Praktikum Informatik I für VIW
Tutorial zum Praktikum WS2009/10

Martin Heinzerling - tud@mheinzerling.de

12. Februar 2010
Technische Universität Dresden

Zusammenfassung

Das hier angebotene Material ist kein Originalmaterial der Lehrbeauftragten. Es wurde nach besten Wissen und Gewissen durch meine Person zusammengestellt. Aus offensichtlichen Gründen kann ich 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 das 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/InfIViw/`

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.

Wir werden die Aufgaben mit DevC++ bearbeiten. Dieses Tool gibt es frei unter:

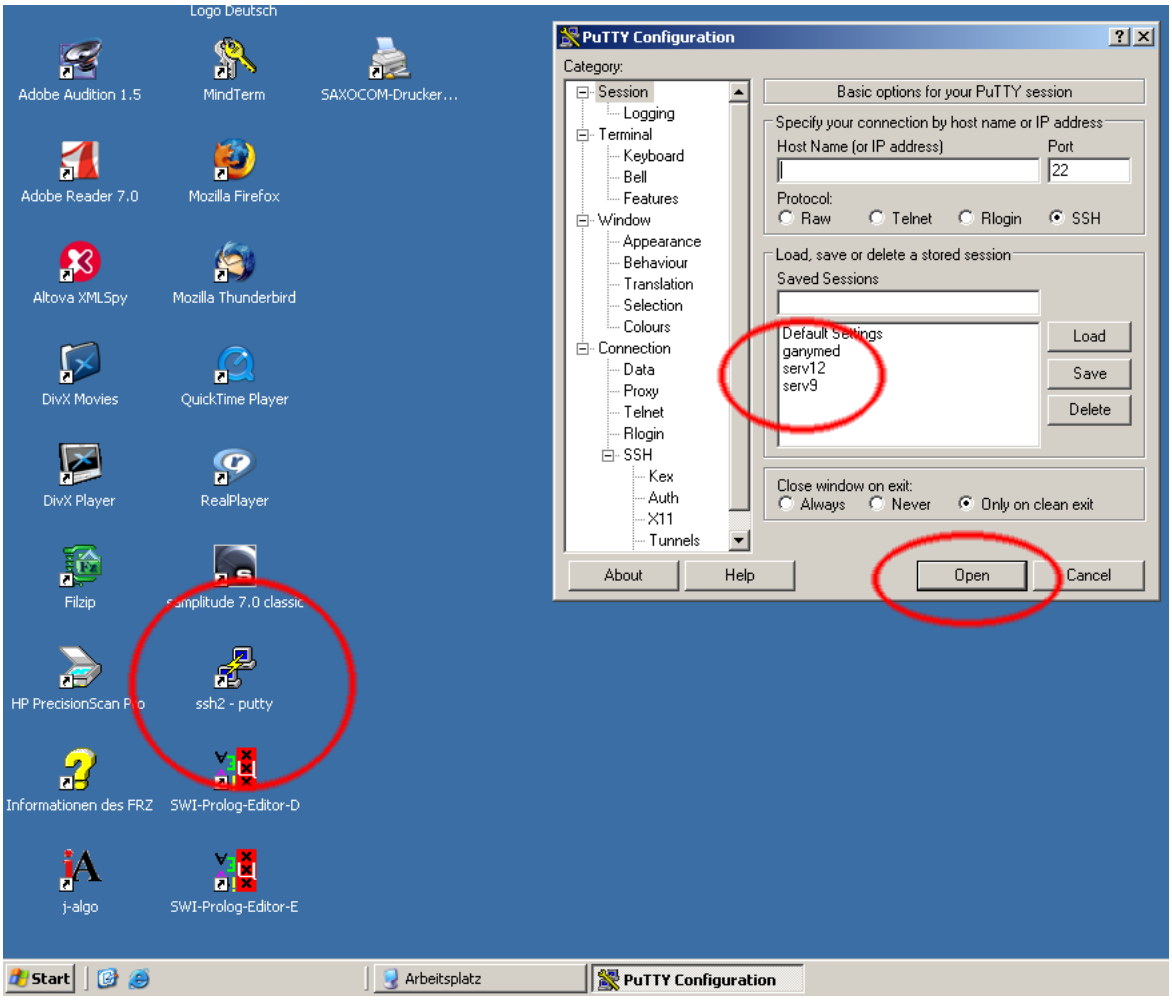
`http://www.bloodshed.net/devcpp.html`

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

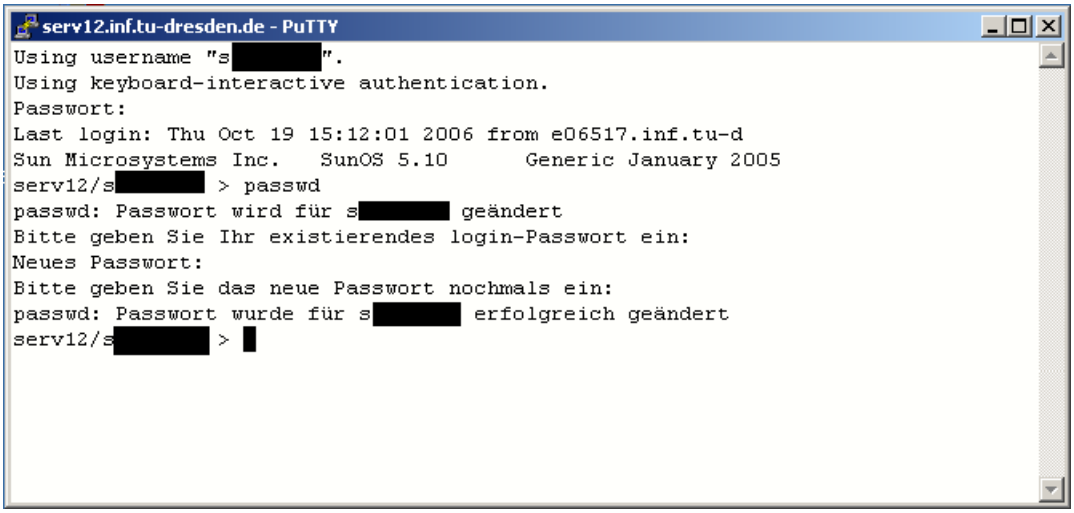
1.2 Passwort ändern

Bevor wir beginnen sollten Sie Ihr Erstpasswort ändern. ACHTUNG das neue Passwort gilt nur im ehemaligen Rechenzentrum der Fakultät Informatik, in allen Anderen gilt weiterhin das Passwort des ZIH (Zentrum für Information und Hochleistungsrechner).

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



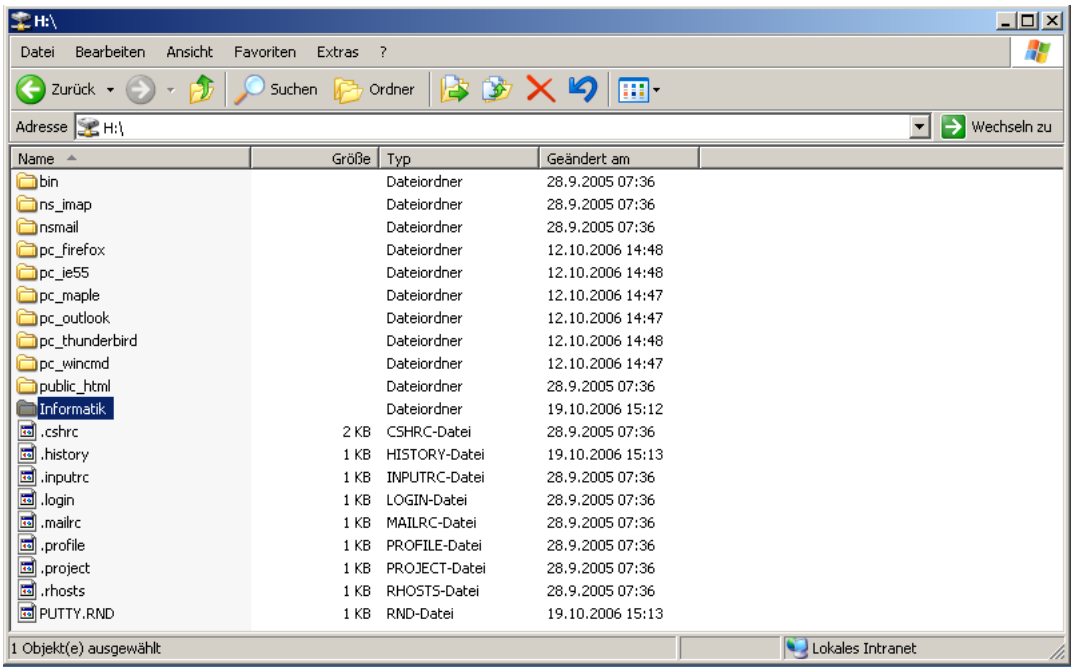
Daraufhin erscheint ein Konsolenfenster in dem Sie aufgefordert werden Ihr aktuelles Passwort einzugeben. Während der Eingabe erscheinen keine Sterne wie unter Windows. Das Passwort wird blind eingegeben. Nach erfolgreicher Anmeldung wird mit dem Befehl `passwd` das Passwort geändert. Geben Sie ein: `passwd` <ENTER>. Anschließend werden Sie erneut zur Eingabe Ihres Passwortes aufgefordert, sowie zur zweimaligen Eingabe Ihres neuen Passwortes. Jede Eingabe schließen Sie mit <ENTER> ab.



Nach Bestätigung der Änderung können Sie das Fenster schließen. Geben Sie ein `exit` <Enter> .

1.3 Homeverzeichnis

Als Nutzer des Rechnernetzes wurde Ihnen auch ein Homeverzeichnis für Ihre Daten bereitgestellt. Dieses können Sie mit dem *Arbeitsplatz* oder *Windows Explorer* als Laufwerk *H:* erreichen. Auf Laufwerk *H:* erstellen Sie einen Ordner *Informatik* (*Datei* → *Neu* → *Ordner*) in dem alle zukünftigen Daten gespeichert werden können.



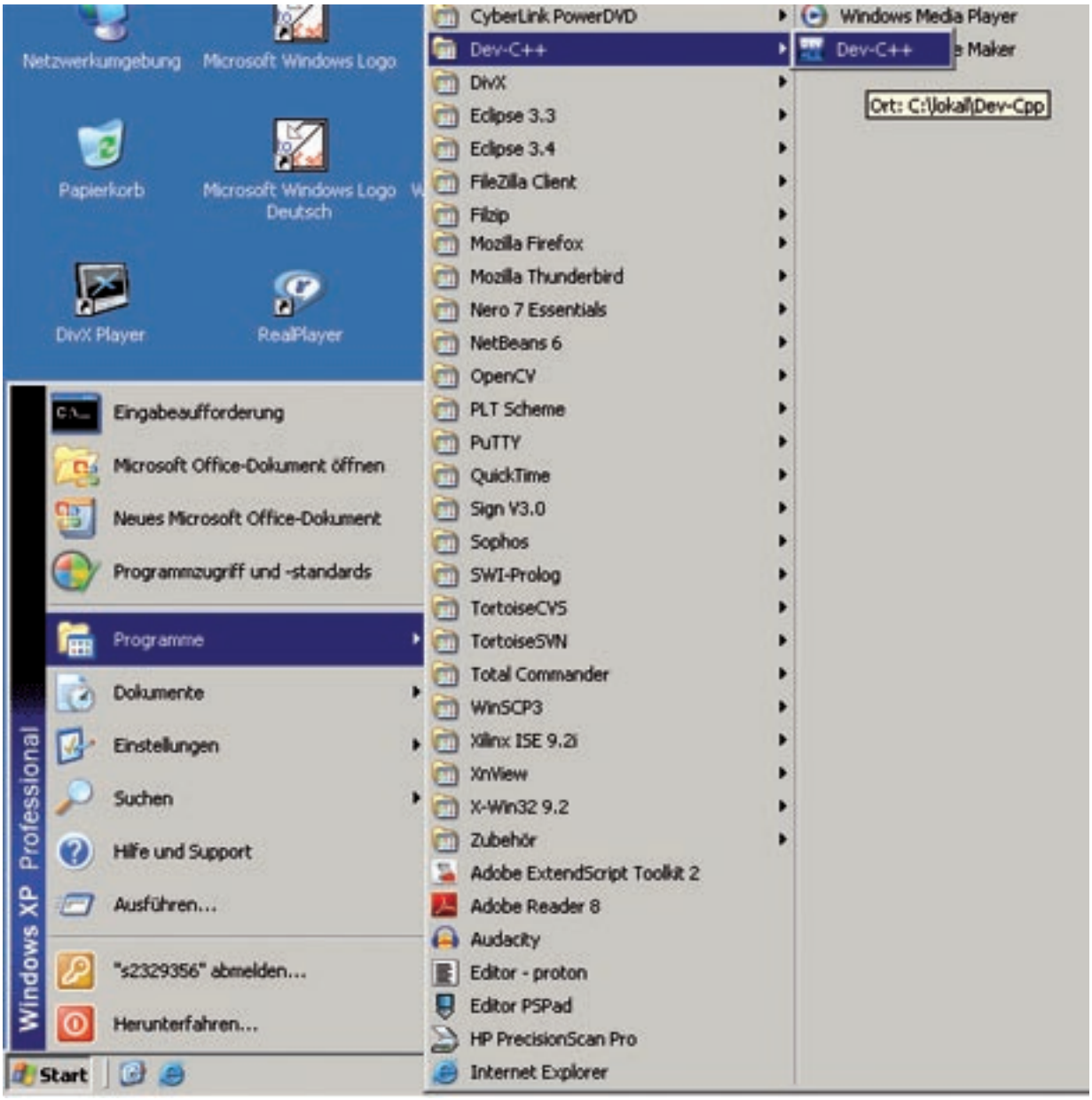
1.4 E-Mail-Weiterleitung

Mit der Immatrikulation haben Sie auch eine E-Mail-Adresse der Form *s0000000@mail.zih.tu-dresden.de* erhalten. Wenn Sie diese nicht regelmäßig nutzen, sollten Sie eingehende E-Mails auf Ihre private E-Mail-Adresse weiterleiten, da gelegentlich über diese Adresse wichtige Informationen versendet werden.

Loggen Sie sich dazu unter *https://mail.zih.tu-dresden.de/horde/* mit Ihren Zugangsdaten ein. Am oberen Rand finden Sie nun die Schaltfläche *Filter*. Nach einem Klick darauf, ändert Sich die obere Navigationsleiste und die Schaltfläche *Weiterleitung* erscheint. In dem folgenden Textfeld können Sie nun Ihre private E-Mail-Adresse eintragen und mit einem Klick auf *Speichern* bestätigen. Danach können Sie sich oben rechts wieder abmelden.

1.5 „Hello World“

Beginnen wir mit einem einfachen und dem Standard-Beispiel. Öffnen Sie zunächst den *Bloodshed Dev-C++* indem Sie auf **Start** in der Taskleiste klicken. Wählen Sie dann *Programme*, *Dev-C++* und nochmals *Dev-C++*.



Sobald die Entwicklungsumgebung gestartet ist, erstellen Sie über *Datei*, *Neu* und *Quelldatei* eine neue Datei.

Bevor es weiter geht, sollten Sie die Datei speichern. **Achten Sie dabei darauf, dass Sie Ihre Dateien immer in ihrem Homeverzeichnis speichern.** Anderfalls werden die Dateien automatisch wieder gelöscht und stehen das nächste mal nicht mehr zur Verfügung.

Der nötigen Code für unser erstes Programm sieht wie folgt aus:

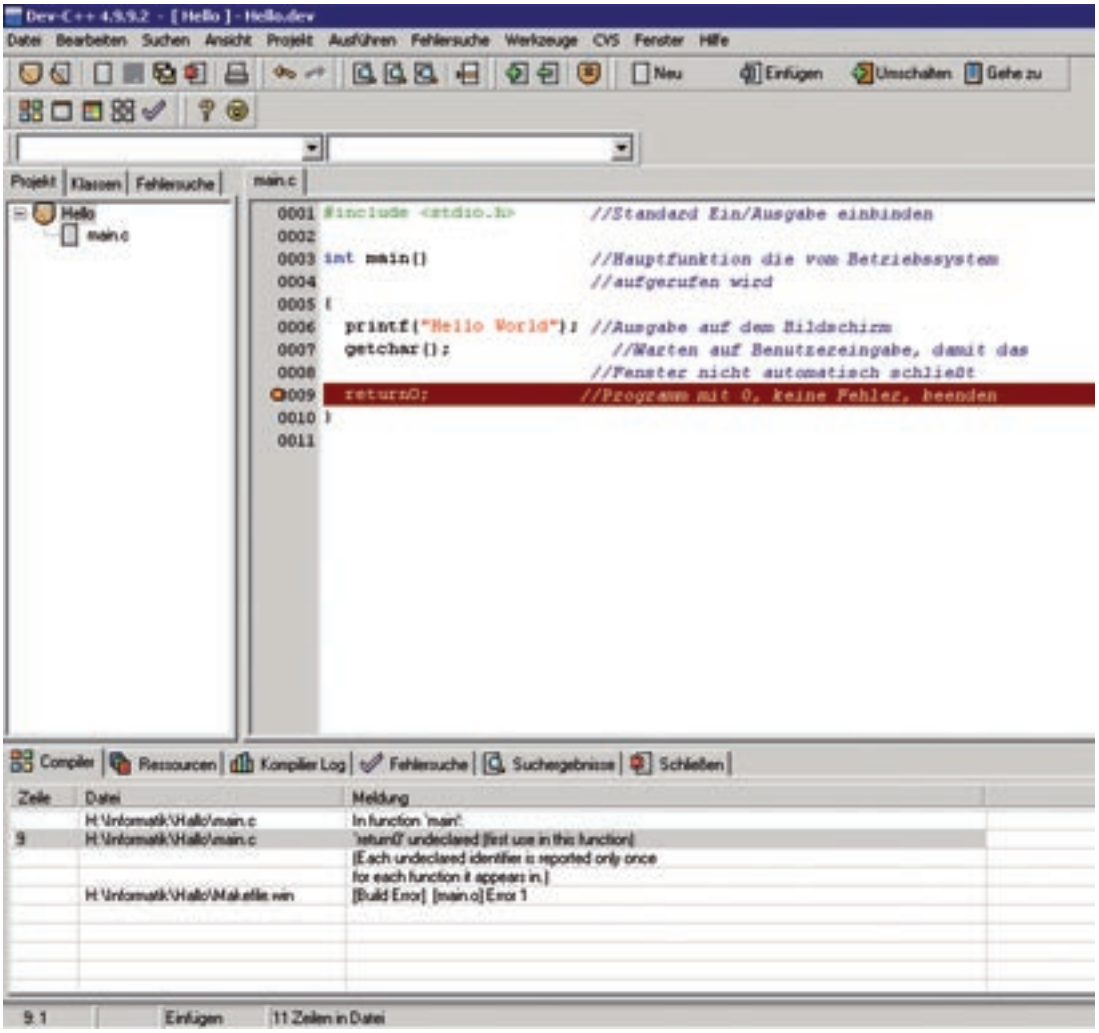
```
#include <stdio.h>          //Standard Ein/Ausgabe einbinden

int main()                  //Hauptfunktion die vom Betriebssystem
                            //aufgerufen wird
{
```

```
  printf("Hello World"); //Ausgabe auf dem Bildschirm
  getchar();              //Warten auf Benutzereingabe, damit das
                          //Fenster nicht automatisch schließt
  return 0;               //Programm mit 0, keine Fehler, beenden
}
```

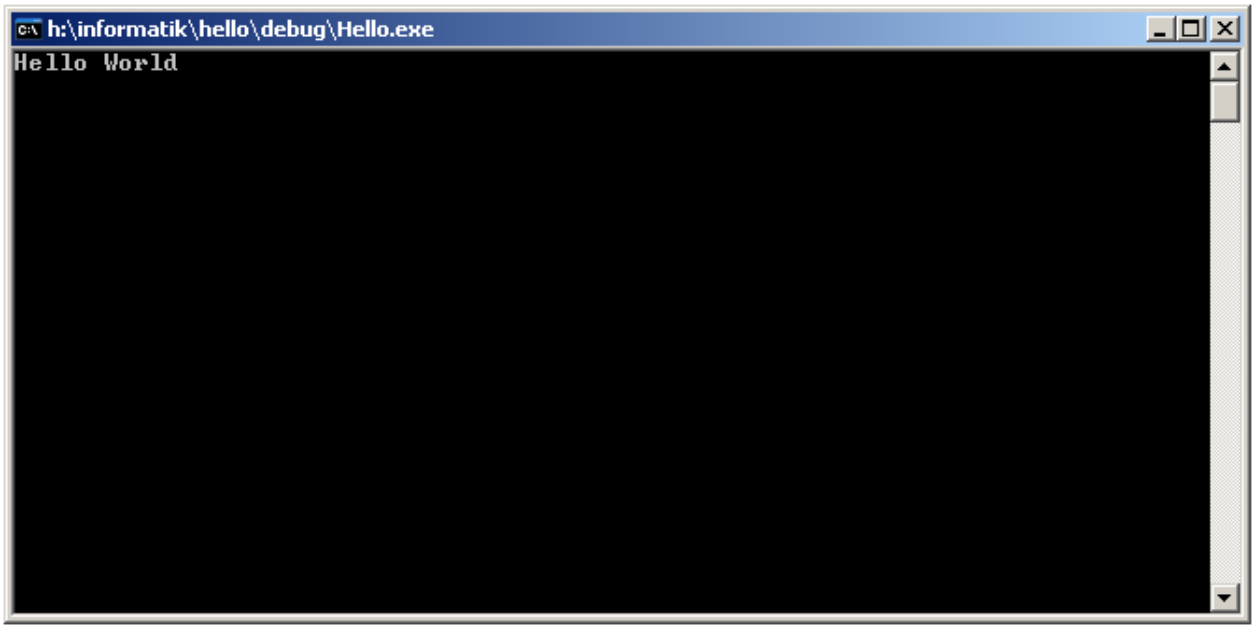
Anschließend drücken Sie F9 um das Programm auszuführen. Dev-C frag dann zunächst nochmal ob die Datei gespeichert werden sollen, was bestätigt werden kann.

Wer jetzt kein schwarzes Fenster mit dem Text Hello World sieht, hat erstmal was falsch gemacht ;).

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 F9.



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

Wenn man bereits eine Quelldatei besitzt, wie in einigen der nächsten Aufgaben, kann diese über *Datei* und *Projekt oder Datei öffnen* direkt geöffnet werden.

Die Zeilennummern können Sie über *Werkzeuge*, *Editor Optionen*, **Anzeige** und den Haken *Zeilennummern* aktivieren.

Kapitel 2

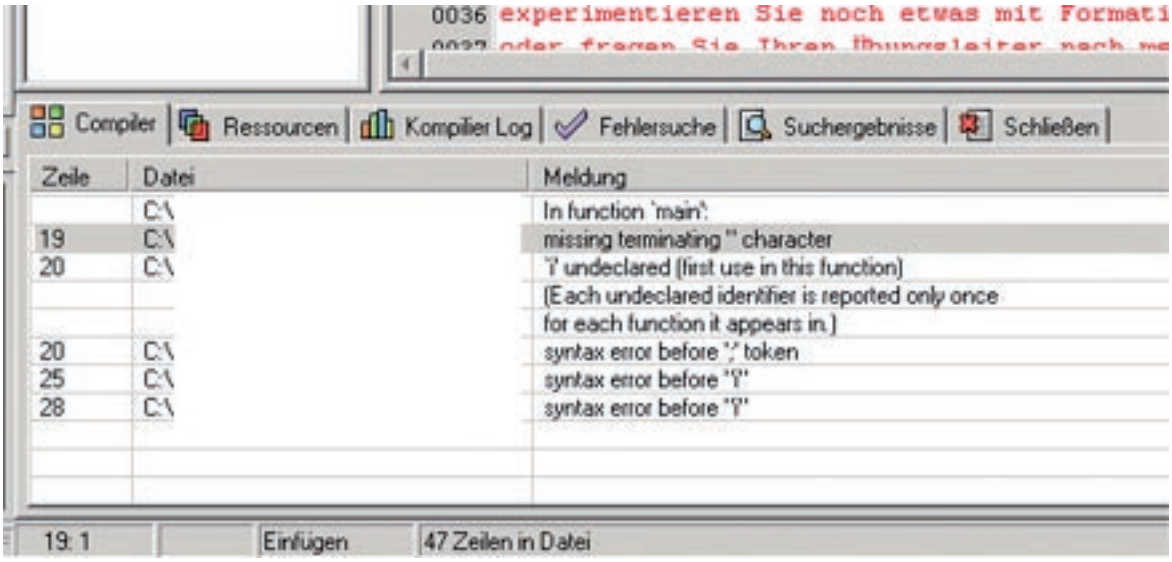
1. Praktikum - Fehler finden

2.1 Datei speichern und öffnen

Öffnen Sie `http://www.orchid.inf.tu-dresden.de/gdp/lehre/InfIViw/` 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\`. Öffnen Sie nun DevC++ wie gewohnt. Wählen Sie anschließend *Datei → Projekt oder Datei öffnen...* . Im folgenden Dialog suchen und öffnen Sie die *aufgabe1.c*. Auf ein Projekt können wir bei nur einer zu bearbeitenden Datei verzichten.

2.2 Aufgabe 1

Lesen Sie sich nun erst einmal die Aufgabe in der Quelldatei durch. Versuchen Sie die Fehler zu finden. Dazu können Sie das Programm mit F9 starten und sich Fehlermeldungen ausgeben lassen. Gehen Sie dabei der Reihe nach von Oben nach Unten vor um alle Fehler zu finden und starten Sie dazwischen immer mal ihr Programm 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 7 Fehler und 3 mögliche Probleme enthalten.

Art	Zeile		
Fehler	19	fehlende “	Strings werden mit “ geöffnen und auch wieder geschloßen
Fehler	19 → 07	I → i	C unterscheidet Groß- und Kleinschreibung. Im nachfolgenden wird immer ein kleines i verwenden, groß wäre aber auch möglich
Fehler	19 → 06	c → ch	ch nicht definiert; c nicht verwendet
Fehler	25	if (..)	If Bedingung gehören in Klammern
Fehler	25	<> → !=	Ungleich wird durch != ausgedrückt
Fehler	28	if ( )	Wieder Klammern, und auch == wenn man vergleichen will, hier wird nur die Zuweisung überprüft
Fehler	35	printf	Standardfunktionen sind alle klein geschrieben

Damit hätten Sie die syntaktischen Fehler der ersten Aufgabe behoben.

Als weitere Fragestellung können Sie sich mal Gedanken machen warum Zeile 8, 9 und 20 zu Problemen führen könnte. Weiterhin ist interessant zu untersuchen warum das „e“ in der alphabetischen Ausgabe klein ist und was an dieser Stelle eigentlich wirklich passieren sollte.

2.3 Aufgabe 2

Die zweite Aufgabe ist bereits aus der Übung bekannt und soll nur noch einmal in Dev-C++ umgesetzt werden um sich mit den Arbeitsabläufe vertraut zu machen.

Kapitel 3

2. Praktikum - Etwas Mathematik

Das zweite Praktikum greift nochmals die Übersetzung von Struktogrammen auf und stellt so auch den Bezug zur Übung dar.

Zum leichteren Verständnis sollte die Aufgaben 1b erst nach einen Blick in die Aufgabe 3 angegangen werde.

3.1 Aufgabe 1

Im Teil a ist wie gewohnt ein Struktogramm zu übersetzen, dies sollte mittlerweile für alle ohne Schwierigkeiten lösbar sein, so dass hier nur eine Lösung angeben wird. (Je nach Ausgangsstruktogramm kann die Lösung etwas variieren)

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

int main()
{
    int s, s1, s2, i, n=0;

    printf("Welche Fib.zahl moechten Sie berechnen: ");
    do
    {
      if (n<0) printf("==> Geben Sie eine positive Zahl an: ");
      scanf("%d", &n);
    }while (n<0);

    s1=1;
    s2=1;

    if (n<2) s=1;
    else
    {
        for(i=2;i<=n;i=i+1)
        {
            s=s1+s2;
            s1=s2;
            s2=s;
        }
    }

    printf("Die %d-te Fib.zahl ist %d.",n,s);
```

```
    return 0;
}
```

Für den Teil b können wir uns die dritte Aufgabe zur Hilfe nehmen und finden darin den folgenden Konstrukt um einzelne Zeichen von der Tastatur zu lesen.

```
char auswahl;
//...
auswahl=getch();
auswahl=toupper(auswahl);   // aus Kleinbuchstabe wird Grossbuchstabe
```

Kombiniert mit der ersten Aufgabe und der Anforderung das Programm beliebig oft wiederholen zu können (also einer Schleife), folgt somit das folgendes Programm.

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

int main()
{
    int s, s1, s2, i, n=0;
    char ch=' ';

    do
    {
        system("cls");
        printf("Welche Fib.zahl moechten Sie berechnen: ");
        do
        {
          if (n<0) printf("==> Geben Sie eine positive Zahl an: ");
          scanf("%d", &n);
        }while (n<0);

        s1=1;
        s2=1;

        if (n<2) s=1;
        else
        {
            for(i=2;i<=n;i=i+1)
            {
                s=s1+s2;
                s1=s2;
                s2=s;
            }
        }

        printf("Die %d-te Fib.zahl ist %d.",n,s);

        printf("\n\nMoechten Sie noch eine Zahl berechnen.
                    (J)a oder (N)ein\n");
        ch=toupper(getch());
    }
    while (ch=='J');
```

```
    return 0;
}
```

3.2 Aufgabe 2

Auch die zweite Aufgabe orientiert sich wieder an einen Struktogramm und erübrigt somit weitere einleitende Worte. Ein Lösung könnte wie folgt aussehen.

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

int main()
{
    float bas, oldbas, erg=1.0f;
    int exp, oldexp;

    printf("Geben Sie die Basis an: ");
    do
    {
      if (bas<0) printf("==> Geben Sie eine positive Zahl an: ");
      scanf("%f", &bas);
    }while (bas<=0);

    printf("Geben Sie den Exponent an: ");
    do
    {
      if (exp<0) printf("==> Geben Sie eine positive Zahl an: ");
      scanf("%i", &exp);
    }while (exp<0);
    oldexp=exp;
    oldbas=bas;
    while (exp>0)
    {
        if (exp%2) erg=erg*bas;
        exp=exp/2;
        bas=bas*bas;
    }

    printf("%f ^ %d = %f",oldbas,oldexp,erg);

    getch();
    return 0;
}
```

An dieser Stelle sei auch nochmals darauf hingewiesen, dass es eigentlich unendlich viele richtige Lösungen gibt. Auch das Speichern der eingegebenen Werte für die später Ausgabe ist eher ein optionales Feature, was lediglich einer besseren Benutzbarkeit des Programms dienen soll.

3.3 Aufgabe 3

In der *aufgabe3.c* ist ein bereits lauffähiges Programm enthalten. Hier sind nur noch einige Lücken für die konkrete Berechnung auszufüllen. Viel wichtiger in dieser Aufgabe ist aber die Konstruktion des Menüs mit den Operationen die ein Benutzer ausführen kann. Bitte beschäftigen Sie sich damit, bis die Funktionsweise klar ist.

An den entsprechenden Stellen sind die folgenden Berechnung einzusetzen und die führenden Kommentarzeichen zu entfernen.

```
oberflaeche=2*M_PI*radius+(radius+hoehe);
//...
volumen=M_PI*radius*radius*hoehe;
//...
gewicht=M_PI*radius*radius*hoehe*dichte;
```

Kapitel 4

3. Praktikum - Zylinder Teil II

4.1 Einführung

Ziel dieses Praktikums 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 Parameter die Variable enthält in die die Eingabe gespeichert werden soll. Und da haben wir ja gelernt, dass wir dies beim Aufruf mit

dem Referenzoperator & machen. Auf der anderen Seite, also der Funktion, müssen wir dies ebenfalls durch den Derefenzierungsoperator * kennzeichnen. 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
```

4.2 Die Aufgabe

Es geht mit dem Programm vom letzten Mal weiter. Alle Menüpunkte sollen in Funktionen ausgelagert werden.

4.3 Eine Lösung

Die folgende Lösung zeigt die Umsetzung für die Oberfläche. Die anderen beiden Berechnung sind analog um zubauen. Die Funktion für die Eingabe sieht etwas anders aus, da wir Referenzparameter verwenden müssen.

```
float oberf(float r, float h)
{
  return 2*M_PI*r+(r+h);
}

void eingabe(float *hoe, float *rad)
{
        printf("\nHoehe? :");
        scanf("%f",&(*hoe));
        printf("\nRadius?:");
        scanf("%f",rad);  //& und * sind inverse Operationen,
                          //heben sich hier auf und können
                          //auch weg gelassen werden
}
```

```
int main()
{
...
  eingabe(&hoehe, &radius);

  while(ende==0)
  {
...
    oberflaeche=oberf(radius, hoehe);
...
  }
}
```

Kapitel 5

4. Praktikum - Feld

5.1 Einführung

Bevor wir uns der Aufgabe zuwenden, müssen wir zunächst unsere Grundlagen etwas erweitern.

In der Aufgabe wird gleich zu Beginn eine symbolische Konstanten definiert. Diese werden 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 und Dimension des Arrays.

```
int feld[MAX]       // 25 Elemente
int arr[4][2][5]; //Dimension ist dann 3 mit insgesamt
                  //40 Zellen vom Typ int
```

Wir verwenden diesmal nur eindimensionale Arrays für die Verwaltung der Zufallszahlen. Die Vorteile der Verwendung von Arrays dürften offensichtlich sein, man benötigt nur eine Variable und nicht 25 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 feld[3]* hat die Elemente *feld[0]*, *feld[1]* und *feld[2]*, und diese können jetzt wie „normale“ Variablen verwendet werden. Vorerst wollen wir erstmal davon ausgehen das die Länge von Arrays immer konstant ist und nicht verändert werden kann. Gibt man einen Index außerhalb der Grenzen an, bekommt man einen Fehler.

Um Felder an Funktionen zu übergeben geht man ähnlich wie mit normalen Variablen vor. Der Aufruf bleibt unverändert und die Deklaration wird um die *[]* für das Feld ergänzt. Felder werden aber im Gegensatz zu den bisherigen Variablen nicht als Kopie sondern als Referenz (also Original) übergeben, d.h. Änderung am Feld innerhalb der Funktion wirken sich auch auf das Feld in der aufrufenden Methode aus.

```
...
void machwas(int f[]);

int main()
{
  int feld[MAX];
  machwas(feld);
```

```
}
...
```

Um Zufallszahlen in C erstellen zu lassen, benötigen wir die Bibliothek *time.h* und *stdlib.h*. Vor der Verwendung der Zufallszahlen, müssen wir den Generator erst mit `srand((unsigned int) time(NULL));` initialisieren. Und mit `int a=rand();` können wir ganz einfach Zufallszahlen erhalten. Zu beachten ist hier, das man den Generator nur einmal am Programmanfang initialisieren darf, sonst erhält man immer wieder die selben Zufallszahlen.

Mit dem nun erfolgreich erweiterten Grundlagen sollte die Aufgabe ein Leichtes sein. Öffnen Sie zunächst wieder eine neue Datei und kopieren die Grundstruktur aus von der Website hinein.

5.2 Die Aufgabe

Für den Einstieg hier zunächst die erste Funktion. Analog dazu können dann die Ausgabe und die drei Berechnungen aufgebaut und anschließen aus der main-Funktion sinnvoll aufgerufen werden.

```
void zufall(int f[])
{
  int i;
  for (i=0;i<MAX;i++)
  {
    f[i]=rand()%100+1;   //1..100
  }
}
```

5.3 Eine Lösung

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

#define MAX 25

void zufall(int f[])
{
  int i;
  for (i=0;i<MAX;i++)
  {
    f[i]=rand()%100+1;
  }
}
void ausgabe(int f[])
{
  int i;
  for (i=0;i<MAX;i++)
  {
    printf("%d ",f[i]);
  }
}
int summe(int f[])
```

```
{
  int i,s=0;
  for (i=0;i<MAX;i++)
  {
    s=s+f[i];
  }
  return s;
}
float mw(int f[])
{
  return (float)summe(f)/(float)MAX;
}

int max(int f[])
{
  int i,m=0;
  for (i=0;i<MAX;i++)
  {
    if (f[i]>m) m=f[i];
  }
  return m;
}

int main()
{
  int feld[MAX];
  srand((unsigned int) time(NULL));
  zufall(feld);
  ausgabe(feld);
  printf("\nSumme: %d",summe(feld));
  printf("\nMW: %.2f",mw(feld));
  printf("\nMaximum: %d",max(feld));
  getch();
  return 0;
}
```

Kapitel 6

5. Praktikum - Matrix

6.1 Einführung

Diese Aufgabe erfordert keine neues Wissen über C, sondern soll eher das herangehen an ein Problem verdeutlichen.

6.2 Die Aufgabe

```
#include <stdio.h>

#define ZEILEN 5
#define SPALTEN 4

int feld[ZEILEN][SPALTEN];
int i,j,su;
int zei_summe[ZEILEN], spa_summe[SPALTEN];

/* Entwickeln Sie ein C-Progamm, das folgendes leistet:

   Als Werte fuer die Elemente des zweidimensionalen ARRAY's 'feld'
   sollen jeweils die Summen der Laufindices 'i' (Zeilen) und
   'j' (Spalten) eingetragen werden.
   Anschliessend sind die Zeilen- und Spaltensummen zu berechnen
   und das Ganze in folgender Form (ohne Indices) auf dem Bildschirm
   auszugeben:

       j -->

   i   feld[0][0]    feld[0][1]    .....     zei_summe[0]

   |   feld[1][0]    .....         .....     .....
   |
   v   .....         .....         .....     .....


       spa_summe[0]  .....
*/
```

Die Aufgabe können Sie ja der Datei entnehmen. Versuchen Sie zunächst das Problem zu erfassen. Entwickeln Sie dabei eine Lösung unabhängig von C. Wie würden Sie das Problem

auf dem Papier lösen? (Erstmal denken und nicht weiter lesen ;))

Alle die jetzt einen Algorithmus im Kopf haben, sollten probieren ihn umzusetzen und nicht weiter lesen. Für alle anderen ein kleiner Hinweis. Dies ist ein recht einfach zu erfassender Algorithmus und auf keinen Fall der Beste oder **der** Richtige. Es gibt immer verschiedene Wege die zum Ziel führen.

- Zelleninhalte ausrechnen
- Reihensumme ausrechnen
- Spaltensumme ausrechnen
- Zelleninhalte und Reihensumme ausgeben
- Spaltensumme ausgeben

An dieser Stelle sollten wieder einige aufhören zu lesen und anfangen zu programmieren. Wer sich dazu noch nicht bereit fühlt liest einfach weiter. Solche Algorithmen zu finden hat einiges mit Praxiserfahrung zu tun und nur die wenigsten stolpern einfach so über einen guten Algorithmus. Splitten wir es nun also etwas auf:

- Zelleninhalte ausrechnen
  1. Starte in Reihe 1 (bzw. 0)
  2. Starte in Spalte 1 (bzw. 0)
  3. Speichere Nummer der Reihe + Nummer der Spalte in der Zelle[Reihe][Spalte]
  4. Gehen eine Spalte weiter und wiederhole 3. und 4. bis zum Ende der Reihen
  5. Gehen eine Reihe weiter und beginne wieder mit Schritt 2.
  6. Wenn alle Reichen und Spalten durch sind gehe weiter
- Reihensumme ausrechnen
  - Analog zu den Zelleninhalten können wir nun wieder 2 Schleifen laufen lassen die die Reihensumme ausrechnet
- Spaltensumme ausrechnen
  - Analog zu den Zelleninhalten können wir nun wieder 2 Schleifen laufen lassen die die Spaltensumme ausrechnet
- Zelleninhalte und Reihensumme ausgeben
  - Analog zu den Zelleninhalten können wir nun wieder 2 Schleifen laufen um die Ergebisse Ausgeben zu lassen
  - Nach jeder Reihe geben wir nun nur noch die Reihen-Summe an
- Spaltensumme ausgeben
  - Hier reicht dann eine Schleife die uns nur noch mal die Spalten-Summen unter unsere Matrix schreibt

Ich hoffe jetzt ist jeder soweit sich an dem Programm zu probieren.

6.3 Eine Lösung

Wie gesagt, dies ist eine Lösung und auch nicht die Beste. Es ist bloß die am einfachsten nachvollziehbare.

```
#include <stdio.h>

#define ROWS 10
#define COLS 8

int feld[ROWS][COLS];
int i,j,su;                          // su brauchen wir eigentlich gar nicht
int row_sum[ROWS], col_sum[COLS];


int main(void)
{
                                     //Zellen ausrechnen
  i=0;
  while (i<ROWS)                     //Reihen durchlaufen
  {
    j=0;
    while (j<COLS)                   //Spalten durchlaufen
    {
      feld[i][j]=i+j;                //Feld die Summe zuweisen
      j++;
    }
    i++;
  }
  //Reihensumme ausrechnen
  i=0;
  while (i<ROWS)
  {
    row_sum[i]=0;
    j=0;
    while (j<COLS)
    {
      row_sum[i]+=feld[i][j]; //Zur Reihensumme addieren
                                     //kurz für:
                                     //row_sum[i]=row_sum[i]+feld[i][j];
      j++;
    }
    i++;
  }

  //Spaltensumme ausrechnen
  j=0;
  while (j<COLS)
  {
    col_sum[j]=0;
    i=0;
    while (i<ROWS)
    {
```

```
      col_sum[j]+=feld[i][j]; //Zur Reihensumme addieren
                              //kurz für:
                              //col_sum[j]=col_sum[j]+feld[i][j];
      i++;
    }
    j++;
  }
//Felder und Reihensumme ausgeben
  i=0;
  while (i<ROWS)
  {
    j=0;
    while (j<COLS)
    {
      printf("%5d",feld[i][j]); //printf mit Formatierung
                                //Integerzahl der Länge 5 ohne
                                //führende Nullen
                                //sieht einfach nur gut aus
      j++;
    }
    printf("%5d\n",row_sum[i]);
    i++;
  }
                                //Spaltensumme ausgeben
  j=0;
  while (j<COLS)
  {
    printf("%5d",col_sum[j]);
    j++;
  }
                                //Fertig - Neue Zeile und Warten
  printf("\n");
  getch();
  return 0;
}
```

Jeder Programmierer würde bei dem Code zwar einen Nervenzusammenbruch erleiden, aber es funktioniert ja :P. Was einem als Anfänger nun doch etwas stören sollte, ist die Tatsache, dass wir 3-mal die selbe Schleife ausführen und diese sich untereinander nicht benötigen, also kein Zwischenergebnisse benötigen. Und bei den anderen Beiden können wir auch noch etwas reduzieren. Also packen wir sie einfach mal zusammen:

```
#include <stdio.h>

#define ROWS 10
#define COLS 8

int feld[ROWS][COLS];
int i,j,su;                     // su brauchen wir eigentlich gar nicht
int row_sum[ROWS], col_sum[COLS];


int main(void)
```

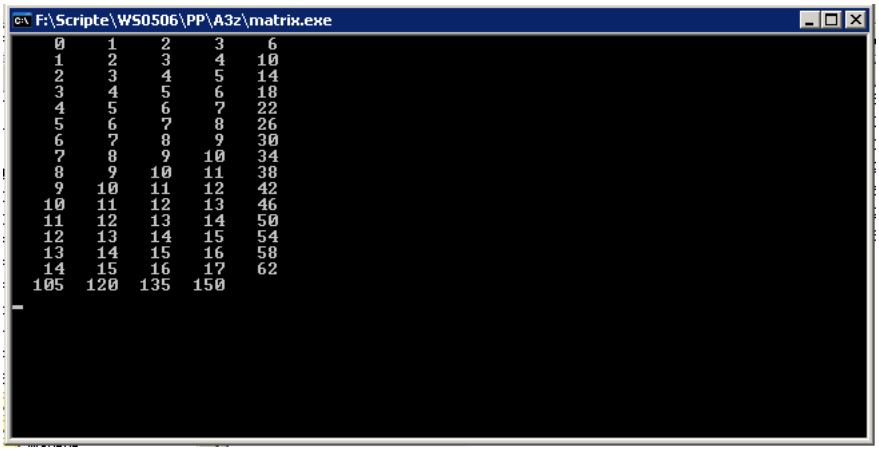
```
{

  i=0;
  while (i<ROWS)                    //Reihen durchlaufen
  {
    row_sum[i]=0;
    j=0;
    while (j<COLS)                  //Spalten durchlaufen
    {
      feld[i][j]=i+j;               //Feld die Summe zuweisen
      row_sum[i]+=feld[i][j];       //Zur Reihensumme addieren
      printf("%5d",feld[i][j]);//Ausgeben
      j++;
    }
    printf("%5d\n",row_sum[i]);
    i++;
  }

  j=0;
  while (j<COLS)
  {
    col_sum[j]=0;
    i=0;
    while (i<ROWS)
    {
      col_sum[j]+=feld[i][j]; //Zur Reihensumme addieren
                              //kurz für:
                              //col_sum[j]=col_sum[j]+feld[i][j];
      i++;
    }
    printf("%5d",col_sum[j]);//Spaltensumme ausgeben
    j++;
  }

//Fertig - Neue Zeile und Warten
  printf("\n");
  getch();
  return 0;
}
```

So einfach etwas zusammen zu schieben geht zwar selten, aber hier bietet es sich natürlich an. Unter Verwendung von for-Schleifen und einigen kleinen Umstrukturierungen würde unser Code nun wie folgt aussehen:

```
#include <stdio.h>

#define ROWS 15
#define COLS 4

int feld[ROWS][COLS];
int i,j;
int row_sum[ROWS], col_sum[COLS];
```

```
int main(void)
{
  for (j=0;j<COLS;j++) col_sum[j]=0; //Wir können nicht davon ausgehen,
                                     //dass Variablen mit 0 initiiert werden
  for (i=0;i<ROWS;i++)
  {
    row_sum[i]=0;
    for (j=0;j<COLS;j++)
    {
      feld[i][j]=i+j;
      row_sum[i]+=feld[i][j];
      col_sum[j]+=feld[i][j];
      printf("%5d",feld[i][j]);
    }
    printf("%5d\n",row_sum[i]);
  }
  for (j=0;j<COLS;j++)   printf("%5d",col_sum[j]);

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

Mit dem letzten Teil sind wir zwar etwas über das Ziel hinaus geschossen, aber wer mir bis hierher folgen konnte, steht sehr gut im Stoff und wird auch mit dem weiteren Aufgaben kaum Probleme haben. Alles in Allem sieht es dann immer so aus.



Kapitel 7

6. Praktikum - Lotto

7.1 Einführung

Diesmal haben wir wieder eine komplexere Aufgabenstellung ohne grundlegend neue Konzepte in C. Die Zufallszahlen können ggf. in Aufgabe 4 nochmals nachgelesen werden.

7.2 Die Aufgabe

Schwerpunkt, neben einer durchdachten Strukturierung des Programms, liegt diesmal im Abfangen sämtlicher Fehleingaben die ein Benutzer machen könnte. Sollte noch Zeit und Motivation vorhanden sein, kann man sein Augenmerk auch auf ein ansprechendes Layout legen.

7.3 Eine Lösung

Die Lösung ist natürlich wieder etwas „schöner“ als nötig. Auch die farbige Ausgabe ist nur ein Feature um zu zeigen, dass es nicht immer Grau auf Schwarz sein muss. Wie man die Funktionen auslagert, kann ebenso variieren. Mittlerweile müsstet ihr an einem Punkt sein, wo ihr den Quellcode ohne Schwierigkeiten mit Hilfe der Kommentare lesen könnt.

```
#include <stdio.h>
#include <time.h> //Zufall
#include <windows.h> //Farbe

//Bildschirm löschen und Kopf malen
void head()
{
  //Bildschirm löschen - nur unter Windows!!!
  system("CLS");
  printf("==========================================\n");
  printf("=              L O T T O               =\n");
  printf("==========================================\n");
  printf("* Chance 1:14 Millionen. \n");
  printf("  Spielteilnahme ab 18 Jahren.\n");
  printf("  Gluecksspiel kann suechtig machen.\n\n");
}
//Zahlenarray ausgeben
void print(int i[])
{
  int j;
  for (j=0;j<50;j++)
```

```
    if (i[j]==1)
      printf(" %d",j);
}
//Eingabe
void getInput(int i[])
{
  int x=0,j;
  for (j=0;j<50;j++) i[j]=0;
  head();
  printf("Geben Sie nun bitte Ihren Tip an:");
  for (j=0;j<6;j++)
  {
    printf("\nDie %d. Zahl: ",j+1);
    scanf("%d",&x);
    while (x<1 || x>49 || i[x]==1) //Gültig und nicht vorhanden
    {
      fflush(stdin);
      printf("\nUngueltige Eingabe: ");
      scanf("%d",&x);
    }
    i[x]=1;

    head();
    printf("Aktuelle Eingabe:");
    print(i);
  }
}
//Ziehung
void getDrawing(int d[])
{
  int x=0,j;
  for (j=0;j<50;j++) d[j]=0;
  for (j=0;j<6;j++)
  {
    x=rand()%49+1;
    while (d[x]==1)
    {
      x=rand()%49+1;
    }
    d[x]=1;
  }
}
//Treffer zählen
int getHitCount(int i[],int d[])
{
  int j;
  int h=0;
  for (j=0;j<50;j++)
  {
      if (d[j]==1 && i[j]==1) h++;
  }
  return h;
```

```
}
//Ausgabe
void visualizeRun(int i[],int d[])
{
  int j;
  printf("\nIhre Eingabe:");
  print(i);
  printf("\nZiehung:");
  print(d);
  printf("\n");
  printf("\n         ");
  for (j=1;j<50;j++)
  {
    //Farbe für Ausgabe setzen - nur unter Windows!!!
    if (d[j]==1 && i[j]==1)
      SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
        FOREGROUND_GREEN | FOREGROUND_INTENSITY);
    else if (d[j]==1 && i[j]==0)
      SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
        FOREGROUND_RED | FOREGROUND_INTENSITY);
    else if (d[j]==0 && i[j]==1)
      SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
        FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_INTENSITY);
    else
      SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
        FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE);
    printf("%3d",j);
    if (j%7==0) printf("\n         ");
  }
  SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
    FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE);
  printf("\n");
}
int main()
{
  int input[50];
  int drawing[50];
  srand((unsigned)time(NULL));

  getInput(input);
  head();
  printf ("\nZiehung laeuft ...");
  getDrawing(drawing);
  head();
  visualizeRun(input,drawing);
  printf("\nSie haben %d von 6 Richtigen!",
                        getHitCount(input,drawing));

  fflush(stdin);
  getchar();
  return 0;
}
```