)->next;
    (*q)->next=*q;
  }
  free(alt);
}
```

Ring ohne Dummy

```
queue createqueue()
{
  return NULL;
}

int leer(queue q)
{
  if (q==NULL) return 1;
```

```
  return 0;
}

int voll(queue q)
{
  queue start=q;
  int c=0;
  if (q==NULL) return 0;
  do
  {
        q=q->next;
        c++;
  }
  while (q != start) ;

  if (c>=MAXLEN)  return 1;
  return 0;
}

void appendqueue(queue *q, int i)
{
  queue neu,help;

  neu = (queue)malloc(sizeof(q_ele));    //Element erzeugen
  neu->wert = i;
  neu->next=NULL;

  if (*q==NULL) neu->next=neu;
  else
  {
    neu->next=(*q)->next;
    (*q)->next=neu;
  }
  *q = neu;
}

void getqueue(queue *q, int *i)
{
  //*q==NULL wird mit Leer extern geprüft
  queue alt=(*q)->next;
  *i=alt->wert;
  if (*q==alt) *q=NULL;
  else (*q)->next=alt->next;
  free(alt);
}
```

Kapitel 7

6. Aufgabe - Binärbaum

7.1 Einführung

Die letzte Aufgabe für dieses Semester behandelt Binärbäume. Einführend muss man sich bloß klar machen was Binärbäume sind und wieso sich darauf so gut arbeiten lässt? Einfache Antwort: Binärbäume sind sortiert, damit kann man also recht schnell in Binärbäumen suchen. Und ansonsten auch auf Grund der Baumstruktur recht einfach rekursiv zu bearbeiten.
Die Listenstruktur aus den letzten Aufgaben wird nun noch um einen zweiten Nachfolger erweitert, und etwas umbenannt.

```
typedef struct ele *node;
typedef struct ele {
        int key;
        node left, right;
    } tree_ele;
```

7.2 Die Aufgabe

Analog zur letzten Aufgabe sollen wir eine Header-Datei vervollständigen. Die Funktionen dürften selbsterklärend sein und man sollte diese Aufgabe ohne weiter Kommentare von mir lösen können(zumal man sich ja eh auf die Prüfung vorbereitet!!).

In den Baum einfügen

Wie bereits angedeutet arbeitet man auf einen Baum am besten rekursiv und analog zu Listen muss der Sonderfall des leeren (Unter-)Baumes gesondert betrachtet werden. Ist der Baum leer wird wieder ein neues Element erzeugt und Speicher mit malloc zugewiesen. Im anderen Fall muss man überprüfen ob der Wert in den linken oder rechten Unterbaum eingefügt werden muss, und entsprechten ruft man TreeInsert wieder auf.

```
void TreeInsert(node *t, int n)
{
  node neu;
  if (*t == NULL)  // Baumzeiger zeigt auf leeren (Unter-)Baum
  {
    neu = (node)malloc(sizeof(tree_ele)); //Element erzeugen
    neu->key = n;
    neu->left = NULL;
    neu->right = NULL;
    *t = neu;  //Zeiger auf neues Element
```

```
  }
  else //sonst Baum durchlaufen, und rekursiv in den linken oder
       //rechten Teilbaum gehen
  {    //Gleichheit der Keys sollte eventuell behandlet werden!!
    if (n < (*t)->key) TreeInsert(&(*t)->left,n);
    else if (n > (*t)->key) TreeInsert(&(*t)->right,n);
  }
}
```

Anzahl, Blätter und Höhe

Drei einfache mathematische Überlegungen :)

$$Anzahl_{Baum} = \begin{cases} Anzahl_{Links} + Anzahl_{Rechts} + 1 & \text{wenn Baum existiert} \\ 0 & \text{sonst} \end{cases}$$

$$Blaetter_{Baum} = \begin{cases} Blaetter_{Links} + Blaetter_{Rechts} & \text{wenn kein Blatt} \\ 1 & \text{wenn Blatt} \\ 0 & \text{sonst} \end{cases}$$

$$Hoehe_{Baum} = \begin{cases} max(Hoehe_{Links}, Hoehe_{Rechts}) + 1 & \text{wenn Baum existiert} \\ 0 & \text{sonst} \end{cases}$$

Und nun in C übersetzt. (max habe ich hier mal mit implementiert, obwohl es in den Standardbibliotheken auch fertig existiert)

```
int Anzahl(node t)
{
  if (t == NULL) return 0;
  return 1 + Anzahl(t->left) + Anzahl(t->right);
}

int Blaetter(node t)
{
  if (t == NULL) return 0;
  else if ((t->left == NULL) && (t->right == NULL)) return 1;
  return Blaetter(t->left) + Blaetter(t->right);
}
// Gibt's schon in der stdlib.h, sollte man aber
// auch selber programmieren können
int max(int a,int b)
{
  if (a>b) return a;
  return b;
}
int Hoehe(node t)
{
  if (t == NULL) return 0;
  return 1 + max(Hoehe(t->left), Hoehe(t->right));
}
```

Die Info-Funktion ist bereits fertig und ruft nur diese drei Funktionen auf.

Ebene ausgeben

Diese Funktion soll, wie der Name sagt, eine Ebene im Baum ausgeben. Dabei soll e die auszugebende und h die aktuelle Ebene im Baum beim rekursiven Aufruf sein. Sind wir in der h-ten

Ebene geben wir also den Key aus.

```
void Ebene(node t, int e, int h)
{
  if (t != NULL)
  {
    //Aktuelle Ebene gleich auszugebende Ebene??
    if (e==h) printf("%5d",t->key);
    else
    {
      //Eine Ebene tiefer in die Teilbäume
      Ebene(t->left,e,h+1);
      Ebene(t->right,e,h+1);
    }
  }
}
```

7.3 Eine Lösung

Nun noch mal alles zusammen.

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

// Diesmal zwei "Nachfolger"
typedef struct ele {
        int key;
                node left, right;
                } tree_ele ;


int max(int a,int b)
{
  if (a>b) return a;
  return b;
}
//(Sortiert) Einfügen in einen Baum
void TreeInsert(node *t, int n)
{
  node neu;
  if (*t == NULL)  // Baumzeiger zeigt auf leeren (Unter-)Baum
  {
    neu = (node)malloc(sizeof(tree_ele)); //Element erzeugen
    neu->key = n;
        neu->left = NULL;
        neu->right = NULL;
        *t = neu;  //Zeiger auf neues Element
  }
  else
  {
    if (n < (*t)->key) TreeInsert(&(*t)->left,n);
```

```
    else if (n > (*t)->key) TreeInsert(&(*t)->right,n);
  }
}
//Ohne Worte ;)
int Anzahl(node t)
{
  if (t == NULL) return 0;
  return 1 + Anzahl(t->left) + Anzahl(t->right);
}

int Blaetter(node t)
{
  if (t == NULL) return 0;
  else if ((t->left == NULL) && (t->right == NULL)) return 1;
  return Blaetter(t->left) + Blaetter(t->right);
}

int Hoehe(node t)
{
  if (t == NULL) return 0;
  return 1 + max(Hoehe(t->left), Hoehe(t->right));
}

void Info(node t)
{
  printf("\nDer Baum hat %d Knoten ( %d davon sind Blaetter) \
 und eine Hoehe von %d\n\n", Anzahl(t), Blaetter(t), Hoehe(t));
}

void Ebene(node t, int e, int h)
{
  if (t != NULL)
  {
    //Aktuelle Ebene gleich auszugebende Ebene??
        if (e==h) printf("%5d",t->key);
    else
    {
      //Eine Ebene tiefer in die Teilbäume
      Ebene(t->left,e,h+1);
          Ebene(t->right,e,h+1);
        }
  }
}
```