

Software-Technik

<http://paranoia.scienceontheweb.net/paranoia> – <mailto:paranoia@hush.com>

22. August 2011

Inhaltsverzeichnis

1	Analysieren von Projekten	4
1.1	Zweck	4
1.2	Kommunikation der Module	4
1.2.1	Kommunikation über Parameter	4
1.2.2	Kommunikation über globale Variablen	4
1.2.3	Kommunikation über Call by Reference	5
2	Projekte	6
2.1	MenschÄrgereDichNicht	6
2.2	Visualisierung	7
2.3	Versandhaus	8
2.4	sort	8
2.5	Bibliothek	9
2.6	Hangman	9
2.7	Memory	9
2.8	indent	10
2.9	Rechentruainer	10
2.10	Handlungsreisender	10
2.11	Game of Life	12
3	Datentypen: TYPE, RECORD, SET	13
3.1	Bitbreite und Vorzeichen	13
3.2	Typvereinbarung TYPE	13
3.3	Datensatz (record)	14
3.3.1	Definition von RECORDs	14
3.3.2	Operationen mit records	14
3.4	Menge (set)	15
3.4.1	Kurze Wiederholung der Mengenlehre	15
3.4.2	Operationen an Mengen	15
3.4.3	Syntax von Mengen in Pascal	15
3.5	Übung Datensätze und Mengen	17
3.5.1	Versandhaus	17
3.5.2	Programm Menge1	17
3.5.3	Mächtigkeit	17
3.5.4	Lotto	17
3.6	Aufzählung (enumeration)	18

4	Zeitmessung	19
4.1	Zeitdifferenzen	19
4.2	Übung	19
4.2.1	Time1	19
4.2.2	SortVergleich	19
5	Cursormanipulation - to be written	20
6	String-Manipulation - to be corrected	21
6.1	Aufbau	21
6.2	Binäre Operationen (2 Operanden erforderlich)	21
6.2.1	concat - Zwei Strings zu einem neuen zusammenfügen	21
6.2.2	Vergleichen	21
6.3	Unäre Operationen (1 Operand erforderlich)	21
6.3.1	Uppcase - wandelt in Großbuchstaben	21
6.3.2	copy - schneidet einen String aus einem anderen	22
6.3.3	pos - sucht einen String in einem anderen	22
6.3.4	val - wandelt String in Zahl	22
6.3.5	str - wandelt Zahl in String	22
6.4	Übung	22
6.4.1	Hallo Welt!	22
6.4.2	Hallelt!o W	22
6.4.3	Zufalls-Strings	22
6.4.4	sed	23
6.4.5	Zahleneingabe	23
7	Dateien	24
7.1	Datentyp <code>filehandle</code>	24
7.2	Dateioperationen und Syntax	24
7.2.1	Syntax und Struktogramm	24
7.2.2	Dateiende	24
7.3	Andere Datentypen [Optional]	25
7.4	Übung	26
7.4.1	Kleines Einmaleins	26
7.4.2	cat	26
7.4.3	bin2dez	26
7.4.4	dez2bin	26
7.4.5	wav2dez2wav	26
7.4.6	Manyfiles	26
7.4.7	bin2hex	26
8	Sortierverfahren	27
8.1	Rückblick: Bubblesort	27
8.1.1	Optimierung	27
8.2	Sortieren durch Maximum oder Minimum	27
8.3	Sortieren durch Maximum UND Minimum	27
8.4	Quicksort	28

8.5	Optimierung	28
8.6	Bozo-Sort [optional]	29
8.7	Bozo-Sort mit Badness [optional]	29
8.7.1	Nachts alleine draußen	29
8.7.2	BozoSort mit Badness	30
8.8	Übung	31
8.8.1	Papier	31
8.8.2	Sort3Verf	31
8.8.3	SortVergleich	31
8.8.4	Quicksort	31
9	Value vs. Referenz (optional)	32
10	Suchverfahren	33
10.1	Binärer Baum	33
10.1.1	Beispiel Bibliothek zu 1000 Büchern	33
10.1.2	Beispiel Bibliothek zu 1 000 000 Büchern	33
10.1.3	Bibliothek zu 500 Büchern	33

Disclaimer

Wissen ist zum Teilen da. Ich teile mein Wissen mit Ihnen, lieber Kollege.

Ich bin aber nicht perfekt. Unter paranoia@hush.com
nehme ich dankbar Ihre Verbesserungsvorschläge entgegen.

*

Legal Blurb: Alle Informationen in diesem Dokument sind falsch, unvollständig,
irreführend, irrelevant und / oder funktionieren einfach nicht.

Wenn Sie es trotzdem benutzen, und es geht dabei etwas kaputt, ist das Ihr Problem,
nicht meins.

*

Bitte teilen Sie meine Web-Adresse nicht Ihren Schülern mit.

1 Analysieren von Projekten

Def.: Analysieren ist das Herunterbrechen von großen Aufgaben in mehrere kleine, leicht lösbare Aufgaben.

1.1 Zweck

Eine größere Aufgabe benötigt meistens ein größeres Programm als Lösung.

Größere Programme werden im BG meistens ohne korrekte Vorbereitung (zB Gliederung) heruntergeschrieben und sind dann sehr schwer bis überhaupt nicht zu debuggen (entwanzen, entfehlern).

Andererseits paßt jedes größere Menüprogramm in folgendes Schema:

```
program myprog;

var ..... ;

begin
  initialisiere;
  repeat
    zeigeMenue;
    readln(wahl);
    case wahl of ...
      ...
    else
      writeln ('Wähle nur 0 bis 5.');
```

Jeder Teil ist dabei als Modul relativ leicht zu schreiben. Da er nur einmal auftaucht, muß er auch nur einmal entfehlt werden.

1.2 Kommunikation der Module

Analysieren verschiebt die Problematik von "Wie sag ichs meinem Programm?" zu "Wie sag ichs meinem Modul?" Schließlich muß ich dem Modul `zeigeUhrzeit` meine Lieblings-Hintergrundfarbe zur Verfügung stellen oder dem Modul `quicksort` bzw. `druckeRechnungen` sagen, wo es sein zu sortierendes Feld bzw. seine Datenbasis findet.

1.2.1 Kommunikation über Parameter

Was die Uhr angeht, kann man ihr stets die Parameter `MyBgColor` und `MyFgColor` zuschieben. Man darf es nie vergessen - sonst meckert der Compiler.

1.2.2 Kommunikation über globale Variablen

Bei der Uhr kann man auch die globalen Variablen `MyBgColor` und `MyFgColor` definieren und benutzen. Beim Sortieren und Datenbasen-Untersuchen ist das auch ein gangbarer Weg. Man ist dabei aber an den Variablennamen gebunden: ein globales Feld `a` heißt dann auch in jeder Prozedur und Funktion `a`.

Besonders schön machen sich globale Variablen wie `n` für die Größe von zu verarbeitenden Feldern oder `debug:boolean`, das an kritischen Stellen Sicherheitsüberprüfungen durchführt, zB

```
if debug then writeln ('Hoppla, i darf nicht 0 sein!');
```

1.2.3 Kommunikation über Call by Reference

NICHT BENUTZEN. DONT USE IT (I MEAN IT). NE PAS DE UTILISER. NON UTILIZZARE. PROHIBIDO FUERTEMENTE. TESCHEKÜR LERIM.

2 Projekte

2.1 MenschÄrgereDichNicht

MenschÄrgereDichNicht von CA, FE, BA, BE, SWT GTS, 29.02.2001

1 Neues Spiel
2 Optionen
0 Exit

Optionen

1 Spiel von c:\mdn\ laden
2 Aktuelles Spiel speichern
3 Neues Spiel
4 Spieler konfigurieren
0 Weiter

Neues Spiel

1 Aktuelles Spiel wegwerfen
2 Aktuelles Spiel speichern
0 Zurück zum Spiel

Wähle : 1

Wieviele Spieler (2-4) : 4

Wählen Sie M für Mensch oder R für Rechner.

Spieler 1 (M/R) : M Nickname: ernie
Spieler 2 (M/R) : R Nickname: rechner2
Spieler 3 (M/R) : M Nickname: bert
Spieler 4 (M/R) : r Nickname: rechner4

Enter für Weiter, andere Taste für Nochmal von vorn :

Spiel laden

Aktuelles Spiel wurde nicht gespeichert! Wegwerfen (J/N) ? : J

a 236748672.txt c 287468274.txt e 12346782.txt
b 237818934.txt d 276345832.txt f 83671723.txt

Drücken Sie einen Buchstaben (a-f) zum Laden,
andere Taste zum Fortsetzen des aktuellen Spiels.

Spieler konfigurieren

1 setze ernie auf Rechner
2 setze rechner2 auf Mensch
3 setze bert auf Rechner
4 setze rechner4 auf Mensch

Wähle : 4 Nickname : bibo

1 weiter konfigurieren
0 zurück zum Spiel

Wähle : 1

1 setze ernie auf Rechner
2 setze rechner2 auf Mensch
3 setze bert auf Rechner
4 setze bibo auf Rechner

Wähle : 3 Spieler 3 wurde auf bert_rechner gesetzt

1 weiter konfigurieren
0 zurück zum Spiel

Wähle : 0

Aktuelles Spiel speichern

2.3 Versandhaus

Ein sehr kleines, hochspezialisiertes Versandhaus hat 1 Mitarbeiter, 1 386er und 1 Telefon. Es liefert 10000 verschiedene Artikel.

artikel.txt Jeder Artikel hat eine eindeutige 5-stellige Nummer, einen Namen, einen Lieferanten, einen Lagerbestand und einen Preis. Manchmal kommen neue Artikel dazu, oder vorhandene werden aus dem Sortiment genommen. Manchmal werden alle Preise um 2-5 % erhöht.

kunden.txt Jeder Kunde hat Nummer, Name, Vorname, Adresse und Telefon. Manche haben Email und Handy. Manchmal kommen neue Kunden dazu oder verschwinden.

auftrag.txt Wenn ein Kunde anruft und einen Auftrag erteilt, wird dieser mit Datum und allen relevanten Daten an die Datei `auftrag.txt` angehängt. Außerdem wird vollautomatisch eine Rechnung gedruckt (auf den Bildschirm). Manchmal bestellt ein Kunde 10 Stück eines Artikels.

Manchmal will der Chef wissen, welches die 10 Kunden mit dem größten Umsatz sind - nach Umsatz sortiert.

Schreiben Sie ein Programm, mit dem eine ungelernete Kraft auf Anhieb arbeiten kann - menügesteuert, unter starker Verwendung des Nummernblocks.

Schreiben Sie auch ein kleines Programm, das ein Testsystem mit 10 Kunden, 3000 Artikeln (Namen wie "sdhjkdsjkh" sind zulässig) und 99 Aufträgen generiert.

Gestalten Sie die Dateien so:

```
Kunde.Nr :
4711
Kunde:
Peter Müller
Straße:
Seestr. 13
Ort:
E-13476 San Iglu de Inacio
usw...
```

2.4 sort

Das Programm bietet

- eine Vorschau mit Zeilennummern, so daß man durch die zu sortierende Datei scrollen kann. Tabs werden dabei durch TAB (Rot hinterlegt) dargestellt.
- die Wahl eines oder mehrerer Spaltenseparatoren (Leerzeichen, Semikolon, Tab, Komma),
- die Wahl eines Ziffernformats (mit Dezimalpunkt oder Dezimalkomma),
- die Wahl, wieviele Kopfzeilen zu ignorieren sind,
- die Wahl, nach welcher Spalte sortiert werden soll.

Das Programm sortiert Dateien mit maximal 30000 Zeilen. Die Namen von Ein- und Ausgabedatei sind frei wählbar.

2.5 Bibliothek

In einer Bibliothek gibt es 100 Bücher mit Titel und Nummer (Datei `buecher.txt`) und 20 Nutzer mit Name und Nummer `nutzer.txt`. In der Datei `ausleih.txt` wird festgehalten, wer wann was ausleiht. Wenn er es zurückbringt, wird der Eintrag gelöscht.

Folgende Sachen können passieren:

neuer Nutzer	Nutzer editieren	Nutzer löschen
neues Buch	Daten Buch ändern	Buch entsorgen
Nutzer entleiht Buch	Nutzer bringt Buch zurück	Nutzer meldet Buch als verloren

Das Programm soll außerdem alle Bücher, deren Leihfrist überzogen ist, mit Entleiher und Überziehzeitraum ausdrucken können (auf den Bildschirm).

Der Bibliothekar sitzt am Eingang neben dem 386er und will mit dem Rechner arbeiten, ohne jemals Dokumentation lesen zu müssen - er liest lieber Romane des 19 Jahrhunderts.

Gestalten Sie die Dateien so:

```
Nr:
2378456
Titel:
djkhflasjkdhflkj asjkdfhlakjsdhflka hsd
Autor:
sn as vajsd voiasnjv
```

2.6 Hangman

Im Spiel Hangman muß man Worte von 4-14 Buchstaben Länge raten. Für das gesamte Wort hat man 60 sec Zeit. Wer 7mal den falschen Buchstaben drückt, scheidet aus.

1-4 Spieler. Schreiben Sie eine deutsche und eine englische Wörterliste (je 200 Wörter).

Am unteren Bildschirmrand läuft ein Zeitbalken in % oder Sekunden.

Während des Laufens können die Spieler die Optionen speichern und sich die Bestenliste (10 Einträge) ansehen. Dabei bleibt der Zeitbalken stehen.

Verschlüsseln Sie die Namen in der Bestenliste, und machen Sie es so, daß das Programm Manipulationen an der Bestenliste erkennt, das ausgibt und sie dann löscht. Beispiel: `anton 1000 8268`. 8268 ist eine Prüfziffer, die durch ein geheimes Verfahren aus der 1000 berechnet wird.

2.7 Memory

1-4 Spieler, 1-3 Runden, Zeitlimit einstellbar von 5-15 sec. Teilen Sie den Bildschirm in 10x10 Felder. Jeder Spieler hat reihum 2 Züge, die er über den Ziffernblock eingibt. Richtig geratene Pärchen werden entfernt, und der Spieler bekommt 1 Punkt. Auf den Kärtchen stehen Wörter in bunten, kontrastreichen Farben.

Am unteren Bildschirmrand läuft ein Zeitbalken. Wenn das Zeitlimit verstrichen ist, verfällt der Zug, und der nächste Spieler ist dran.

Die Bestenliste wird am Ende des Spiels angezeigt und auf der Platte in `c:\hidden\bestmem.txt` gespeichert.

2.8 indent

Schreiben Sie ein Pascal-Programm, das ein Pascal-Programm einliest und schön einrückt. Beispiel:

```

program demo;

var
  i, j, k : integer;           (* alle Kommentare *)
      f : array [1..20] of byte; (* in Spalte 50-80 *)

begin
  for i := 1 to 17 do
    writeln (i:3);
    for i := 1 to 17 do begin
      writeln (i*i:8);
    end;
  end;
end.

```

Gehen Sie modular vor. 0 Bilden Sie ein Programm `badform.pas`. 1 Lassen Sie es lesen und auf den Bildschirm schreiben. 2 = 1 + Doppelte Leerzeichen und Zeilenwechsel rauswerfen. 3 = 2 + setzen und beachten von VAR BinZwischenHochkommas : BOOLEAN. ... usw. ... n = alles vorige + user wählt datei + schreiben in datei.

2.9 Rechentruainer

`trainer.exe` fragt nach dem Usernamen, dann erscheint das Menü:

```

1 Rechenuebung starten
2 Optionen
3 Bestenliste
4 Credits
0 Beenden

```

Optionen-Menü:

```

Optionen
1 Setzen der Bearbeitungszeit pro Aufgabe
2 Fortschrittsbalken an (Status: AUS)
3 Uhr in Bildschirmecke AUS (derzeitig: AN)
4 Anzahl der Aufgaben pro Runde
9 Speichern und Schliessen
0 Schliessen ohne Speichern

```

Die Ergebnisse kann man mit den Ziffern-Tasten (über QWERTZUIOP) oder den Keypad-Tasten eingeben. ENTER bestätigt, BACKSPACE geht eine Ziffer zurück.

Mit der neuen Aufgabe erscheint rechts oben "RICHTIG" oder "FALSCH, richtig: XXXXX".

Nach einer Runde erscheint eine Statistik:

```

  Sie haben 100 Aufgaben bearbeitet.
  Richtig:   75          75 \%  Falsch:   25          25 \%
press eniki

```

2.10 Handlungsreisender

Gegeben sind n Punkte in der x - y -Ebene. Sie sind durchnummeriert von 1 bis n . Gesucht ist der kürzeste Weg von Punkt1 über alle Punkte und wieder zurück zu Punkt1. $0 < n < 99$.

Menübaum:

- Einlesen der Punkte
 - + Eintippen von Hand, beenden mit q
 - + fuer $i = 1$ to n : $x_i = \cos(i)$; $y_i = \sin(i)$
 - + Punkte liegen regelmaessig in Matrix $p \times q$ (zB 4x4) mit Abstand 1;
 - + Punkte liegen zufaellig verstreut von -10 bis 10
 - + unbekannte Anzahl Punkte wird von a:punkte.txt eingelesen
 - zusaetzliche meldungen, zB "starte lesen...", "13 punkte eingelesen."
 - Wahl des Algorithmus
 - + ein analytischer Algorithmus, der garantiert den k"urzesten Weg liefert, aber in quadratischer Zeit;
 - + ein bozo-fizierter algorithmus wie folgt:
 - suche irgendeine punktreihenfolge;
 - wiederhole
 - berechne den weg, der dazugehoert;
 - wenn er kuerzer ist als jeder bisher berechnete weg, merk ihn dir;
 - vertausche n-mal 2 punkte;
 - bis keine verkuerzung mehr moeglich oder zeit abgelaufen
 - + 5 bozo-fizierte Durchlaeufer hintereinander, mit darstellung der 5 ergebnisse
 - einstellen der optionen
 - + abbruch nach XX sekunden. wenn $XX=0$, dann laufe fuer immer.
 - + fortschrittsanzeige.
 - fuer jeden berechneten weg zeige die laenge oder
 - fuer jeden berechneten weg gib einen "." aus oder
 - fuer jeden berechneten kuerzesten weg gib ein "+", sonst ein "-"
 - wahlbar: uhr in rechter oberer bildschirmcke
 - wahlbar: stoppuhr in linker oberer bildschirmcke
- exit

2.11 Game of Life

Gegeben ist ein Gitter, max. 80 Spalten breit und 22 Zeilen hoch. Darauf sind zufällig ein paar Bakterien (Keime, germs) verteilt. Keime sind entweder tot oder lebendig. Jeder Keim hat 8 Nachbarfelder.

Es gelten folgende Regeln:

1. Ein Keim mit 0 oder 1 Nachbarn stirbt an Einsamkeit.
2. Ein Keim mit 4 bis 8 Nachbarn stirbt an Hunger.
3. Ein Keim mit 2 bis 3 Nachbarn überlebt.
4. Ein leeres Feld mit genau 3 Nachbarn wird ein Keim.

Beim Starten des Programms läuft eine Demo: 1 Titelzeile, 20 Spalten x 20 Zeilen Keime, zufällig belegt, Delay=0,5 sec. Die Keime werden in 2 Spalten Breite durch "[]" angezeigt. Nach 500 Iterationsschritten wird das Feld neu belegt.

Durch Drücken der Leertaste wird die Iteration angehalten bzw. fortgesetzt.

Mit ESC kommt man ins Menu:

Menu

- 1 Neue Spaltenanzahl eingeben
 - 2 Neue Zeilenanzahl eingeben
 - 3 Neues Delay
 - 4 Neu Anzahl der Iterationsschritte
 - 5 Erneut Starten
 - 6 Zähle bei Rand-Keimen den gegenüberliegenden Rand mit (zZt: OFF)
- 0 Beenden

Das Menü ist so programmiert, daß man nach der Wahl eines Menüpunktes nicht Enter drücken muß.

3 Datentypen: TYPE, RECORD, SET

3.1 Bitbreite und Vorzeichen

Wir kennen bereits die Datentypen `integer`, `real`, `char`, `string`, und `boolean`.

Außerdem gibt es noch `double`, `byte`, `word`, `smallint`.

`double` ist eine Zahl mit Nachkommastellen. Es sind ca 13 Stellen gültig (bei `real` nur 8, und der Exponent geht bis ca 204 (bei `real` nur bis ca 59)).

`byte` ist eine vorzeichenlose 8-Bit-Zahl, d.h. zwischen 0 und 255 (inclusive).

`word` ist eine vorzeichenlose 16-Bit-Zahl, d.h. zwischen 0 und 65535 (inclusive).

`integer` ist eine vorzeichenbehaftete 16-Bit-Zahl, d.h. zwischen -32768 und 32767 (inclusive).

Achtung Unter Unix/Linux sind `integer` in der Regel 32-Bit-Zahlen.

3.2 Typvereinbarung TYPE

Statt

```
var
  b   : byte;
  f   : array [0..999] of integer;
```

hätte ich auch schreiben können:

```
type
  tMySpecialByte = byte;
  tMyFeld        = array [0..999] of integer;
var
  b   : tMySpecialByte;
  f   : tMyFeld;
```

`type` definiert und benennt einen Variablentyp, den man dann wie jeden anderen Variablentyp auch benutzen darf.

Ein vorangestelltes `t` und die Großschreibung der Wortanfänge ist nicht vorgeschrieben, erhöht aber die Lesbarkeit.

3.3 Datensatz (record)

Ein Datensatz ist ein Satz (üblicherweise eine Zeile) aus Daten beliebigen Typs. Die Typen und deren Reihenfolge ist konstant.

Beispiel Zum Verwalten von Freunden bietet sich folgender Datensatz an:

```
vorname:nachname:adresse:plz:ort:festnetz:handy:email:homepage
```

Zum Verwalten von Versuchsdaten an einem Ottomotor, der mit 20 Temperatursensoren Tag und Nacht in einem Versuchslabor läuft, bietet sich an:

```
timestamp:lufttemp:luftfeucht:luftdruck:sens01:sens02:sens03... usw.
```

Und für Leute, die nicht so viel schreiben wollen:

```
datum : jahr:monat:tag (Beachte: Most Significant First.)
```

3.3.1 Definition von RECORDs

```
VAR
tMyFriend : RECORD
  vorname,nachname,adresse : STRING;  (*als ob sie einzeln dastuenden*)
  plz : LONGINT;
  ort, festnetz, handy, email, homepage : STRING;
END;
```

Übung Und jetzt dasselbe nochmal für den den Versuchsmotor.

Und anschließend nochmal für den Versuchsmotor, ABER: zunächst einen Typ `mein_motor_record` definieren, und dann erst eine Variable.

3.3.2 Operationen mit records

Man kann nicht auf ganze records zugreifen, sondern bloß auf die Komponenten. Zuweisungen und Vergleiche funktionieren wie gehabt.

ARRAYs und FILEs OF records probiert bitte selber aus.

```
TYPE
  tDatum : RECORD
    jahr, monat, tag : INTEGER;
  END;
```

```
VAR
  mydatum : tDatum;
```

```
begin
  mydatum.jahr := 1995;
  mydatum.monat := 12;
  mydatum.tag := 31;
  ...
```

3.4 Menge (set)

3.4.1 Kurze Wiederholung der Mengenlehre

**Eine Menge ist ein Satz von Elementen.
 Alle Elemente sind ungleich.
 Die Reihenfolge der Elemente ist beliebig, d.h. unsortiert.
 Die Anzahl der Elemente reicht von 0 bis ∞ .**

Wir nehmen nur positive ganze Zahlen und 0 als Elemente.

Beispiele Menge der natürlichen Zahlen : $N = \{1, 2, 3, \dots\}$

Leere Menge: $\emptyset = \{\}$

Menge der erlaubten Eingaben : $Erlaubte_Eingaben = \{ 'j', 'J', 'n', 'N' \}$

3.4.2 Operationen an Mengen

Schnittmenge ist die Menge derjenigen Elemente, die Menge A und Menge B gemeinsam haben - also die in beiden Mengen vorhanden sind.

Falls Menge A und Menge B keine Elemente gemeinsam haben, ist es die leere Menge, \emptyset .

Vereinigungsmenge sind alle Elemente aus Menge A und Menge B zusammengenommen. Doppelte Elemente werden dabei automatisch gelöscht.

Differenzmenge Menge A - Menge B bedeutet: nimm Menge A, und wirf alle Elemente von Menge B heraus.

A - B ist nicht dasselbe wie B - A. Differenzmengenbildung ist nicht kommutativ.

3.4.3 Syntax von Mengen in Pascal

Mengen in Pascal dürfen nur Elemente vom Typ `byte` oder `char` enthalten. Die Elemente sind auf 8 Bit begrenzt.

Definition

VAR

m1 : SET OF BYTE;

m2 : SET OF CHAR;

m3 : SET OF 55..99;

Initialisierung

Mengen in Pascal darf man Elemente, Listen, Aufzählungen oder Kombinationen daraus zuweisen.

```

m1 := [];                (* die leere Menge *)
m2 := ['a', 'b'..'y', 'z'];
m3 := [55, 66, 99];     (* neugierig was passiert wenn man die *)
                        (* Grenzen ueberschreitet? Ausprobieren. *)

```

Ein Element dazu

Der Operator + ist überladen:

- Zahlen addiert er.
- Strings hängt er hintereinander.
- Mengen vereinigt er.

```

m1 := m1 + [222, 333];   (* das geht nicht - wieso? *)

```

Mengen verknüpfen

Vereinigungsmenge	+
Differenzmenge	-
Schnittmenge	*

Existenz eines Elements testen Wenn a in Menge B ist...

```

IF a IN m1...           (* IN ist ein reserviertes Wort *)

```

Mengen ausgeben Es gibt keine direkte Möglichkeit, wir behelfen uns bei Mengen aus byte:

```

for i := 0 to 255 do
  if i in Mangel
    then writeln (i)
;

```

Elemente zählen Es gibt keine direkte Möglichkeit, wir behelfen uns bei Mengen aus byte:

```

z := 0;
for i := 0 to 255 do
  if i in Mangel
    then z := z + 1
;

```

3.5 Übung Datensätze und Mengen

3.5.1 Versandhaus

Legen Sie 10 Kunden an mit `nr`, `anrede`, `vorname`, `nachname`, und Zufallswerten. Bsp.: Kunde 013, Frau, Hlabfsdfgb, Hwfnbsda.

Legen Sie 10 Artikel an mit `Nummer`, `Preis`, `Beschreibung`.

Legen Sie eine 10x10 Tabelle OF INTEGER an. In die Felder schreiben Sie, wer wieviel von was gekauft hat.

Geben Sie anschließend für alle Kunden die Rechnung aus:

* Kunde 03 hat nichts gekauft.
Weiter mit CR...

Sehr geehrter Herr Gsdjfjknvnpn,

Sie kauften bei uns:

5 Stueck	wedknvklasj fgkjasdklf				
	Einzelpreis:	EUR	80.45	EUR	401.80
2 Stueck	dfui vnqeijvn pasj vn				
	Einzelpreis:	EUR	5.00	EUR	20.00
...					
=====					
GESAMT				EUR	499.97
+ 17 Prozent Mehrwertsteuer				EUR	84.99
TOTAL				EUR	584.96

Weiter mit CR...

3.5.2 Programm Menge1

- Menge a kann bytes aufnehmen; Menge b kann die Zahlen von 0 bis 28 aufnehmen;
- Menge c kann das Ergebnis der Verknüpfung von a und b aufnehmen.
- Stecken Sie in Menge a die Zahlen von 240 bis 270, und geben Sie sie aus.
- Stecken Sie in Menge b die Zahlen von 4 bis 35, und geben Sie sie aus.
- Bilden Sie in c nacheinander die Vereinigungsmenge, Schnittmenge und Mengen-Differenz(en), und geben Sie sie aus.

3.5.3 Mächtigkeit

Die Mächtigkeit einer Menge ist die Anzahl der Elemente.

Schreiben Sie die Funktion `maechtigkeit` für ein `set of byte`, und benutzen Sie sie im Programm `Menge1`.

3.5.4 Lotto

Schreiben Sie ein Programm, das zufallsgesteuert 1000mal Lottozahlen 6 aus 49 + Zusatzzahl + Superzahl zieht.

a) Benutzen Sie dabei den Datentyp `Menge`.

b) Machen Sie's nochmal, aber benutzen Sie diesmal ein `array of boolean` von 1 bis 49..

3.6 Aufzählung (enumeration)

Warnung: Verschiedene Compiler behandeln verschiedene Aufzählungen verschieden.

```
program enum2;

var g : 30..240;

begin
  for g := 20 to 290 do write (g:4);
  writeln;
end.
```

Compiling enum2.pas

enum2.pas(6,12) Warning: range check error while evaluating constants

enum2.pas(6,18) Warning: range check error while evaluating constants

Assembling enum2

Linking enum2

7 Lines compiled, 0.1 sec

***** fpc start output *****

20 21 22 23 24 25 26 27 28 29 30 31 32 33 34

***** fpc end output *****

Statt das zu erklären, drücke ich mich lieber komplett vor Aufzählungen.

4 Zeitmessung

```

program zeitmes;
uses dos;

const days : array [0..6] of string
      = ('So', 'Mo', 'Di', 'Mi', 'Do', 'Fr', 'Sa');

var
  h,mi,s,hund : word;
  y,mo,d,dow  : word;
  c : char;

begin
  c := ':';
  gettime (h,mi,s,hund);           //
  getdate (y,mo,d,dow);           //
  writeln (y,c,mo,c,d,c,h,c,mi,c,s,c,hund);
  writeln ('day of week = ', days[dow]);
end.
+++++++ running ++++++
2004:4:28:21:3:47:33
day of week = Mi
+++++++ finishd ++++++

```

4.1 Zeitdifferenzen

Die Bestimmung von Zeitdifferenzen ist unglaublich mühsam.

Zweckmäßigerweise basteln Sie sich die Variable `daytime`, die die Anzahl der seit Mitternacht verflassenen Sekunden aufnimmt. Oder Sie basteln sich einen universellen Zeitstempel (`timestamp`), der die Tage seit Christi Geburt zählt (00:00:00 am 01.01. des Jahres 1). Tageszeiten können Sie hierbei durch Nachkommastellen nachmachen. Excel macht das auch so, allerdings zählt es ab 1901 bzw. 1904 (PC- bzw. Mac-Version).

Oder Sie zählen in Julianischen Tagen (Tag 0 ungefähr 5000 v.u.Z.).

Oder Sie benutzen die übliche Unix-Zeit: die Sekunden, die seit dem Beginn der Unix-Epoche verflassen sind (ohne Schaltsekunden).

Die Unix-Epoche beginnt am 01.01.1970, 00:00:00.

4.2 Übung

4.2.1 Time1

Schreiben Sie ein Programm, das die Zeit zwischen zwei Tastendrücken mißt und ausgibt.

4.2.2 SortVergleich

Schreiben Sie ein Programm, das nach der Anzahl der zu sortierenden Werte fragt ($9 < n < 9999$). Dann erzeugt es immer ein neues `array of byte`, sortiert dieses 5mal und merkt sich die Zeit, die es dafür braucht.

Am Schluß werden die benötigten Zeiten und Verfahren übersichtlich ausgegeben:

```

sortiere 5mal 999 werte...
bubblesort      12.34 sec
maxisort         6.54 sec
minisort         6.45 sec

```

5 **Cursormanipulation - to be written**

6 String-Manipulation - to be corrected

6.1 Aufbau

```
var s,t : string;
```

In Turbo-Pascal ist ein string ein array [0..255] of char. Das erste bis 255. Feld sind die `ascii`-Codes der Buchstaben, im 0. Feld steht die Länge des Strings.
`length(s)` gibt die Länge des Strings zurück.

6.2 Binäre Operationen (2 Operanden erforderlich)

6.2.1 concat - Zwei Strings zu einem neuen zusammenfügen

```
s := s + t; oder s := concat (s,t,s); (*so viele wie man will*)
```

6.2.2 Vergleichen

```
if (s<t) then writeln (s, ' ist kleiner als ', t);
```

Es sind die üblichen Operatoren für kleiner, größer, gleich, ungleich zulässig.

Kleinbuchstaben haben `ascii`-Werte von 97 bis 122. Großbuchstaben haben `ascii`-Werte von 65 bis 90. Daher sind Strings, die mit Großbuchstaben anfangen, kleiner als Strings, die mit Kleinbuchstaben anfangen.

6.3 Unäre Operationen (1 Operand erforderlich)

```
var s,t : string;
    i : integer;

s := 'abcdefgh';
```

6.3.1 Uppcase - wandelt in Großbuchstaben

Uppcase ist zwar keine string-Funktion, aber dafür der Workaround für das case-insensitive Vergleichen von Strings.

```
function upcase (inchar) : outchar; (*gibt Großbuchstabe von inchar zurück*)
```

Lies: Die Funktion `upcase` nimmt eine Variable vom Typ `char` als Parameter und gibt den passenden Großbuchstaben zurück.

`upcase` funktioniert nur für [a..z], also nicht für deutsche Sonderzeichen.
 ä 132 ö 148 ü 129 Ä 142 Ö 153 Ü 154 ß 225

6.3.2 copy - schneidet einen String aus einem anderen

```
function copy (string, startpos, wieviele) : string;
t := copy (s,3,4); (* t = 'cdef' *)
```

Wenn `s` zu kurz ist, ist auch das Ergebnis kurz.

6.3.3 pos - sucht einen String in einem anderen

```
function pos (substring, string) : integer position;
i := pos ('fgh', s); (* i = 6 *)
```

Falls der Substring nicht im String vorhanden ist, gibt die Funktion 0 zurück.

6.3.4 val - wandelt String in Zahl

```
procedure val (string, var num-var, var error);
num-var ist von einem beliebigen Zahlentyp.
```

Falls die Konvertierung gelingt, ist `error` 0, sonst zeigt es auf die fehlerhafte Position in `string`.

Der Aufruf bedeutet folgendes:

- `string` wird als Eingabe-String verwendet, d.h. nicht geändert.
- `num-var` und `error` werden als zu ändernde Variablen übergeben, d.h. sie werden höchstwahrscheinlich geändert.

6.3.5 str - wandelt Zahl in String

```
procedure str (num-var:Format, var string);
```

Die Variable `num-var` wird inclusive (optionalem) Format übergeben, `string` wird geändert.

```
i := 14;
str(i:13,s); (* s = '          14' *)
```

6.4 Übung

6.4.1 Hallo Welt!

Erstellen Sie drei Substrings von 'Hallo Welt!', und geben Sie sie aus.

6.4.2 Halle!o W

Erstellen Sie drei Substrings von 'Hallo Welt!', sortieren Sie sie aufsteigend, und geben Sie sie aus.

6.4.3 Zufalls-Strings

Erzeugen Sie 20 Zufalls-Strings der Länge 60. Geben Sie in der letzten Zeile folgenden Hinweis:

Key a steigend | d fallend | Shift: case-sensitive | q Quit
und implementieren Sie ihn.

6.4.4 sed

Schreiben Sie ein Programm, das den User nach Dateinamen (default: mytext.txt), oldstring (default: old) und newstring (default: new) fragt, die Datei einliest und alle oldstrings durch newstrings ersetzt.

6.4.5 Zahleneingabe

Schreiben Sie die Funktion `inputreal`, die vom User solange eine `real`-Zahl anfordert, bis sie eine bekommt, und diese dann zurückgibt.

Der Anforderungs-String soll als Parameter übergeben werden, zB: 'Geben Sie den Kreisdurchmesser in mm ein:'

7 Dateien

7.1 Datentyp `filehandle`

Wir wollen Daten in Dateien lesen und schreiben.

Rückblick Bisher haben wir unsere Eingabedaten von der Tastatur gelesen und auf den Bildschirm geschrieben. Dazu war keine Mehrarbeit nötig, denn das ist so voreingestellt.

Standard-Ausgabe ist der Bildschirm. Standard-Eingabe ist die Tastatur.

Ausblick Sobald wir in Dateien schreiben oder lesen wollen, müssen wir den Rechner informieren, daß die Daten nunmehr **nicht** über die Standard-Kanäle laufen sollen, sondern über andere Kanäle.

Daten-Kanäle bzw. Wegweiser für Dateien heißen `filehandle`.

Ein- und Ausgabebefehle haben dann ein File-Handle als ersten Parameter.

7.2 Dateioperationen und Syntax

7.2.1 Syntax und Struktogramm

```

program dateiops;                                (*Dateioperationen *)

var zeile : string;
    infile, outfile : text;                      (*Dateien mit ascii-Text drin*)

begin
  assign (outfile, 'myfile.txt');                (*myfile.txt zum schreiben öffnen*)
  rewrite (outfile);
  zeile := 'Irgendein Text';
  writeln (outfile, zeile);                      (*etwas hineinschreiben*)
  close (outfile);                              (*und schließen*)

  assign (infile, 'myfile.txt');                 (*myfile.txt zum lesen öffnen*)
  reset (infile);
  readln (infile, zeile);                       (*aus der datei lesen*)
  writeln (zeile);                              (*auf bildschirm schreiben*)
  close (infile);                              (*und schließen*)
  readln;                                      (* user verwirren ;>) *)
end.
```

7.2.2 Dateiende

Leider weiß man bei den meisten Dateien nicht, wie lang sie sind. TurboPascal kann aber mit der Funktion `eof(filehandle)` (EndOfFile) feststellen, ob eine Datei zu Ende ist. Der Ergebnistyp ist `boolean` und kann daher vorteilhaft in einer `while-do`-Schleife verwendet werden.

```

[...]
```

```

while (not eof(infile)) do begin
  readln (zeile);
```

```
    [... do something ...]
end;
[...]
```

7.3 Andere Datentypen [Optional]

Der Datentyp `text` ist eigentlich nur eine Abkürzung für `file of char` – mit einer Ausnahme:

Eine Text-Datei ist am Zeichen EOT (ASCII 26) zu Ende – eine Datei `file of char` nicht.

Dennoch darf man statt der Klartext-Dateien auch folgendes schreiben:

```
fhr : file of real;
fhh : file of boolean;
```

Dabei muß man die Werte mit `write` (nicht `writeln`) schreiben und mit `read` lesen. Dateien mit Zahlen darin eignen sich nicht zum Speichern von Leerzeichen und CR-LFs.

Eine Datei mit 1000 `real`-Zahlen darin umfaßt dann 6000 Byte, denn 1 `real` umfaßt 6 Byte. Man kann sie sogar in einem Hexadezimal-Editor betrachten - nur das Rückwärtsrechnen wird schwierig.

1 `boolean`, 1 `byte` oder 1 `char` wird als 1 Byte gespeichert.

Selbstdefinierte Typen wie `records` und `arrays` sind als Dateitypen nicht erlaubt.

Nicht alle Dateien, die man auf diese Weise einliest, werden beim Wiedereinlesen auf Plausibilität geprüft.

7.4 Übung

7.4.1 Kleines Einmaleins

Geben Sie das Kleine Einmaleins in die Datei `einmalei.txt` aus.

7.4.2 cat

Geben Sie die Datei `einmalei.txt` auf den Bildschirm aus.

7.4.3 bin2dez

Schreiben Sie ein Programm, das die Datei `turbo.exe` einliest und den ASCII-Code jedes Zeichens, formatiert auf 4 Stellen, in die Datei `turbo.txt` schreibt.

`turbo.exe` ist eine Binär-Datei, d.h. eine `file of char`.

7.4.4 dez2bin

Kopieren Sie `turbo.txt` nach `turbo2.txt`.

Schreiben Sie ein Programm, das `turbo2.txt` nach `turbo2.exe` re-konvertiert, und führen Sie es aus.

7.4.5 wav2dez2wav

Suchen Sie sich eine `*.wav`-Datei aus, und speichern Sie sie als 44kHz, mono, 8-Bit. Wandeln Sie sie mittels Pascal nach `*.txt`.

Erzeugen Sie einen Block aus 44100mal 127 oder 44100 Zufallszahlen zwischen 0 und 255. Fügen Sie ihn irgendwo in der Mitte in die `*.txt`-Datei ein. Wandeln Sie sie zurück zu `*.wav`, und spielen Sie sie ab.

7.4.6 Manyfiles

Erzeugen Sie im aktuellen Verzeichnis 500 oder 5000 kleine Textdateien.

Hinweis: `str(string1, integer1)` erzeugt einen String aus einem Integer.

7.4.7 bin2hex

Suchen Sie sich eine Java-Klassen-Datei, und wandeln Sie sie Byte für Byte in hexadezimalen Format.

Beispiel (mit Word¹-Format):

```
This is hexdump of krz.class.
00000000 feca beba 0300 2d00 3601 0001 6b03 7a72
00000010 0007 0101 0e00 616a 6176 612f 7477 462f
...
```

¹Ein Byte hat 8 Bit. Ein Word hat 16 Bit.

8 Sortierverfahren

8.1 Rückblick: Bubblesort

Wir haben bereits **Bubblesort** kennengelernt:

- laufe von links nach rechts durch das Feld;
- wenn das aktuelle Feld größer ist als sein rechter Nachbar, tausche die beiden;
- solange bis keine Vertauschung mehr stattfand.

8.1.1 Optimierung

Folgende Optimierungen sind dabei denkbar:

- In jeder Schleife läßt man `sorted:boolean` mitlaufen. Zu Anfang wird sie `TRUE` gesetzt. Falls man etwas vertauschen muß, wird sie `FALSE` gesetzt. Falls sie am Ende der Schleife noch immer `TRUE` ist, ist die Sortierung fertig.
- Nach dem ersten Durchlauf ist die größte Zahl mit Sicherheit ans Ende des Bereiches sortiert. Nach dem zweiten Durchlauf sind die beiden größten Zahlen mit Sicherheit am Ende des Bereiches, usw. Man lasse daher `j` als Index der inneren Schleife nicht jedesmal bis $n-1$ laufen, sondern bis $n-i$. Theoretische Überlegungen ergeben, daß sich die Sortierzeit damit halbieren müßte.

vorher:	nachher:
n-mal	n-mal
n xxxxxxxxxxxx	n xxxxxx
- xxxxxxxxxxxx	- xxxx
m xxxxxxxxxxxx	m xxx
a xxxxxxxxxxxx	a xx
l xxxxxxxxxxxx	l x

8.2 Sortieren durch Maximum oder Minimum

Kein normaler Mensch sortiert so. Er wird eher wie folgt verfahren:

- für $i = 1$. bis vorletztes Feld:
- suche das Minimum rechts vom i . Feld;
- tausche es mit dem i . Feld.

Wir wollen dies "Sortieren durch Minimum" oder kurz "MinSort" nennen. Ein analoges Verfahren funktioniert mit dem Maximum.

Da die Zahl der Vertauschungen bei beiden Verfahren stark zurückgeht, müßten sie schneller laufen als Bubblesort.

8.3 Sortieren durch Maximum UND Minimum

Bei MiniSort und MaxiSort muß man jedesmal einmal durch das Feld laufen und das Maximum oder Minimum finden. Man könnte das abkürzen, indem man bei einem Lauf sowohl Maximum als auch Minimum findet und es mit den passenden Werten am Anfang und am Ende des Feldes tauscht. Dieses wollen wir "Sortieren durch Maximum **und** Minimum" oder kurz "MiniMaxSort" nennen.

Es gibt allerdings ein Riesenproblem dabei: Wenn im zu sortierenden Feldabschnitt das erste Feld das Maximum ist und das letzte Feld das Minimum ist, werden beide zweimal getauscht. Dabei wird das Resultat falsch.

Lösung: Falls man dies vorher abprüft und entsprechend handelt, wird die Lösung wieder richtig. Da die Anzahl der Durchläufe halbiert wird, müßte MiniMaxiSort schneller laufen als alle Vorgänger.

8.4 Quicksort

Quicksort ist superschnell. Es funktioniert wie folgt:

- Du bekommst ein Feld mit den Indizes von l(inks) bis r(echts).
- Such dir ein Pivot-Element p , zB $a[l]$.
- Sortiere alle Elemente $< p$ nach links.
- Sortiere alle Elemente $> p$ nach rechts.
- Setze p auf den verbleibenden Platz.
- Nun sortiere die Kleinen auf dieselbe Weise.
- Nun sortiere die Großen auf dieselbe Weise.

8.5 Optimierung

- Suche von links ein zu großes Element.
- Suche von rechts ein zu kleines Element.
- Tausche die beiden.
- Bis alle Kleinen links sind und alle Großen rechts.

8.6 Bozo-Sort [optional]

Bozosort sortiert ohne Sortieralgorithmus, wie folgt:

1. Sind die Werte sortiert?
2. Wenn ja: ausgeben, fertig.
3. Wenn nein: Misch sie, und zurück zu 1)

Als Ausgangsfeld wollen wir $n, n-1, n-2, \dots, 1$ benutzen.

Als Mischen wollen wir das Vertauschen von irgend zwei Werten betrachten.

Bozo prüft (fast) alle Permutationen der Zahlenfolge ab. Eine Folge aus n Zahlen hat $n!$ Permutationen, und nur eine ist richtig. Die Sortierzeit geht also mit $n!$.

Übung Bozo sortiert 6 Zahlen im Mittel in 1 sec. Wie lange braucht er für 12 Zahlen?

Lösung $1 \text{ sec} / 6! * 12! = 665280 \text{ sec} = 184.8 \text{ h} = 7.7 \text{ Tage}$

BozoSort ist prima geeignet, einen Rechner tagelang unter höchster Rechenlast zu halten.

8.7 Bozo-Sort mit Badness [optional]

Sortieren ohne Sortieralgorithmus ist nicht so sinnlos, wie es scheint. Man kann es prima verwenden für Aufgaben, die keinen bekannten Lösungsweg haben.

8.7.1 Nachts alleine draußen

Sie befinden sich in stockfinsterer Nacht in einem hügeligen Gelände (ohne Fallen, Raubtiere, Flüsse etc.). Sie sollen den höchsten Hügel finden. Sie haben ein Gerät, das auf Knopfdruck Ihre x-y-Koordinaten und die Höhe über NN (= Koordinate z) ansagt.

Bozo Sie laufen ziellos in der Gegend herum, und speichern gelegentlich einen Punkt. Am Schluß suchen Sie den Punkt mit der höchsten Höhe.

BozoBad

- Sie laufen in einer Richtung los (= variieren die x-Koordinate).
- Nach 5 Schritten gucken Sie auf die Höhe (= Badness).
- Falls Sie sich verbessert haben, merken Sie sich die neue Badness und variieren erneut.
- Sonst gehen Sie zurück und variieren einen anderen Parameter (zB die y-Koordinate).

Letzteres Verfahren läuft besser. Wir haben nach wie vor keine Ahnung, wo der höchste Hügel ist - aber wir haben immerhin einen Bewertungsmaßstab für verschiedene, zufällig generierte Lösungen.

Bei dem Beispiel mit der hügeligen Landschaft ist die Höhe der Maßstab: je höher, desto besser. Höhe ist gut.

Manchmal kann man nur angeben, wie sehr einem das Ergebnis MISSfällt. Das ist die Schlechtheit oder "Badness". Wir werden also Lösungen mit möglichst geringer Badness suchen.

8.7.2 BozoSort mit Badness

Algorithmus:

1. Berechne `badness` der gegenwärtigen Sortierung.
2. Vertausche irgend zwei Werte.
3. Berechne `badness_neu`.
4. Falls `badness_neu` besser als `badness`, nimm dies als neuen Startpunkt.
5. Sonst mache Tausch rückgängig und versuche irgendwas anderes.
6. Brich irgendwann ab.

badness - falsch `Badness=0`.

Laufe die Zahlenreihe entlang. Wenn `AktuelleZahl > NächsteZahl`, erhöhe `Badness` um 1.

Diese `Badness` führt zu sortierten Untergruppen, aus denen der Algorithmus keinen Ausweg findet. Bsp: aus 987654321 wird 1472569378 mit `Badness=2`.

badness-richtig `badness=0`.

Für `i` von 1 bis `n-1` : Für `j` von `i+1` bis `n` : wenn `zahlI > zahlJ` dann `badness=badness+1`.

Überraschenderweise sortiert BozoBad auf diese Weise 50 Zahlen in vertretbarer Zeit.

BozoBad kennt keinen Algorithmus zur Lösung, sondern nur einen zur Beurteilung der gegenwärtigen Situation.

Offensichtlich eignet sich BozoBad zur Lösung von Problemen, wenn

- sie mathematisch modellierbar sind;
- ein Bewertungsalgorithmus zur Verfügung steht;
- genug Rechenpower da ist;
- eine suboptimale Lösung besser ist als gar keine.

8.8 Übung

8.8.1 Papier

Sortieren Sie auf einem Blatt Papier die Zahlenfolge 2 7 4 1 6 9 4 aufsteigend nach allen Verfahren. Schreiben Sie nur eine neue Zeile auf, wenn Sie etwas vertauschen müssen. Beispiel:

```

2 7 4 1 6 9 4
   X
2 4 7 1 6 9 4
     X
2 4 1 7 6 9 4

usw.
```

8.8.2 Sort3Verf

Schreiben Sie ein menügesteuertes Programm. Die Anzahl der sortierten Werte sei die Konstante n . Setzen Sie zunächst auf 20, später auf 25000.

```

1 erzeuge byte-Werte von 0 bis 255
2 sortiere nach Bubblesort
3 sortiere nach Maximum
4 sortiere nach Maximum und Minimum
0 exit
```

8.8.4 Quicksort

Mit Hilfsfeld Benutzen Sie in `quicksort` das zu sortierende Feld `a` und ein Hilfsfeld `b`. Anschließend kopieren Sie `b` nach `a`.

Struktogramm Machen Sie zunächst einen Schreibtischtest für (1,3,5,2,4).

```

prozedur quicksort (feld a, l, r)
int i, j, p, h;
i=l; j=r;
p=a[i];
wiederhole
  wenn a[i]<p inkrementiere i;
  wenn a[j]>p dekrementiere j;
  wenn i<j dann
    tausche a[i] und a[j]; inkrementiere i; dekrementiere j;
bis i>j;
wenn j>l dann quicksort (feld a, l, j);
wenn i<r dann quicksort (feld a, i, r);
```

8.8.3 SortVergleich

Schreiben Sie ein Programm, das nach der Anzahl der zu sortierenden Werte fragt ($9 < n < 9999$). Dann erzeugt es immer ein neues `array of byte`, sortiert dieses 5mal und merkt sich die Zeit, die es dafür braucht.

Am Schluß werden die benötigten Zeiten und Verfahren übersichtlich ausgegeben:

```

sortiere 5mal 999 werte...
bubblesort      12.34 sec
maxisort        6.54 sec
minisort        6.45 sec
minimaxisort    4.32 sec
bozobadsort     439.87 sec
```

9 Value vs. Referenz (optional)

Funktionen und Prozeduren kann man Parameter übergeben, mit denen sie arbeiten sollen. Das kennen wir schon.

Bisher haben wir nur `by value` übergeben. Dh, die genannten Variablen werden ausgelesen, und der Funktion oder Prozedur werden die ausgelesenen Werte übergeben.

**Beim `call by value` werden die Werte der Parameter übergeben.
Die Variablen im aufrufenden Programm bleiben dabei konstant.**

Wenn wir nun zB einen Algorithmus zum Sortieren entwickelt haben, wäre es schön, wenn wir der Prozedur `meinSortierVerfahren` ein Feld übergeben könnten, das anschließend sortiert IST.

Das geht in TurboPascal. Wir müssen nur darauf achten, daß wir derProzedur nicht die Werte des Feldes, sondern die Variable, in der das Feld sitzt, übergeben.

```
PROCEDURE inc1 (i : INTEGER);      (* um 1 erhoeen heisst INKREMENTIEREN *)
BEGIN
  i := i + 1;
END;

PROCEDURE inc2 ( VAR i : INTEGER);
BEGIN
  i := i + 1;
END;

BEGIN
  i := 2;
  inc1 (i);
  Writeln (i);
  inc2 (i);
  Writeln (i);
END.
```

**Beim `call by reference` wird nicht der Wert, sondern der Name der Variablen übergeben.
Die Variablen im aufrufenden Programm können dabei geändert werden.**

10 Suchverfahren

10.1 Binärer Baum

10.1.1 Beispiel Bibliothek zu 1000 Büchern

Gegeben sei eine Bibliothek mit exakt 1000 Büchern, die von 1 bis 1000 durchnummeriert sind.

Wie die Bücher auf Gängen, Zimmern und Regalen verteilt sind, spiele keine Rolle.

Wie muß man die Bücher ausschildern, damit man zB Buch 953 möglichst schnell findet?

Lösung 1: Ausschildern nach 100er-Gruppen (1..100, 101..200, ..., 901..1000), dann nach 10er-Gruppen (901..910, 911..920, ..., 991..1000), dann nach 1er-Gruppen, also nach Büchern.

Nach 3 Ebenen zu 10 Schildern haben wir das Buch.

Lösung 2 (Rechnernäher): Immer zwischen 2 Hälften wählen, zB:

501..1000, 751..1000, 825..1000, ..., 952..953, 953.

Nach 10 Ebenen zu 1 Schild (wieso eigentlich nur 1? :)) haben wir das Buch.

Beide Aufteilungsarten kann man als Baum darstellen (Wurzel=oben). Der 10er-Baum hat an (fast) jedem Knoten 10 Zweige, der 2er-Baum 2 Zweige.

In der ersten Lösung muß man im Schnitt 15 Wegweiser lesen, um ans Ziel zu kommen – in der zweiten 10 Wegweiser.

**Eine nicht-ringförmige Ordnungsstruktur aus Knoten und Kanten heißt Baum.
Bäume mit max. 2 neuen Kanten pro Knoten heißt Binärer Baum.**

10.1.2 Beispiel Bibliothek zu 1 000 000 Büchern

Wir können die Wegweiser im 10er- oder im 2er-System aufstellen. Im ersten Fall brauchen wir 6 Ebenen und müssen im Schnitt 30 Schilder lesen, im zweiten Fall brauchen wir 20 Ebenen und müssen 20 Schilder lesen.

Binäre Bäume sind ein pfiffiges Mittel, um innerhalb großer Datenmengen schnell Daten zu finden.

Bedingungen:

- Die Datenmengen müssen sortiert sein.
- Der Binäre Baum muß pfiffig konstruiert sein.

10.1.3 Bibliothek zu 500 Büchern

In die Bibliothek zu 1000 Büchern kommen nachts Einbrecher und stehlen die Bücher 301-800.

Jetzt sind nur noch 500 Bücher da, d.h. unser Binärer Baum kommt theoretisch mit 9 Ebenen aus.

Leider müssen wir entweder die Nummern an vielen Büchern ändern (was ineffektiv wäre) oder den Binären Baum komplett umbauen:

1..500
1..250 251.. 1000
1..125 126..250 251..875 876..1000
usw...

**Binäre Bäume eignen sich gut zum Finden von Daten in großen Datenmengen, die nicht häufig geändert werden.
Änderungen ziehen Neusortierung bzw. Einsortierung und ggf. den Umbau des Binären Baumes nach sich.**

11 Software-Engineering [optional]

11.1 Zweck

Software-Engineering wird eingesetzt, um komplexe Programmprojekte besser, schneller, transparenter und effektiver umsetzen zu können.

11.2 Beispiel

Gegeben sei ein mittelständisches Unternehmen von 1000 MA, das eine bestimmte Sorte Maschinen produziert.

Da es ein deutsches Unternehmen ist, gelten folgende Besonderheiten:

- Es muß um seine nationalen und internationalen Kunden kämpfen.
- Es muß besser sein als die Konkurrenz:
 - schneller liefern;
 - besserer Service;
 - bessere Erweiterbarkeit der Maschinen;
 - zuverlässigere Maschinen;
 - Maschinen mit sinnvollen Alleinstellungsmerkmalen;
- es muß intern hocheffizient und gleichzeitig hochflexibel sein²;
- Vertrieb, Konstruktion, Produktion, Lager, Logistik und Management müssen zielführend und effektiv zusammenarbeiten;
- es hat eine klar gegliederte Aufbau-Organisation;
- es hat eine gewachsene IT-Struktur;
- es hat ein komplexes Wissens- und Innovationsmanagement;
- es muß jederzeit Raum für Sonderwünsche des Chefs, der Kunden, der Genehmigungsbehörden und Zertifizierungsorganisationen vorhanden sein.

Diese Liste ist unvollständig.

11.3 Vorgehensweise

Die Vorgehensweise beim SWE ist methodisch und planvoll:

1. Analyse;
2. Design;
3. Programmierung;

Dies ist ein iterativer Prozeß.

Genau wie der Kunde selbst ist auch das Pflichtenheft dynamisch, kontinuierlichen Veränderungen und Verbesserungen unterworfen und eigentlich nie ganz fertig.

Außerdem findet man beim Durchführen häufig Verbesserungsmöglichkeiten, bessere Lösungsansätze, zusätzliche Möglichkeiten und unlösbare Probleme.

Zu den Aufgaben des Software-Ingenieurs gehört deshalb auch der permanente Kundenkontakt. Entwerfen muß er die Software-Architektur zwar selber, aber er muß sie dem Kunden auch verkaufen, d.h. diesen vom Nutzen seiner Arbeit für den Auftraggeber kontinuierlich überzeugen.

²Es ist normalerweise nicht möglich, etwas Neues auf Anhieb hocheffizient zu machen, da Effizienz ist in der Regel das Ergebnis eines Optimierungsprozesses ist.

12 Die einzelnen Phasen

12.1 Analyse

Bevor man loslegt, muß zunächst bestimmt werden, welche Aufgaben die neue Software erfüllen soll. Dabei gibt es Konflikte, da man mehrere Ziele gleichzeitig erreichen will:

- wirkungsvolle Verbesserungen,
- zu geringen Kosten,
- mit hohem Unternehmens- und Kundennutzen,
- bei hoher Akzeptanz der neuen Software bei Management (die sie bezahlen) UND Mitarbeitern (die damit arbeiten müssen).

12.2 Beispiel: Mittelständisches Unternehmen

In einem mittelständischen Unternehmen soll der gesamte interne Informationsfluß über die einzelnen Projekte verbessert werden.

Dabei wird der Prozeß "von Kundeninteresse bis Zahlungseingang" untersucht und zweckmäßigerweise in Unterprozesse aufgeteilt, zB

1. Beratungsgespräche;
2. Verhandlungen und Vertrag;
3. Konstruktion;
4. Produktion;
5. Lieferung und Montage;
6. Inbetriebnahme und Abnahme;
7. Rechnung und Rechnungsverfolgung.

Die Aufteilung muß fortgeführt werden, bis man eine Prozeßkette vorliegen hat, in der jede Übergabestelle mit Eingangs- und Ausgangsobjekten, Verantwortlichkeiten, Kontrollmechanismen etc. klar definiert ist.

Die Prozeß-Modelle müssen auch Eventualitäten abdecken wie zB:

1. Lieferverzug eines Lieferanten;
2. Ausfall von Schlüsselpersonen;
3. unvorhergesehene Probleme;
4. Liquiditätsengpässe.

12.2.1 Informationsbeschaffung im Unternehmen

In der Regel geben Management und Mitarbeiter nur dann vollständige und richtige Informationen über Abläufe und Methoden, wenn sie vom Sinn der Informationsübergabe überzeugt sind und fest glauben, daß sie sich und ihre Position dabei keinesfalls verschlechtern.

Am Schluß der Analyse steht das Finden von Verbesserungsmöglichkeiten.

12.3 Design

12.3.1 Entscheidung

Am Anfang des Designs steht die Entscheidung, welche Prozesse oder Rechnersysteme auf welche Weise umgestaltet, modelliert und programmiert werden sollen.

12.3.2 Lastenheft, Pflichtenheft

Dann werden die Aufgaben des neuen Software-Systems festgehalten.

Das Pflichtenheft soll so dynamisch sein, daß man nützliche, machbare Ideen, die sich während der Abwicklung ergeben, noch einbauen kann, und so statisch, daß es nicht beliebig wirkt.

Hierbei sollten berechnete Sonderwünsche des Chefs (zB "Ich will überall reingucken können!"), verschiedener Abteilungen (zB "Wir können hier alle nur Powerpoint.") und einzelner User ("Wo ist meine Linux-Console?") - soweit vertretbar - eingebunden werden, um die Akzeptanz des neuen Systems zu erhöhen.

12.3.3 Designprinzipien: Hierarchisierung und Modularisierung

Beispiel

Ein Autobauer erfindet nicht für jedes Auto das Rad oder das Getriebe etc neu, sondern baut es, wo möglich, aus vorhandenen, getesteten, genormten, preiswerten Teilen zusammen.

Hierarchisierung ist die Aufteilung einer Aufgabe in mehrere Prozesse, die jeder für sich übersichtlicher sind als die Gesamtaufgabe - mit den resultierenden Vorteilen: kürzere Entwicklungszeit, bessere Testmöglichkeit, bessere Wartbarkeit, bessere Wiederverwendbarkeit.

Modularisierung ist das Aufteilen von Aufgaben in Module. Dabei ist das Innenleben eines Moduls für seine Benutzer (die anderen Module) irrelevant (jedenfalls solange es fehlerfrei ist).

Niemand muß wissen, was in einem bestimmten Modul passiert. Um es zu benutzen, müssen nur die Software-Schnittstellen (die Übergabepunkte von Daten und Botschaften) genau bekannt sein.

Wie der Auto-Bauer manches von Dritten kauft, muß man nach der Modularisierung auch überprüfen, welchen Teil des Codes man selbst schreiben will, ob man Code aus vorangegangenen Projekten wieder verwenden kann, oder ob man für bestimmte Teilaufgaben auf Programme Dritter zurückgreift.

12.4 UML - Unified Modeling Language

Durch diese Vorgehensweise werden die Beziehungen zwischen den Modulen kritisch für die Funktion des Programms (Software-Systems).

Verbale Beschreibungen sind den Anforderungen oft nicht gewachsen. Grafische Beschreibungen sind besser.

Es gab eine Fülle von Ansätzen zur Standardisierung der grafischen Darstellung der Modul-Beziehungen, die mittlerweile in einem Standard zusammengefaßt wurden: UML.

Beziehungen zwischen Akteuren eines Software-Systems werden in UML (Unified Modeling Language) grafisch dargestellt.

12.5 Programmierung

Der modulare Aufbau des Pflichtenheftes ist möglichst weit nachzubilden.

Code, der auf sensible Unternehmensprozesse zugreift, muß sorgfältig erstellt und getestet werden, zB anhand eines Prototyps.

Versionsmanagement und Dokumentation sind kritisch für spätere Fehlersuche, Verbesserungen, Erweiterungen und Code-Wiederverwendbarkeit.

12.5.1 Dokumentation

Eine vollständige Dokumentation umfaßt

- Pflichtenheft;
- Produktmodell;
- Konzept der Benutzungsoberfläche;
- Benutzer-Handbuch.

12.5.2 CASE - Computer Aided Software-Engineering

Aufgrund der Flexibilität und Eindeutigkeit von UML sind die Abläufe vollständig und konsistent (widerspruchsfrei) dargestellt. Die Umsetzung in Code ist anschließend mehr oder weniger Routine.

Da Rechner existieren, um ihre Benutzer von Routineaufgaben zu entlasten, gibt es Ansätze, aus einem Malprogramm für UML-Diagramme den zugehörigen Programmcode automatisch erzeugen zu lassen.

Verschiedene CASE-Systeme werden dieser Aufgabe in unterschiedlichem Umfang gerecht.

12.6 Lebensdauer

Ein Unternehmensmodell ist eine Dauer-Baustelle, in der folgende Fehler auftauchen können:

- Benutzerunfreundlichkeit;
- nicht flexibel genug für Innovationen;
- zu flexibel in Hinsicht auf sensible Daten;
- die Mitarbeiter fühlen sich überwacht;
- der Chef muß sich an eine neue Benutzeroberfläche gewöhnen;
- manche Manager verlieren Mitarbeiter und damit Status;
- unzuverlässig.

Diese Liste ist unvollständig.

Dieser Zustand ist aber normal. Solange das Unternehmen dabei gedeiht, innovativ ist, neue Märkte erobert und wächst, muß das Modell mitwachsen.

12.7 Multi-Tier-Architekturen

Es hat sich durchgesetzt, die Informationssysteme in einer Drei-Schichten-Architektur (three-tier³ architecture) zu modellieren.

Diese Architektur besteht aus

- der Benutzungsoberfläche (GUI-Schicht)
- der Fachkonzeptschicht
- und der Schicht für die Datenspeicherung (Datenhaltungsschicht)

Die GUI-Schicht ist für die Darstellung aller Daten und für die Dialogführung zuständig⁴.

In der Fachkonzeptschicht wird der Kern der Anwendung losgelöst von Speicherung und Präsentation festgelegt. Für den Austausch mit der Benutzungsoberfläche werden Standardmethoden (setAttribut und getAttribut) zur Verfügung gestellt. Die Fachkonzeptschicht besitzt kein Wissen über die Benutzungsoberfläche. Bei einer Änderung der Benutzungsoberfläche muss das Fachkonzept nicht angepasst werden.

³tier [ˈtaie], engl.: Schicht

⁴GUI - Graphical User Environment - grafische Benutzeroberfläche.

Die Datenhaltungsschicht sorgt für die Datenspeicherung, z. B. in einer Datenbank. Die GUI-Schicht weiß nicht wie der Zugriff auf die Daten erfolgt.

“Die Entkoppelung von Benutzungsoberfläche und Fachkonzept ist heute ein Grundprinzip des Softwareentwurfs. Viele dieser Ideen haben ihren Ursprung in der Architektur MVC, die erstmalig in Smalltalk-80 verwendet wurde.” (Heide Balzert 1999, S. 373)

12.7.1 Vorteile von Multi-Tier-Architekturen

Der Vorteil dieses Vorgehens liegt in der guten Wartbarkeit. Die Hardwareabhängigkeiten können in einer Schicht isoliert werden. Änderungen der Benutzungsoberfläche sind unabhängig vom Rest des Systems.

12.7.2 Noch mehr Schichten

Eine Weiterentwicklung ist die Mehr-Schichten-Architektur (multi-tier architecture). Dieses Vorgehen ergänzt die Drei-Schichten-Architektur um eine Fachkonzept-Zugriffsschicht und eine Datenhaltungs-Zugriffsschicht. Auf diese Weise werden einerseits komplexe Fachkonzeptstrukturen auf einfache Datentypen der GUI-Schicht abgebildet und andererseits werden die Strukturen der Fachklassen in Typen der Datenhaltungsschicht (z.B. relationale oder objektorientierte Datenbanken) konvertiert.

13 Softwarequalität

Quelle: Hartmut Härtl, <http://www.oszhdl.be.schule.de/index.htm>

Qualitätsmerkmale können nicht nachträglich definiert und in ein Produkt hineingeprüft werden, sondern müssen bereits beim Systementwurf konstruktiv berücksichtigt werden. Nachträgliche Korrekturen oder Änderungen an fertiggestellter Software bedeuten in aller Regel ein Mehrfaches an zusätzlichem Entwicklungsaufwand. Nachfolgend werden die wichtigsten Qualitätsmerkmale für Softwareprodukte aufgeführt.

13.1 Gebrauchstauglichkeit (Usability)

Usability wird von der International Organization of Standardization (ISO) als Ausmaß definiert, zu welchem ein Benutzer ein bestimmtes Produkt in einem spezifischen Umfeld zur effizienten Erledigung seiner Aufgaben nutzen kann.

a) Funktionalität

Das System muss den Anforderungen der Anwender entsprechen.

b) Zuverlässigkeit (reliability)

Arbeitet das Produkt im vorgesehenen Zeitraum und Anwendungsbereich stabil und fehlerfrei?

c) Integrität / Korrektheit

Ist das Produkt gegen Mißbrauch geschützt, ist es vertrauenswürdig?

d) Benutzerfreundlichkeit

Grad, in dem das Produkt seine Aufgaben erfüllt, ohne Zeit und Energie des Benutzers zu verschwenden bzw. ohne seine Motivation herabzusetzen. (Ist es leicht erlernen und benutzbar; bereitet seine Benutzung wenigstens kein Missvergnügen?)

e) Robustheit

Die Fähigkeit des Systems auch unter außergewöhnlichen Bedingungen korrekt zu funktionieren: Abweisung von Falscheingaben, Erkennen von Störungen und Fehlern, wie z.B. Eingabe eines falschen Datums, Aufheben einer versehentlichen Löschung, Eingabedatei ist leer oder nicht vorhanden, Stromausfall, Verhalten bei extremer Systemauslastung.

f) Effizienz

Bezüglich Laufzeit und belegten Systemressourcen (Speicher, Geräte, Netzwerklast).

13.2 Revisionsfähigkeit

a) Änderbarkeit / Erweiterbarkeit / Ausbaufähigkeit

Aufwand für die Lokalisierung und Durchführung von Änderungen in einem Softwareprodukt, wenn die Art der gewünschten Änderung festliegt.

b) Wartbarkeit

Aufwand zur Lokalisierung und Behebung von Fehlerursachen, wenn die Fehlerwirkung bekannt ist.

c) Überprüfbarkeit (Verifizierbarkeit)

Aufwand für die Vorbereitung, Durchführung und Auswertung von Kontrollen, mit denen die Einhaltung der Spezifikation überprüft werden kann. (Nach jeder Änderung des Produktes muß kontrolliert werden, ob es noch einwandfrei funktioniert.)

13.3 Transitionsfähigkeit

Können das Produkt oder Produktteile auch noch anderweitig genutzt werden ?

a) Portabilität

Mit welchem Aufwand kann das Softwareprodukt auf andere Hardwaresysteme oder in eine andere Softwareumgebung übertragen werden?

b) Wiederverwendbarkeit

Können bestimmte Module oder Objekte auch in anderen Systemen verwendet werden?

c) Interoperabilität / Kompatibilität

Kann das Produkt auch mit anderen Systeme zusammenarbeiten oder mit diesen Daten austauschen? (Z.B. Kann ein Fakturierungsprogramm mit dem Buchhaltungsprogramm, der betrieblichen Datenbank oder mit der Bank Daten austauschen? Bietet das Programm definierte Schnittstellen für bestimmte Internetdienste bereit?)