

Übungsbogen zur Vorlesung Einführung in die Informatik SS99 von Prof. Ertelt; FHO-Emden. Erklärungen von Sascha Atrops. Keine Gewähr, ob hier irgendwas stimmt ;) Dankschreiben/Fragen/kleine und große Geldgeschenke an sascha@atrops.com. Beschwerden/Mailterror können bei nichtbestandener Klausur alternativ auch an ertelt@et-inf.fho-empden.de. Ich kann da echt nichts für ;) Rechtschreibfehler sind beabsichtigt und dienen der allgemeinen Verständlichkeit!

Besten Dank und viel Glück in der Klausur!
-- Xin --

3-7; 7-6 fehlt noch

Zur 2. Version: Hier stehen keine Musterlösungen und viele Antworten sind lediglich weitergehende Erklärungen, wenn das Skript keine klare Antworten liefert – dieser Text ersetzt das Skript nicht (daher auch die Verweise...).

Beschwerden über Rechtschreibung in der ersten Version überhöre ich gekommt, da ich hier keinen Rechtschreibwettbewerb veranstalte, sondern lediglich mein Verständnis zum Fragebogen zum Besten gebe. Gegen eine Überarbeitung durch diejenige Person habe ich nichts einzuwenden....

Zur 3. Version:

Da das Ding hier recht beliebt zu sein scheint, gibt's nochmal 'ne neue Auflage ;))

In den letzten Semestern hat sich einiges geändert. Alles, was ich ins Netz stelle aus meinem Studium landet nun auf meiner Webpage atrops.com und ist dort jeweils in aktueller Version zu finden – zumindest, bis mein Studium in ferner Zukunft zu Ende ist ;)

Die in der vorherigen Version angegebene E-Mailadresse ist nicht mehr gültig!

Besucht meine Page, schreibt Lobeshymnen in mein Gästebuch, damit ich meinen Eltern klarmachen kann, daß meine schlechte Noten oder Exmatrikulation definitiv die Schuld der Professoren ist ;)

An sofort dürfen gefundene Rechtschreibfehler behalten werden.

Viel Erfolg in Euren Klausuren.
Sascha 'Xin' Atrops

1. Grundbegriffe der Informatik

1. Erläutern Sie den Begriff **Information!**

Information gilt als dritte elementare Größe neben Materie und Energie und bedeutet soviel wie interpretierte Nachricht. Interpretiert muß hier so gesehen werden, daß jede Nachricht von jeder Person/System unterschiedlich verstanden/interpretiert werden kann, so daß es trotz identischen ‚Wortlauts‘ zu unterschiedlichem Verständnis/Reaktionen kommen kann. (vgl. 1-6)

2. Erklären Sie den Zusammenhang von **Syntax** und **Semantik** aus der Sicht der Informatik.

Unter Syntax wird die richtige Schreibweise von Wörtern bzw. der für den Computer verständliche Satzbau in DOS-Befehlen oder Programmiersprachen verstanden. Die Semantik klärt die Bedeutung von Begriffen, die syntaktisch richtig geschrieben sind und für den Computer verständlich. Die Semantik des ‚dir‘-Befehls besagt, daß das Inhaltsverzeichnis angezeigt werden soll. (vgl. 1-10)

3. Erklären Sie die Begriffe **Codierung**, **Byte**, **Bit**, **analog** und **digital!**

Codierung: Die Codierung beschreibt die Art und Weise, wie Informationen im Speicher abgelegt werden. So werden die Informationen eines Bildes so codiert, daß das Bild in Zeilen und Bildpunkte strukturiert wird jedem Bildpunkt in jeder Zeile ein Farbwert zugeordnet wird. Die Strukturierung erfaßt in der Regel nur die wesentlichen Aspekte, die Information kann beliebig detailliert dargestellt werden, wird jedoch im Regelfall nur eine vereinfachte Form der Abbildung darstellen. So wird von einer Person z.B. nur die Name und Adresse in einer Kundendatei gespeichert, allerdings nicht die Augenfarbe, Schuhgröße etc. Die Abbildung ist also nicht vollständig: die wichtigen Aspekte sind strukturiert und werden so im Arbeitsspeicher abgelegt. (vgl. 1-8)

Des weiteren werden Programmquelltexte in Maschinensprache (maschinenlesbar) codiert, um den Rechner Programme aus dem Speicher heraus ausführen zu lassen (vgl. 2-2)

Byte: Ein Byte ist die kleinste vom Computer direkt ansprechbare Größe im Arbeitsspeicher. In einem Byte können Werte zwischen –128 bis 127 (bzw. 0 bis

255) gespeichert werden. Dies geschieht durch eine Aneinanderreihung von 8 Bits. (keine Quelle in Kapitel 1 gefunden)

Bit: Ein Bit ist die kleinste Einheit, in der Informationen gespeichert werden. Der Prozessor des Computers kann Bits nicht einzeln ansprechen, sondern muß sie jeweils Byteweise lesen und schreiben. Ein Bit besteht nur aus der Information Ja oder Nein (Wahr/Falsch, Null/Eins, Ein/Aus, etc...). Es kann immer nur einen der beiden Zustände annehmen, es gibt keine Zwischenzustände zwischen Ja oder Nein (...). (vgl. 1-12

Analog: Unter einer analogen Speicherung versteht man das Aufzeichnen von nicht exakten Werten. Es werden gewissermaßen ‚Abschätzungen‘ aufgezeichnet: z.B. auf einer Audio- oder Videokassette. Die Qualität des analog aufgezeichneten Signals ist sehr störanfällig (Magnetband verschmutzt, nutzt ab...), so daß das aufgezeichnete Signal nicht mehr exakt wiedergegeben werden kann. Analoge Aufzeichnungen und Übertragungen (z.B. Telefonleitung, Chinch-Verkabelung an der Stereoanlage) sind technisch einfach zu realisieren. (vgl. 1-12 ff)

Digital: Um digitale Aufzeichnungen zu erstellen bedarf es eines höheren technischen Aufwands, da die zu speichernden Daten zunächst codiert werden müssen. Statt der analogen Welle auf dem Tonband wird auf einer Compact Disc die Lautstärke der Musik rund 40000mal pro Sekunde aufgezeichnet. Diese Aufzeichnung läßt sich jedoch nahezu beliebig oft abspielen und übertragen ohne Qualitätsverluste (z.B. Glasfaserverkabelung an qualitativen Stereoanlagen). (vgl. 1-12)

4. Geben Sie drei charakteristische Interpretationsmöglichkeit für **Bitmuster** an!

Beispiele für die Interpretation von Bitmustern geben Farbtabelle (z.B. 0001 für Rot, 0010 für Grün, 0011 für Violett etc...), Maschinensprachebefehle, oder Zeichensatztabellen (z.B. ASCII (7-Bit pro Zeichen), ISO Latin-1 (8-Bit pro Zeichen)). (vgl. 1-15 – 1-17)

2. Grundlagen des Programmierens

1. Durch welche wesentlichen Eigenschaften ist eine höhere **Programmiersprache** (in Abgrenzung zu anderen) gekennzeichnet.

Höhere Programmiersprachen kennzeichnen sich durch eine abstraktere Beschreibung des Problems. Das bedeutet, daß eine Programmiersprache niedrigeren Levels mehr auf die Maschine eingeht, auf der es programmiert wurde, als eine höhere, welche erst mittels eines Compilers für die jeweilige Maschine umgesetzt wird. Sogenannte High-Level-Programmiersprachen widmen sich konkreten Problemen, wie die Maschine das nachher umgesetzt ist nicht mehr das Problem des Programmierers, sondern des Compilers. Somit lassen sich Programme, welche in höheren Programmiersprachen geschrieben wurden auch leichter von einer Rechnerplattform auf eine andere konvertieren (z.B. Windowsprogramme nach Linux). Höhere Programmiersprachen bieten oftmals auch bereits eingebaute Lösungen für Probleme, die häufig auftreten. Programmteile und Daten (Variabel) bekommen konkrete Namen und können mit diesen auch direkt angesprochen, aufgerufen oder verändert werden. (vgl. 2-4; 2-9)

2. Erläutern Sie die **formalen Bestandteile** eines Programms!

(keine konkrete Quelle im 2. Kapitel gefunden)

3. Erläutern Sie die Begriffe **Datenobjekt**, **Ausdruck** und **Funktion**!

***Datenobjekt:** Unter einem Datenobjekt wird ein Speicherbereich verstanden, in dem Daten eines Programmes abgelegt werden können. Man deklariert ihn in den meisten Programmiersprachen so, daß man angibt, von welchem Typ er ist (z.B. eine Zahl, ein Buchstabe, eine Struktur in der Name und Adresse einer Person o.ä. gespeichert ist) und ihm einen Namen gibt. Mit diesem Namen kann man nun das Datenobjekt ansprechen und damit auf den Speicher zugreifen, den das Datenobjekt belegt. So kann man entsprechend die Zahl, den Buchstaben oder die Adresse – je nach dem, welchen Datentyp man eben angefordert hat – im Speicher ablegen. (vgl. 2-12)*

Ausdruck: Unter einem Ausdruck versteht man ein Buchstaben/Wortgebilde, daß der Semantik und der Syntax der Sprache entspricht, die man programmiert. Ein vollständiger Befehl mit allen Parametern, Semikolons etc. (Parameter sind die Zusatzanweisungen, die man einem Befehl oder einer Funktion mitübereben muß, damit diese weiß, was sie tun soll: z.B. printf(PARAMETER); bzw. Output.WriteChar(PARAMETER);) wird Ausdruck genannt. Dies können allerdings auch Anweisungen sein, die keinen direkt erkennbaren Befehl besitzen, wie zum Beispiel eine Wertzuweisung einer Variablen (a:= 4;). (vgl. 2-13)

Funktion: In einer Funktion beschreibt man ein Unterprogramm, daß nicht zum den Standard-Befehlssatz (if, then, else, while, for,...) einer Sprache gehört, sondern nachträglich in die Sprache integriert wurde. Sie besitzt einen Namen (z.B. printf(); WriteChar(); etc.) und man kann ihr Parameter übergeben, um Einfluß auf ihren Ablauf zu nehmen (ihr zum Beispiel einen Buchstaben anzugeben, den sie auf dem Bildschirm ausgeben soll). Funktionen können einen Rückgabewert (üblicherweise eine Zahl oder eine Adresse – also auch eine Zahl) besitzen, müssen aber nicht. Es gibt auch Funktionen, die keinen Wert zurückliefern. Sobald die Funktion beendet ist, kehrt sie zu dem Punkt im Programm zurück, von dem sie aufgerufen wurde. (vgl. 2-13)

4. Erläutern Sie die das prinzipielle Verfahren der Umsetzung von **abstrakten** Programmstrukturen in eine Maschinensprache!

(keine konkrete Quelle im Kapitel 2 gefunden)

Die Umsetzung von abstrakten Programmstrukturen in die Maschinensprache wird mit einem Programm geleistet, welches man als ‚Compiler‘ bezeichnet. Dieser Compiler überprüft Syntax und Semantik der Programmquelltextes und übersetzt ihn, sofern er keine erkennbaren Fehler enthält in Maschinensprachebefehle. Abstrakte Konstruktionen, wie z.B. objektorientierte Programmierweise, werden mit elementaren Grundbefehlen umschrieben, die der Prozessor beherrscht.

5. Was verstehen Sie unter **Programmierzklus**? Nennen Sie seine Bestandteile und die jeweils erforderlichen Software-Werkzeuge! Welche Gründe können den Programmierer zwingen, erneut in den Zyklus (-und wo dann) einzutreten?

Der Programmierzklus beschreibt die einzelnen Etappen, die benötigt werden, um von der Idee eines Programms zum fertigen Produkt zu kommen. Er beginnt mit dem Programmwurf, also der Ausarbeitung der eigentlichen Idee. Dieser Entwurf wird nun mittels einer Programmiersprache codiert, so daß der Compiler dieser Programmiersprache den Quellcode verstehen kann. Diesen Vorgang des eigentlichen Programmierens ist als ‚Codieren‘ bezeichnet. Den geschriebenen Quelltext wird nun mittels des Compilers übersetzt (‚Übersetzen‘). Der Compiler erzeugt nun ein sogenanntes Objekt-File, welches vom Rechner noch nicht ausführbar ist. In diesem Objektfile sind noch keine direkten Verweise zu den einzelnen Funktionen bzw. Variablen gemacht worden, die im ausführbaren Programm benötigt werden. Diese werden im nächsten Schritt eingebaut: dem ‚Linken‘. Nun kann das Programm mittels eines Debuggers getestet werden, bzw. mit dem Computer direkt ausgeführt werden.

Der Programmierzklus wird unterbrochen im Fall, daß ein Fehler gefunden wurde. Syntaktische Fehler meldet der Compiler direkt und bricht den Zyklus ab (‚Syntaktischer Fehler‘). Für den Fall, daß Verweise nicht gefunden werden, weil z.B. eine Variable oder eine Funktion noch nicht deklariert bzw. programmiert wurde, meldet der Linker einen Fehler und bricht ebenfalls ab(‚Referenzfehler‘). Sofern keine syntaktischen Fehler und keine Referenzfehler aufgetreten sind, kann das Programm ausgeführt werden. Sollte es jedoch nicht das tun, was erwartet wurde, sind ‚logische Fehler‘ im Programm. Dies kann durch den Debugger erkannt werden, bzw. dadurch, daß der Computer nicht das macht, was ursprünglich geplant war. In diesem Falle, muß der Quelltext überarbeitet werden und der Zyklus beim Compilieren neu gestartet werden. (vgl. 2-21)

3. Zahlendarstellung für Computer

1. Durch welche wesentlichen Eigenschaften sind **reelle Zahlen** gekennzeichnet? Welches Konzept wird für ihre **Speicherung** verwendet und welche speziellen Probleme sind bei ihrer **Verarbeitung** in einem Computer zu berücksichtigen?

*Reelle Zahlen sind dadurch gekennzeichnet, daß sie nicht endlich sind, also einen unendlich langen Nachkomma-Bereich besitzen. Da Computer allerdings keinen unendlichen Speicher besitzen, müssen Sie gerundet gespeichert werden. Damit entstehen Ungenauigkeiten beim Rechnen. (vgl. 3-8)
Die Zahlen werden mittels des Datentyps ‚real‘ gespeichert, der Aufbau von ShortRealzahlen ist in Frage 3-9 erklärt.*

2. Was läßt sich zu den **hexadezimalen Zahlen** sagen und wofür werden sie benötigt?

*Hexadezimale Zahlen sind eine weitere Möglichkeit Zahlen darzustellen. Unterschied zu den Dezimalen ist die Basis 16 (hexadezimal). Es existieren 16 verschiedene Ziffern, wobei die Ziffern von 0-9 dem im dezimalen Zahlensystem gleichwertig sind, die Zahl 10 wird durch die Ziffer ‚A‘ dargestellt. Entsprechendes gilt für 11 = ‚B‘; 12 = ‚C‘; 13= ‚D‘; 14= ‚E‘ und 15= ‚F‘. Mit der Null existieren nun 16 verschiedene Ziffern.
Wichtig ist nun, daß jede Stelle weiter Links nun nicht mehr den zehnfachen Wert der rechtsanliegenden Stelle besitzt, sondern den sechszehnfachen Wert! Die Zahl ‚12‘ in hexadezimaler Darstellung entspricht damit $1*16+2*1$, also dezimal gesehen ist ‚12 (Basis 16=hexadezimal)‘ = ‚18 (Basis 10 = dezimal)‘.
Vorteilhaft ist, daß ein sogenanntes Nibble (bestehend aus 4 Bits) genau die Zahlen von 0-15 darstellen kann, ein Nibble also mit einer hexadezimalen Ziffer darstellbar ist. Da ein Byte aus zwei Nibbles besteht, ist ein Byte nun mit zwei hexadezimalen Ziffern darstellbar, indem man die Nibbles einfach direkt ins hexadezimale übersetzt:*

Byte (binär)	Nibble(dezimal; hex)	Nibble2 (dezimal; hex)	Hexadezimal
01011110	0101 (9, 9 hex)	1110 (13, D hex)	9D

*Dezimalwert: $9*16 + ,D'*1 = 9*16 + 13*1 = 144+13 = 157$ (dezimal)
(vgl.3-5)*

3. Ermitteln Sie die **Oktal-Darstellung** der **Dezimalzahl 65521,375!** Verdeutlichen Sie die Konvertierungsschritte durch Angabe aller Zwischenergebnisse.

(ich habe keine direkte Quelle im Skript Kapitel 3 gefunden)

Zunächst ist mit der Oktal-Darstellung die Darstellung mit Zahlen der Basis 8 gemeint (vgl. 3-5).

Als erstes nimmt man den Ganzzahlteil (hier 65521) und teilt ihn durch die Basis (hier 8).

Dabei ermittelt man, wie oft die Zahl sich durch 8 ganz teilen läßt und schreibt sich den Rest auf:

$$\begin{array}{rcl}
 65521 / 8 = & 8190 & \text{Rest } 1 \\
 8190 / 8 = & 1023 & \text{Rest } 6 \\
 1023 / 8 = & 127 & \text{Rest } 7 \\
 127 / 8 = & 15 & \text{Rest } 7 \\
 15 / 8 = & 1 & \text{Rest } 7 \\
 1 / 8 = & 0 & \text{Rest } 1
 \end{array}$$

Es bleibt kein Wert zum Teilen übrig, bzw. der Rest wird ewig Null sein.

ACHTUNG:

*Das Ergebnis muß von unter nach oben gelesen werden!
Würde man also weiterrechnen, würde man lediglich führende Nullen erhalten.*

65521 (dez) ist also gleich mit 177761 (okt).

Die Probe:

$$\begin{aligned}
 & 1*8^5 + 7*8^4 + 7*8^3 + 7*8^2 + 6*8^1 + 1*8^0 = \\
 & 32768 + 28672 + 3584 + 448 + 48 + 1 = \\
 & 65521
 \end{aligned}$$

(eine klare Anleitung zur Berechnung der Nachkommastelle habe ich nicht gefunden, daher gehe ich das ganze mal folgendermaßen an:)

$0,375 * 8 = 3,000 \Rightarrow$ Nachkommastelle 3; Rest 0,000 \Rightarrow Fertig

$\Rightarrow 65521,375$ (dez) = 177761,3 (okt)

Um das ganze weiter zu verdeutlichen möchte ich nochmals den Wert 0,466796875 ins Oktalsystem umrechnen (no panic! - sieht komplizierter aus, als es ist!):

$0,466796875 * 8 = 3,734375 \Rightarrow$ Nachkommastelle 3; Rest 0,734375

$0,734375 * 8 = 5,875 \Rightarrow$ Nachkommastelle 5; Rest 0,875

$0,875 * 8 = 7,000 \Rightarrow$ Nachkommastelle 7; Rest 0,0 \Rightarrow Fertig.

ACHTUNG: Bei den Nachkommastellen muß also **multipliziert** werden und normal von oben nach unten gelesen werden!

$\Rightarrow 0,466796875$ (dez) = 0,357 (okt)

Die Probe: $0*8^0 + 3*8^{-1} + 5*8^{-2} + 7*8^{-3} =$
 $0 + 0,375 + 0,078125 + 0,013671875 =$
 $0,466796875$

4. Skizzieren Sie einen Algorithmus zur **Multiplikation binärer Zahlen!**

Gegeben sind jeweils Faktor1 und Faktor2

Möglichkeit 1:

Man deklariert sich eine Variabel, setzt diese auf Null und addiert Faktor1 sooft mal auf, wie es in der Variablen Faktor 2 angegeben ist: (vgl. 3-10)

Beispiel in C:

```
long multiplic(long, long Faktor2)
{
    long result = 0;

    if(Faktor1)
        while(Faktor2)
        {
            result = result + Faktor1;
            Faktor2 = Faktor2 - 1;
        }
    return(result);
}
```

Möglichkeit 2:

Es besteht weiterhin die Möglichkeit auf die Stellen im einzelnen einzugehen: (vgl. 3-11).

Auch hier wird eine Ergebnis-Variable benötigt, die zunächst auf Null gesetzt werden muß. In diesem Fall nimmt man sich einen Faktor (1) und zählt die Anzahl der Stellen. Man beginnt an der linken Seite und addiert entsprechend der linken Stelle mal den anderen Faktor (2) auf. Für den Fall, daß es rechts noch eine weitere Stelle gibt, so multipliziert man das Ergebnis mit 10 (10er Basis!) und beginnt von vorne, bis man alle Stellen durch hat.

Das Prinzip des Algorithmus als Beispiel:

123 * 11:

Faktor1 = 123;

Faktor2 = 11;

result = 0;

betrachte die erste Stelle von Faktor 1: [1]23.

Es wird entsprechend oft Faktor2 aufaddiert:

result = result + Faktor2; ⇒ result = 0 + 11; ⇒ result = 11;

es existiert noch eine stelle rechts der Aktuellen, also mal 10 und Faktor1 eine Stelle weiter recht einstellen:

result = result * 10; ⇒ result = 11 * 10; ⇒ result = 110;

ab jetzt wird die zweite Stelle von Faktor1 betrachtet: 1[2]3

Es wird entsprechend oft Faktor2 aufaddiert:

result = result + Faktor2 + Faktor2 ⇒

result = 110+11+11; ⇒ result = 132;

es existiert noch eine stelle rechts der Aktuellen, also mal 10 und Faktor1 eine Stelle weiter recht einstellen:

result = result * 10; ⇒ result = 132 * 10; ⇒ result = 1320;

ab jetzt wird die dritte Stelle von Faktor1 betrachtet: 12[3]

Es wird entsprechend oft Faktor2 aufaddiert:

result = result + Faktor2 + Faktor2 + Faktor2 ⇒

result = 1320+11+11+11; ⇒ result = 1353;

(vgl. 3-11)

5. Skizzieren Sie einen Algorithmus zur **Division binärer Zahlen!**

Auch hier zwei Möglichkeiten, wobei ich allerdings nur die leichtere skizzieren werde.

Analog zur Multiplikation wird hier einfach, statt addiert, subtrahiert. Man übergibt Zähler und Nenner und so wird der Nenner einfach sooft von Zähler subtrahiert, bis der Zähler kleiner als der Nenner ist. Man zählt einfach, wie oft man den Zähler abziehen kann.

Beispiele in C:

Zur Bestimmung der Division:

```
long div(long Zaehler, long Nenner)
{
    long result = NULL;

    while(Zaehler > Nenner)
    {
        result ++;
        Zaehler = Zaehler - Nenner;
    }

    return(result);
}
```

Zur Bestimmung der Restes gibt man statt dessen einfach zurück, was vom Zähler übrig bleibt:

```
long mod(long Zaehler, long Nenner)
{
    while(Zaehler > Nenner)
        Zaehler = Zaehler - Nenner;

    return(Zaehler);
}
```

Zur 2. Möglichkeit:

Prinzipiell läuft die zweite Möglichkeit wieder analog zur Multiplikation über die einzelnen Stellen. Da moderne Prozessoren den ganzen Kram jedoch in einer einzigen Assembler-Anweisung erledigen, ist es sicherlich nicht sinnvoll, sich über solche Konstrukte den Kopf zu zerbrechen, und da der Text heute noch fertig werden soll, skizzieren wir eben nur eine Möglichkeit, wie verlangt.

6. Lösen Sie die Multiplikationsaufgabe $131 \cdot 11$ in binärer Darstellung! Verdeutlichen Sie die Konvertierung dezimal \Rightarrow binär durch Angabe aller Zwischenergebnisse!

$131 / 2 = 65 \text{ R } 1$	$11 / 2 = 5 \text{ R } 1$
$65 / 2 = 32 \text{ R } 1$	$5 / 2 = 2 \text{ R } 1$
$32 / 2 = 16 \text{ R } 0$	$2 / 2 = 1 \text{ R } 0$
$16 / 2 = 8 \text{ R } 0$	$1 / 2 = 0 \text{ R } 1$
$8 / 2 = 4 \text{ R } 0$	
$4 / 2 = 2 \text{ R } 0$	
$2 / 2 = 1 \text{ R } 0$	
$1 / 2 = 0 \text{ R } 1$	

wieder von unten lesen (Vorkommastellen!): 1000 0011 (bin), bzw. 1011(bin)

Nun gehen wir die einzelnen Stellen eines Faktors durch und definieren eine Variable für das Ergebnis:

$$\text{result} = 0;$$

Faktor2 hat vier Bitstellen, begonnen wird mit der linken Bitstelle: [1]011
Ich nehme den 2. Faktor, da hier nur 4 Durchläufe notwendig sind, wobei der größere Faktor1 8 Durchläufe verlangen würde:

$$\begin{aligned} \text{result} = \text{result} + [1] \cdot \text{Faktor 1} &\Rightarrow \text{result} = 0 + 1 \cdot 1000 \\ 0011 & \\ & \text{result} = 1000 0011 \end{aligned}$$

Es existiert noch eine weitere linke Stelle: 1[0]11, daher wird das vorläufige Ergebnis mal 2 (Basis) genommen. Multipliziert man mit der Basis, braucht man nur eine 0 hinten anzufügen (vgl. $4 \cdot 10 = 40$ im dezimalen System; ‚shift left‘):

$$\text{result} = \text{result} * 2; \quad \Rightarrow \quad \text{result} = 1 0000 0110$$

Entsprechend der Bitstelle wird aufaddiert. Im Struktogramm auf Seite 3-15 wird dieser Fall abgefangen und nichts gemacht. Dies ist notwendig, wenn man den Wert der Bitstelle nicht mit einbezieht.

$$\text{result} = \text{result} + [0] \cdot \text{Faktor 1} \quad \Rightarrow \quad \text{result} = 1 0000 0110 + 0;$$

Es existiert noch eine weitere linke Stelle: 10[1]1, daher wird das vorläufige Ergebnis mal 2 genommen.

$$\text{result} = \text{result} * 2; \quad \Rightarrow \quad \text{result} = 10 0000 1100$$

Entsprechend der Bitstelle (10[1]1) wird aufaddiert.

$$\begin{aligned} \text{result} = \text{result} + [1] \cdot \text{Faktor 1} &\Rightarrow \text{result} = 10 0000 1100 + \\ 1000 0011; & \\ &\Rightarrow \text{result} = 10 1000 1111 \end{aligned}$$

Es existiert noch eine weitere linke Stelle: 101[1], daher wird das vorläufige Ergebnis mal 2 genommen.

$$\text{result} = \text{result} * 2; \quad \Rightarrow \quad \text{result} = 101 0001 1110$$

Entsprechend der Bitstelle (101[1]) wird aufaddiert.

$$\begin{aligned} \text{result} = \text{result} + [1] \cdot \text{Faktor 1} &\Rightarrow \text{result} = 101 0001 1110 \\ &+ 1000 0011; \\ &\Rightarrow \text{result} = 101 1010 0001 \end{aligned}$$

Es existiert keine weitere linke Stelle: 1011[], das war's!

Das Ergebnis ist also : 101 1010 0001.

Probe: $1 \cdot 2^{10} + 0 \cdot 2^9 + 1 \cdot 2^8 + 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 =$
 $1024 + 0 + 256 + 128 + 0 + 32 + 0 + 0 + 0 + 0 + 0 + 1 =$
 1441

7. Lösen Sie die Divisionsaufgabe 1025/51 in binärer Darstellung auf 4 Nachkommabits genau! Verdeutlichen Sie Konvertierung \Rightarrow binär durch Angabe aller Zwischenergebnisse.

$$\begin{array}{r}
 1025 / 2 = 512 \text{ R } 1 \\
 512 / 2 = 256 \text{ R } 0 \\
 256 / 2 = 128 \text{ R } 0 \\
 128 / 2 = 64 \text{ R } 0 \\
 64 / 2 = 32 \text{ R } 0 \\
 32 / 2 = 16 \text{ R } 0 \\
 16 / 2 = 8 \text{ R } 0 \\
 8 / 2 = 4 \text{ R } 0 \\
 4 / 2 = 2 \text{ R } 0 \\
 2 / 2 = 1 \text{ R } 0 \\
 1 / 2 = 0 \text{ R } 1
 \end{array}
 \qquad
 \begin{array}{r}
 51 / 2 = 25 \text{ R } 1 \\
 25 / 2 = 12 \text{ R } 1 \\
 12 / 2 = 6 \text{ R } 0 \\
 6 / 2 = 3 \text{ R } 0 \\
 3 / 2 = 1 \text{ R } 1 \\
 1 / 2 = 0 \text{ R } 1
 \end{array}$$

$\Rightarrow 1025 \text{ (dez)} = 10\ 0000\ 0001 \text{ (bin)}$ und $51 \text{ (dez)} = 11\ 0011 \text{ (bin)}$

$$10\ 0000\ 0001.0000 / 11\ 0011 = 10100.00011$$

$$\begin{array}{r}
 \underline{-1\ 1001\ 1} \\
 0\ 0110\ 10 \\
 \underline{} \\
 110\ 100 \\
 \underline{110\ 011} \\
 10 \\
 \underline{} \\
 101. \\
 \underline{} \\
 101.0 \\
 \underline{} \\
 101.00 \\
 \underline{} \\
 101.000 \\
 \underline{} \\
 101.0000 \\
 \underline{11.0011} \\
 1.11010 \\
 \underline{1.10011} \\
 0.00111
 \end{array}$$

Das Ergebnis lautet also:

$$\begin{array}{l}
 10100.0001 \\
 \text{bzw. wenn es gerundet wird} \\
 10100.00011 \approx 10100.0010
 \end{array}$$

$$\begin{array}{l}
 \text{Probe: } 10100 = 20 + \\
 \phantom{\text{Probe: }} .00011 = 2^{-4} + 2^{-5} = 0,09375 \\
 1025 / 51 \approx 20,09804 \approx 20,09375
 \end{array}$$

8. Ermitteln Sie die Zweier-Komplement-Darstellung der Dezimalzahl -250 und verifizieren Sie das erhaltene binäre Ergebnis! Verdeutlichen Sie den Algorithmus der Konvertierung dezimal \Rightarrow binär durch Angabe aller Zwischenergebnisse.

Für das Zweier-Komplement wird zunächst die binäre Darstellung der Zahl 250 benötigt. Es werden 8 Bits benötigt, um die Zahl darzustellen. Für den negativen Raum wird allerdings noch ein Vorzeichen benötigt, es werden also indestens 9 Bit benötigt, im Beispiel wird mit 10 Bit gerechnet: 00 1111 1010m.

Nun muß die Zahl 00 1111 1010 (bin) nur noch Negiert werden. Dafür wird als erstes die Zahl mit einem exklusiven Oder binär ‚umgedreht‘: Aus jeder ‚0‘ wird eine ‚1‘ und aus jeder ‚1‘ wird eine ‚0‘: 11 0000 0101. Damit erhält man die Zahl -251 . Es muß also noch eins aufaddiert werden: 11 0000 0110 (bin) = -250 (dez) . (vgl. 3-18)

9. Erklären Sie den inneren Aufbau und die charakteristischen Eigenschaften einer **ShortRealZahl!**

Eine ShortReal-Zahl ist eine 32-Bit große Zahl, die in drei Teile unterteilt ist: Das linke Bit (Bit 31) kennzeichnet das Vorzeichen der Mantisse. Es folgen 8 Bit für den Exponenten, ebenfalls vorzeichenbehaftet, wobei 0111 1111 (bin) gleich Null ist. 1111 1111 (bin) gilt für NaN (Not A Number).

Im rechten Bereich befindet sich die Mantisse. In den Bits 0 – 22 wird der Wert der Zahl gespeichert. Zu Beachten ist, daß die erste ‚1‘ der Mantisse nicht gespeichert wird. Der Wert der Zahl kann unter Umständen nicht exakt gespeichert werden. Es kann passieren, daß Nachkommastellen abgeschnitten werden müssen.

(vgl. 3-26, vgl. Aufgabe 8)

10. Was verstehen Sie unter einer **BCD-Zahl?**

BCD steht für BinaryCodedDigit. Dabei gibt es zwei Möglichkeiten, wie die Zahl codiert sein kann.

Die ungepackte BCD-Ziffer entspricht dem ASCII-Code (48-57). Zusätzlich besteht die Möglichkeit die Zahlen in Nibbles zu packen, so daß 2 dezimale Zahlen in den beiden Nibbles eines Bytes gespeichert werden können.

(vgl. 3-28)

4. Grundlagen digitaler Schaltungen

1. Was ist unter einer **logischen Aussage** zu verstehen?

(keine eindeutige Quelle für ‚logische Aussage‘ gefunden)

Unter einer logischen Aussage versteht man eine Bedingung nach der Boole'schen Algebra.

Es existieren nur zwei Zustände: Wahr oder Falsch. Diese kann mit beliebig vielen Bedingungen verknüpft sein, das Ergebnis wird jedoch nur den Wert von Wahr oder Falsch annehmen.

Wichtigste Operatoren in der Boole'schen Algebra sind AND (C-Operator &&), OR (||), und NOT (!).

2. Erklären Sie das Wesen der logischen **Konjunktion!**

Die logische Konjunktion entspricht dem AND-Operator (&& in C).

Aus zwei angegebenen Wahrheitswerten (also ‚Wahr‘ oder ‚Falsch‘) wird ein Wert nach einer Wahrheitstabelle erstellt. Beim AND-Operator müssen beide angegebenen Werte ‚Wahr‘ sein, damit ein ‚Wahr‘ zurückgegeben wird. Sollte ein oder beide Werte ‚Falsch‘ sein, so wird das Ergebnis der AND-Verknüpfung ebenfalls ‚Falsch‘.

3. Erklären Sie das Wesen der **logischen Disjunktion!**

Die logische Disjunktion entspricht dem OR-Operator (|| in C).

Aus zwei angegebenen Wahrheitswerten wird ein Wert nach einer Wahrheitstabelle erstellt. Beim OR-Operator muß nur einer von beiden Werten oder beide Werte ‚Wahr‘ sein, damit die Verknüpfung ‚Wahr‘ wird. Sind beide Werte ‚Falsch‘, so wird das Gesamtergebnis auch falsch.

4. Welche logischen Grundaussagen erlauben das **Kürzen** logischer Ausdrücke?

Das Kürzen ist immer dann erlaubt, wenn Aussagen unabhängig einer Variablen immer ‚wahr‘ bzw. immer ‚falsch‘ werden. Dies wäre zum Beispiel dann der Fall, wenn eine Anweisung ($a \vee \underline{a}$) auftauchen würde. Sie wäre durch ein Grundsätzliches ‚wahr‘ zu kürzen. Im Gegensatz dazu ($a \wedge \underline{a}$), welches niemals ‚wahr‘ werden kann und daher durch ‚falsch‘ ersetzt werden kann.

5. Was beinhaltet das Inversionsgesetz? Geben Sie zur Verdeutlichung geeignete Logikschaltungen an!

Das Inversionsgesetz besagt, daß man einen Ausdruck komplett negieren kann, in dem entweder ein NOT vor den kompletten Ausdruck setzt oder alle Variablen einzeln negiert und alle logischen Konjunktionen durch Disjunktionen bzw.

Disjunktionen durch Konjunktionen vertauscht. (vgl. 4-10)

C-Notation : $!(a \ \&\& \ b)$ entspricht $!a \ || \ !b$, bzw.

$$\neg (a \wedge b) \text{ entspricht } (\neg a) \vee (\neg b)$$

6. Was ist unter einer **DKNF** zu verstehen und welche Grundschialtung logischer Elemente ist ihr äquivalent?

DKNF bedeutet Disjunktiv Kanonsche Normalform und bedeutet, daß alle Elemente mittels des OR-Operators verknüpft sind. (vgl. 4-7)

7. Was ist unter einer **KKNF** zu verstehen und welche Grundschialtung logischer Elemente ist ihr äquivalent?

KKNF bedeutet Konjunktiv Kanonsche Normalform und bedeutet, daß die Elemente mittels des AND-Operators verknüpft sind. (vgl. 4-7)

8. Vereinfachen Sie mittels Kürzungsregeln die Schaltfunktion

$$y = a \cdot b \cdot \underline{c} \vee a \cdot \underline{b} \cdot \underline{c} \vee \underline{a} \cdot b \cdot \underline{c} \vee \underline{a} \cdot \underline{b} \cdot \underline{c}$$

Geben Sie eine dem Ergebnis entsprechende **Schaltung** an!

\underline{c} läßt sich zunächst überall herausziehen:

$$\underline{c} \wedge (a \cdot b \vee a \cdot \underline{b} \vee \underline{a} \cdot b \vee \underline{a} \cdot \underline{b})$$

Schließlich läßt sich erkennen, daß alle vier Möglichkeiten, die $a \cdot b$ annehmen können auf jeden Fall einen ‚Wahr‘-Wert zurückliefert. Sie sind also nicht relevant für das Gesamtergebnis:

Die gekürzte Form lautet daher simpelst: \underline{c}

9. Vereinfachen Sie mit Hilfe des **KV-Diagramms** die Schaltfunktion
 $y = p \cdot q \cdot r \cdot \underline{s} \vee p \cdot q \cdot r \cdot s \vee p \cdot q \cdot \underline{r} \cdot \underline{s} \vee p \cdot q \cdot r \cdot s \vee p \cdot q \cdot \underline{r} \cdot s$
 Geben Sie eine dem Ergebnis entsprechende **Schaltung** an!

	p	p	\underline{p}	\underline{p}	
r	0	0	0	0	s
r	0	0	0	0	\underline{s}
\underline{r}	0	1	1	0	\underline{s}
\underline{r}	0	1	1	1	s
	q	q	q	\underline{q}	

$$y = q \underline{r} \vee p \cdot q \cdot r \cdot s$$

Bei $q \underline{r}$ spielt es keine Rolle, welcher Wahrheitsgehalt von r oder s in die Gleichung eingeht, das Ergebnis ist immer „wahr“.
 Es ist zusätzlich noch möglich aus dem zweiten Ausdruck das q zu entfernen, da die daraus zusätzlich entstehende Möglichkeit $\underline{p} \cdot q \cdot \underline{r} \cdot s$ bereits in $q \underline{r}$ enthalten ist: $y = q \underline{r} \vee p \cdot \underline{r} \cdot s$ (vgl. 4-14)

10. Wie funktioniert eine **Transistor-Negationsstufe**?

Hier definiere ich, daß dies nicht in der Klausur drankommt. ;)
 Wer anderer Meinung ist siehe Skript 4-22.

11. Was verstehen Sie unter einem **Gatter**?

Ein Gatter ist die einfachste Form, eine AND bzw. OR – Verknüpfung technisch umzusetzen. (vgl. 4-25)

12. Skizzieren Sie den Aufbau eines einfachen **Flip-Flop**-Elementes! Erläutern Sie anhand eines Signal-Diagramms aller möglichen Signalzustände seine Funktion und seine Anwendung.
 (vgl. 4-28)

13. Wie funktioniert ein **Halb-** bzw. **Volladder**?

(vgl. 4-29)

5. Architektur und Aufbau von Rechnern?

1. Ordnen Sie die wesentlichen Bestandteile eines Mikroprozessors den Elementen der **von Neumann**'schen Architektur zu!

Die von Neumann'sche Architektur unterscheidet in 4 wesentliche Bestandteile: Steuerwerk (Programm bzw. ProzessorROM), Arithmetik-Logik-Einheit („ALU“, Prozessorkern), Ein-/Ausgabe (Datenbus zu den Interface-Ports (Tastatur, Grafikkarte), bzw. Arbeitsspeicher) und Speicher (RAM-Speicher). (vgl. 5-5)

2. Welche wesentlichen Konzeptaspekte kennzeichnen heutige Rechner?

(keine konkrete Quelle im Skript gefunden)

Es wird keine so scharfe Trennung mehr zwischen den einzelnen von Neumann genannten Bauteilen gemacht. Vieles hat der Prozessor übernommen, neue Gebiete im Bereich Ein-/Ausgabe werden durch Interfaces erledigt (Soundkarte, Netzwerkkarte etc.)

3. Erklären Sie die Hauptbestandteile eines PC-Motherboard und ordnen Sie sie den Elementen der **von Neumann**'schen Architektur zu!

Hauptbestandteile sind CPU (Algorithmic-Logic-Unit – ALU), RAM (Arbeitsspeicher), Tastatur-, Maus-, Festplatten-, Floppy-, Seriell-, Parallelinterface, Netzwerkkarte, Soundkarte, Grafikkarte (Ein-/Ausgabe). Das Steuerwerk wird weiterhin vom Programm im Arbeitsspeicher, dem Menschen und ROM dargestellt.

4. Welche Grundelemente bilden den Kern des **Rechenwerks** eines Mikroprozessors? Beschreiben Sie die Funktion dieses Grundelementes. Skizzieren Sie die komplette Struktur eines solchen Rechenwerkes!

Keine brauchbare Quelle im Skript gefunden. (vgl. 5-10)

Erklären Sie das Schema eines **RAM-Chips** und erklären Sie seine Funktion!

Ein RAM-Chip ist so aufgebaut, daß die einzelnen Bit-Speicher in Spalten und Zeilen angeordnet sind. Über den Adreßbus wird dem Chip codiert die Adresse und die Zeile des abzufragenden Bits mitgeteilt. Mittels eines Spalten- bzw. Zeilendekoders, werden die entsprechenden Leitungen (jeweils nur eine senkrecht und eine waagrecht) aktiviert. Am Punkt, wo sie sich kreuzen ,schreibt' der Transistor seinen Wert auf den Datenbus, so daß er vom Prozessor gelesen werden kann. Zusätzlich existiert ein Chip-Select-Flag, welches angibt, ob dieser Chip auf die Anfrage reagieren soll oder nicht, so daß mittels eines zusätzlichen Hardware-Registers, der Speicherausbau verdoppelt werden kann, ohne die Adreßleitungen zu erweitern. (vgl. 5-14)

5. Erklären Sie die Zweckbestimmung und Funktion von **Interfacebausteinen**!

Interfacebausteine sind spezielle Chips, die eine Schnittstelle des Computers nach Außen darstellen. Der Prozessor hat keine direkte Verfügungsgewalt über sie. Sie können Daten über den Datenbus empfangen und absenden und sind mit dem Adreßbus mit dem Prozessor ansteuerbar, um so Daten in die Register des Interfacebausteins zu speichern, bzw. zu versenden etc. Der Interface-Baustein stellt üblicherweise auch ein Register zur Verfügung, mit dem der Status/Zustand der Schnittstelle abgefragt werden kann (z.B. Beschäftigt, es liegen Daten an etc...) Beispiele sind für Interfacebausteine sind die Control-Chips der seriellen, und parallelen Schnittstelle, aber auch Maus, Tastatur, Floppy etc. (vgl. 5-19f)

6. Welche Aufgaben und welche allgemeinen Eigenschaften hat das **Bussystem** eines Rechners?

Unter dem Bussystem versteht man eine bestimmte Anzahl von Drähten (Breite des Busses: 16 Drähte = 16 Bit) auf der Platine, über die Daten oder Adressen gesendet werden. Somit gibt es auf der Platine den Datenbus von den einzelnen Bausteinen (RAM, Interface etc.) zum Prozessor und den Adreßbus vom Prozessor zu den einzelnen Bausteinen, um anzugeben von welchem Chip bzw. welcher Speicheradresse zur Zeit etwas angefordert wird. Werden Daten vom Prozessor geschrieben, so werden diese ebenfalls über den Datenbus gesendet. Beispiele für andere Datenbussysteme sind SCSI-Bus, PCI-Bus etc. Weiterhin gibt es noch den Steuerbus innerhalb von Prozessoren/Chips, die die interne Funktion kontrollieren.

7. Wie ist eine **Computertastatur** aufgebaut und durch welches Konzept wird ihre flexible Verwendbarkeit für unterschiedliche Sprachen realisiert?

Eine Computertastatur ist ähnlich aufgebaut, wie ein RAM-Gatter. (vgl. 5-14) Wird eine Taste gedrückt, so sind ihr klare x- und y-Werte zuzuordnen. Dieser werden in einem sogenannten ,ScanCode' (RAW-Key)codiert und über das Interface der Tastatur an den Rechner übermittelt. Hier wird nun der mittels eines Treiberprogramms (Tastaturtreiber) dieser Scancode in ein beliebiges ASCII-Zeichen konvertiert. Beispiel: Die Taste rechts neben dem ,T' hat einen festen Scancode. Dieser ist unveränderlich. Mittels des deutschen Tastaturtreibers wird der Scancode nun in ein ,Z' gewandelt und das ,Z' an das Betriebssystem weitergeleitet. Installiert man einen amerikanischen Tastaturtreiber, so wird der gleiche Scancode in ein ,Y' gewandelt. (vgl. 5-26)

8. Welche Hauptbestandteile hat der **Bildschirmadapter** eines Rechners und was sind seine wesentlichen technischen Parameter?

Ich vermute, daß mit Bildschirmadapter die Grafikkarte gemeint ist.

Die Grafikkarte benötigt zunächst einen Anschluß an den Computer. Dies geschieht über den Bus (Adreß- und Datenbus sind üblicherweise an den Steckkarten zusammengelegt) des Computer-Systems. Auf der Grafikkarte befinden sich nun ein eigener Arbeitsspeicher (Video-RAM) und ein eigener Grafikprozessor, welcher die Informationen vom Bus entsprechend aufarbeitet (z.B. 3D-Funktionen in Bitmaps umrechnen und im Video-RAM ablegen). Damit der Grafikprozessor arbeiten kann, benötigt er Programme, die ihm sagen, was er zu tun hat, diese sind im ROM der Grafikkarte gespeichert. Zur Bildausgabe wird ein DAC (digital-analog-converter) benötigt, der das Bild Zeilenweise über das VGA-Kabel zum Monitor sendet.

9. Erklären Sie die Unterschiede zwischen Grafikmodus und Textmodus eines Rechners!

Beim Textmodus werden lediglich die ,ASCII'-Werte (ein Byte pro Zeichen) gespeichert. Die Bildausgabe muß diese dann mittels eines Zeichengenerators in Buchstaben-Grafiken umwandeln, die auf dem Bildschirm ausgegeben werden können. Im Grafikmodus bedarf es einer solchen Umwandlung nicht mehr. Der

Computer schreibt bereits alle Informationen als Grafik auf den Bildschirm. Dies erfordert wesentlich mehr Arbeitsspeicher und Rechenzeit.

10. Beschreiben Sie die Funktion von Druckern nach dem **Impact-** bzw. **Non-Impact-**Prinzip!

Impact- (Einschlags-) Drucker, drücken mittels eines Stempels ein Farbband auf das Papier und bilden so eine dem Stempel entsprechende Form auf dem Papier ab. Dieser Stempel ist bei Typenraddruckern oder Zeilendruckern in der Form eines Buchstabens, bei Nadeldruckern werden kleine Punkte abgebildet. Aus vielen Punkten lassen sich so Buchstaben konstruieren. Mit den Nadeldruckern ist es auch möglich, Linien und Grafiken zu drucken, während Typenrad und Zeilendrucker lediglich das drucken können, wofür sie Stempel besitzen (Buchstaben, Zahlen, wenige Sonderzeichen).

Non-Impact-Drucker berühren das Papier in der Regel nicht direkt. Tintenstrahldrucker spritzen feine Tintentröpfchen auf das Papier nach dem gleichen Prinzip wie die Nadeldrucker. Mit den Tintenstrahldruckern konnte erstmals brauchbar in Farbe gedruckt werden, da die Druckköpfe im Vergleich zum Nadeldrucker kleiner sind und so drei Druckköpfe mit den Druckgrundfarben (cyan, magenta und gelb, nicht rot, grün, blau!) zusammengefaßt werden konnten. Weiterhin sind Laserdrucker aufgeführt, welche lediglich komplette Seiten drucken können. (vgl. 5-31).

11. Wie funktioniert die **Datenspeicherung** mit einer **Festplatte** und wie ist sie **organisiert**?

Wodurch sind ihre wesentlichen technischen Parameter bestimmt?

Daten werden auf der Festplatte (und auch auf Floppies, jedoch nicht auf CDs) in Tracks, Sectors und Cylinders unterteilt. Dabei stellt ein Track eine komplette Umrundung der Festplatte an einem eindeutig definierten Abstand zum Mittelpunkt dar. Wird der Abstand verändert befindet man sich auf einem anderen Track. Wird eine Festplatte von beiden Seiten beschrieben bzw. besteht sie aus mehreren physischen Platten, dann nennt man die alle Spuren gleichen Abstands zum Mittelpunkt auf allen Seiten und Platten zusammengenommen Cylinder. Ein Sektor ist ein Ausschnitt, ein Teilbereich eines Tracks. In einem Sector werden in der Regel eine vom Hersteller festgelegte Menge von Bytes gespeichert. Sektoren können von der Platte nur vollständig gelesen und geschrieben. (Blockdevice) (vgl. 5-33)

6. Zahlendarstellung für Computer

1. Wie ist die **Rechnersoftware** strukturiert und welchen Zweck erfüllt die Strukturierung?

Bevor überhaupt ein Anwendungsprogramm gestartet werden kann, wird zunächst die Hardware mit einem Kernel umschlossen. Der Kernel beinhaltet die elementarsten Funktionen des Computers. Dazu gehört zum Beispiel das Lesen und Schreiben von und auf Schnittstellen, wie zum Beispiel Floppy, Festplatten, Serieller Anschluß oder Tastatur. So können zum Beispiel andere Programme geladen und gestartet werden. Um den Kernel schließt sich die Shell. Sie ermöglicht dem Anwender Zugriff auf die elementaren Funktionen des Computers mittels verständlicher Befehle (wie zum Beispiel ‚dir‘, oder ‚del‘ etc.). Auch normale Anwendungsprogramme sind in der Lage auf den Kernel des Rechners zuzugreifen, um zum Beispiel Texte von einer Textverarbeitung auf die Platte zu schreiben. Sinn der Sache ist, daß der Nutzer, der nur über die Shell, bzw. die Anwenderprogramme an den Kernel herankommt keine Gelegenheit hat in den internen Bereichen des Computers (Kernel) Änderungen vorzunehmen. Alles muß über standardisierte Schnittstellen laufen.

Der Kernel ist computerspezifisch. Die Funktionen und deren Aufruf sind jedoch standardisiert. So können neue Motherboards andere Hardware besitzen, die Benutzung der Hardware wird jedoch über den Kernel geregelt, dessen Aufrufe sich nicht verändert haben. (vgl.6-2)

2. Was gehört zum **Betriebssystem** eines Rechners und welche Aufgaben werden durch das Betriebssystem erfüllt?

Ein Betriebssystem ist ein abstrakte Plattform für einen Computer und steht zwischen Hardware und dem Benutzer. Es ermöglicht den einfachen Zugriff auf angeschlossene Geräte (seriell, Festplatte...) durch Gerätetreiber ohne das der Nutzer wissen muß, wie die Hardware der angeschlossenen Geräte funktionieren muß. Der Kern des Betriebssystems ist dauerhaft im Speicher vorhanden (resident), genauso wie alle dauerhaft benötigten Gerätetreiber. Oftmals bietet es zusätzliche Funktionen, um z.B. grafische Elemente aufzubauen (Fenster, Menüs etc.) und diese zu verwalten. (vgl. 6-4)

Das Betriebssystem unterteilt sich in 6 Bereiche: Es verwaltet die Dateien, den Speicher, die laufenden Prozesse (Programme), ggfs. ein Netzwerk, bei Mehr-

Benutzer-Systemen die einzelnen User und muß für die Sicherheit von Daten sorgen, auf die nicht alle User beliebig zugreifen können (darf lesen?; darf löschen?; darf ändern?) (vgl. 6-6)

3. Was verstehen Sie unter dem **Datenflußkonzept**?

Das Datenflußkonzept besagt, daß Daten von einer Quelle in eine Senke fließen. Das bedeutet, daß Daten erzeugt und vernichtet werden. Dies geschieht zum Beispiel in dem Daten mit der Tastatur erzeugt werden (Quelle, Input) und auf den Drucker geschrieben werden (Senke, Output). Massenspeicher wie Festplatten sind gleichzeitig Quelle als auch Senke. Daten werden geschrieben (Ziel, Senke) und gelesen (Quelle). (vgl. 6-7)

4. Welche Informationen über eine Datei sind für das **Betriebssystem** eines Rechners relevant (Ein- bzw. Mehr-Benutzer-System?)

Grundlegende Informationen sind Name und Größe der Datei. Bei blockorientierten Dateisystemen ist zudem der Block relevant, bei dem die Datei beginnt. Bei einem Mehrbenutzersystem ist zusätzlich wichtig zu wissen, wer Zugriffsrechte an dieser Datei hat. Somit muß gespeichert werden, wer der Erzeuger der Datei ist und wem bzw. welchen Gruppen er Zugriff auf diese Datei gibt.

Die meisten Betriebssysteme speichern zusätzlich den Zeitpunkt, an dem die Datei zuletzt verändert wurde, sowie andere zusätzliche Informationen zu der Datei. (vgl. 6-13)

5. Was ist eine **File Allocation Table** und wie funktioniert sie?

Die File Allocation Table (FAT) ist eine Tabelle, die für jeden Cluster (Datenblock) jeweils einen Wert besitzt. In dieser Tabelle wird gespeichert, ob ein Cluster frei ist, oder belegt. Die Cluster sind mit der Tabelle gleich angeordnet: die dritte Position in der Tabelle entspricht dem dritten Datencluster auf der Festplatte etc... Ist der Wert in der Tabelle Null, so ist der Cluster frei und es können dort Daten gespeichert werden; ist der Wert in der Tabelle ungleich Null, so ist der Cluster belegt. Die Zahl, die statt der Null in der FAT gespeichert wurde kann mehrere Bedeutungen haben. So gibt der Wert FFFF (hex) an, daß die Datei in dem der Position in der Tabelle entsprechenden Datencluster zuende ist und es keinen weiteren Cluster mehr gibt, in dem die Datei fortgesetzt wird. Für den Fall,

daß die Datei so groß ist, daß sie nicht in einem Cluster gespeichert werden kann, so ist in der Tabelle die Nummer des nächsten Clusters angegeben. Entsprechend dieser Nummer kann dieser Datencluster ausgelesen werden und in der Tabelle kann nachgeschlagen werden, ob dieser zweite Datencluster das Ende der Datei enthält. Falls ja, findet sich nun in der Tabelle der Wert FFFF, sonst steht dort die Nummer des nächsten Datencluster.

Die File Allocation Table wird immer zweifach auf die Platte geschrieben und befindet sich immer an einer festen Position auf der Festplatte.

6. Erläutern Sie die Verwaltung größer Dateien mit Hilfe der **I-Node**!

Die INode-Struktur ist die Verwaltungsstruktur im Minix-Filesystem. Es werden u.a. Name und Größe der Datei gespeichert. Zusätzlich werden sofern vorhanden die Verweise auf die ersten 7 Datenblöcke (Cluster) direkt in der I-Node-Struktur gespeichert. Für den Fall, daß die 7 Datenblöcke nicht ausreichen, existiert ein Verweis auf einen Datenblock ‚einfach indirekt‘. Dort existiert ein vollständiger Cluster, dessen Inhalt nicht zur Datei gehören, sondern lediglich Verweise auf weitere Datencluster enthält, deren Inhalt der Datei angehören. Somit lassen sich bei einer Datencluster-Größe von 1024Bytes in diesen einfach indirekten Cluster 512 Verweise (je 2 Bytes) unterbringen. Für den Fall, daß die Datei dennoch größer ist, als die 519 Verweise aus I-Node-Struktur und einfach indirekten Cluster existiert ein weiterer Verweis in der I-Node-Struktur: der zweifach indirekte Cluster. Analog zum einfach indirekten Cluster werden hier nur Verweise auf andere Cluster gespeichert, diesmal jedoch nicht auf Datencluster, die Teile der Datei enthalten, sondern auf einfach indirekte Cluster. Somit wird es möglich weitere 512 einfach indirekte Cluster zu verwalten, die ihrerseits jeweils auf 512 Datencluster zeigen. Insgesamt sind somit 7 (direkt) + 512 (einfach indirekt) + 512 * 512 (zweifach indirekt) Datenblöcke in einer Datei verwaltbar. (vgl. 6-16)

7. Erklären Sie den Begriff des Prozesses und den Zusammenhang, in dem er auftritt. Welche Prozeßzustände lassen sich unterscheiden?

Ein Prozeß ist ein ausführbares Programm, daß aktiv im Speicher arbeitet oder auf Eingaben wartet. Prozesse sind nur in Multitasking-Systemen möglich (Rechner ist in der Lage mehrere Programme gleichzeitig im Speicher zu halten), sie müssen sich die Ressourcen des Rechners (RAM, Rechenzeit, Schnittstellen etc.) teilen. (vgl. 6-25)

Auf einem Computer mit nur einem Prozessor können die Programme nicht alle gleichzeitig laufen. Ein ‚Scheduler‘ genannter Teil des Betriebssystems weist den Prozessen in preemptiven Multitasking-Systemen (vgl. 6-23) Rechenzeit zu und nimmt sie ihnen nach kurzer für den User meist unmerklicher Zeit wieder weg und weist die Aufmerksamkeit des Prozessor einem anderen Prozeß zu. In der Zwischenzeit ist der Prozeß deaktiviert. (vgl. 6-27)

8. Was verstehen Sie unter einem **Kommando-Interpreter**? Und wie funktioniert er?

Ein Kommandointerpreter (Unix-Shell, MS-DOS, CLI) ist ein Programm bzw. Teil des Betriebssystems welches Texteingaben des Nutzers ‚interpretiert‘ und ausführt. Die eingegebene Kommandozeile wird analysiert, üblicherweise ist das erste Wort der Name des Kommandos, die dahinter stehenden Daten sind Anweisungen an das Kommando. Das Kommando ist üblicherweise ein kleines Programm, welches die Anweisungen übergeben bekommt und entsprechend abarbeitet. Wurde kein Programm gefunden, daß den Namen des Kommandos trägt, so wird eine Fehlermeldung ausgegeben. (vgl. 6-28ff)

9. Nennen Sie die wichtigsten Fähigkeiten einer modernen Textverarbeitung!

Wysiwyg etc. (vgl. 6-33)

10. Wie arbeitet ein **Tabellenkalkulationsprogramm**?

Tabellen sind Datenblätter, die in Zeilen und Spalten aufgegliedert sind. In jedem Feld kann ein Text, eine Zahl oder Formel enthalten sein. Mit den Formeln können Werte aus anderen Feldern gelesen und in Rechnungen mit einbezogen werden können. Beispiel:

	A	B	C	D
1	10	20	30	=SUMME(A1, B2)
2	30	15	12	=B1

Die Tabellenkalkulation rechnet nun im Feld D1 die Summe aus und schreibt statt ‚=SUMME(A1, B2)‘ den Wert 25 in das Feld D1. Wird nun der Wert A1 oder B2 verändert, ändert sich das Feld D1 entsprechend mit. Im Feld D2 wird der Wert vom Feld B1 einfach kopiert. Es erscheint ‚20‘ im Feld D2. Wird der Wert im Feld B1 verändert, ändert sich auch der Wert im Feld D2 entsprechend.

11. Welche Hauptaktionen müssen für die Arbeit mit einer **Datenbank** realisiert werden?

Die Datenbank muß installiert werden... Mir ist nicht ganz bewußt, wonach hier gefragt wird.

(vgl. 6-38)

Bemerkenswerte Dinge bei einer Datenbank sind meines Erachtens, daß Daten möglichst nur einmal (Primärschlüssel) gespeichert werden und alles weitere mittels Verweisen auf diesen Primärschlüssel (genannt Sekundärschlüssel) erledigt wird. So entstehen Beziehungen zwischen z.B. bestellten Artikeln und dem Käufer. Ein Beispiel:

Die Datenbank hat alle Artikel gespeichert und Ihnen Artikelnummern zugewiesen. Ebenfalls kennt sie die Daten der Kunden und hat Kundennummern vergeben. Ein Auftrag läßt sich nun so realisieren, daß nur die Kundennummer und Anzahl der jeweiligen Artikelnummern gespeichert werden müssen. Ändert sich zum Beispiel die Adresse des Kunden, so wird diese in der Kundendatei geändert und braucht in allen noch laufenden Aufträgen nicht mehr geändert werden.

12. Wie funktioniert **elektronische Post** weltweit?

Elektronische Post funktioniert ähnlich der normalen Sackpost.

Die E-Mailadresse ist entsprechend von hinten nach vorne zu lesen:

Xin@check.bdm.de : de – User ist im deutschen Netz, die Mail wird weitergeleitet an Subnetz bdm.
 : bdm – Subnetz ist bdm, wird weitergeleitet an ‚check‘-Server. Der Server des bdm-Netzes ist ein Gateway zwischen de und bdm-Netz.
 : check – Vor ‚check‘ steht das @-Zeichen.
 User ist in System ‚check‘ angemeldet.
 (Checkpoint BBS Erfstadt)

Wenn der Checkpoint-Server einen User ‚Xin‘ kennt, so wird die Mail in das Postfach gelegt, andernfalls wird eine Fehlermeldung an den Absender verschickt. (vgl. 6-41)

7. Algorithmen

1. Erklären Sie die Begriffe **Schlüssel** und **Container** (im Zusammenhang mit Such- oder Sortieralgorithmen)!

Schlüssel: Als Schlüssel wird ein Datenobjekt bezeichnet, welches gesucht wird, der Schlüssel paßt nur, wenn das gesucht Datenobjekt gefunden ist, bei keinem anderen.

Container: Als Container wird der Bereich von Datenobjekten genannt, in dem ein Schlüssel zu suchen ist.

2. Unter welcher Voraussetzung kann in einem Datenbestand binär gesucht werden? Vergleichen Sie die Arbeitsweise der binären Suche mit der linearen Suche und entwickeln Sie jeweils einen **Programmablaufplan** der wesentlichen Schritte. Was läßt sich über den **Zeitaufwand** beider Suchverfahren sagen?

- a) *Es kann binär gesucht werden, wenn die Daten sortiert vorliegen. (vgl. 7-4)*
- b) *Das binäre Suchen beginnt in der Mitte und halbiert den Suchbereich (Container) laufend, bis das gesuchte Element gefunden ist, bzw. der Anfang des Suchbereiches gleich dem Ende des Suchbereiches ist. Das lineare Suchen beginnt auf einer Seite des Suchbereiches und sucht linear ein Element nach dem anderen ab.*
- c) *(vgl. 7-5)*
- d) *Bei x Elementen braucht das lineare Suchen benötigt im ungünstigen Fall genauso viele Vergleiche, wie Elemente vorhanden sind, wohingegen das binäre Suchen im ungünstigsten Fall nur Wurzel aus x Vergleiche benötigt. Damit ist das binäre Suchen wesentlich effektiver. (vgl. 7-2; 7-4)*

3. Beschreiben Sie den Algorithmus des **SelectionSort**-Verfahrens!

Beim Selection-Sort-Algorithmus wird zunächst das kleinste Element im zu sortierenden Bereich herausgesucht. Sobald es gefunden ist, wird es mit dem ersten Element vertauscht und der zu sortierende Bereich wird so verkleinert, daß das erste Element herausfällt, da es ja bereits an der richtigen Position steht. (vgl. 7-9)

4. Beschreiben Sie den Algorithmus des **InsertionSort**-Verfahrens!

Der InsertionSort-Algorithmus sortiert die Elemente, indem er die Elemente der Reihe nach aus dem unsortierten Bereich herausnimmt und solange weiter nach rechts in den sortierten Bereich verschiebt, bis es an der richtigen Position gelandet ist. Dann wird das nächste Element aus dem unsortiertem Bereich in den sortierten Bereich einsortiert, bis der unsortierte Bereich die Größe Null erhält. Damit sind alle Elemente sortiert worden. (vgl. 7-12)

5. Beschreiben Sie den Algorithmus des **BubbleSort**-Verfahrens!

Beim BubbleSort-Verfahren werden die benachbarte Datensätze miteinander verglichen und sofern sie in der falschen Reihenfolge zueinander stehen, werden sie vertauscht. Hat der Algorithmus das Ende des zu sortierenden Bereichs erreicht, wird wenn eine Vertauschung vorgenommen wurde der Algorithmus erneut gestartet. Mußte keine Vertauschung mehr unternommen werden, so sind alle Elemente nach ihrer Größe sortiert.

6. Beschreiben Sie den Algorithmus des **QuickSort**-Verfahrens!

Hier sei auf Robert Sedgewicks Buch Algorithmen verwiesen, was in der Bibliothek und in jedem gut sortierten Informatikerbuchregal steht.

7. Geben Sie eine Abschätzung der Arbeitsgeschwindigkeit der obigen Sortierverfahren für die günstigste, die durchschnittliche und die ungünstigste Verteilung von N Datensätzen an und begründen Sie diese Aussagen!

	<i>SelectionSort</i>	<i>InsertionSort</i>	<i>BubbleSort</i>	<i>QuickSort</i>
<i>Günstigster Fall</i>	N (Überprüfung, daß alle Elemente korrekt sortiert sind)	N (Überprüfung, daß alle Elemente korrekt sortiert sind)	N (Überprüfung, daß alle Elemente korrekt sortiert sind)	N (Überprüfung, daß alle Elemente korrekt sortiert sind)
<i>Durchschnittlich</i>	$N*N/2$ (das gesuchte Element findet sich durchschnittlich nach der Hälfte der Gesamtelemente, es muß N mal gesucht werden)	$N*N/4$ N Elemente müssen überprüft und neu sortiert werden. Sortierung ist durchschnittlich nach $N/2$ Elementen beendet. Keine Ahnung, wo die restlichen $1/2$ herkommen :($0,375 N*N$ Es müssen N Elemente sortiert werden * durchschnittliche Entfernung zum Ziel ($0,375$) * Durchläufe des Algorithmus	(keine Angabe im Skript)
<i>Ungünstigster Fall</i>	$N*N$ Das erste Element steht ganz rechts, bzw. das letzte ganz links.	$N*N/2$ Die Sortierung braucht statt der durchschnittlichen n Suche nach dem kleinsten Element nicht $N/2$ Versuche, sondern muß über alle Elemente (N) suchen.	$N*N/2$ Das erste Element steht ganz rechts, bzw. das letzte ganz links.	$N*N/2$

8. Wie funktioniert der Vergleich von Zeichenketten hinsichtlich ihrer alphabetischen Einordnung?

Sofern ich die Frage richtig verstanden habe:

Bei Texten wird mittels der ASCII-Codes sortiert. Das ‚A‘ besitzt zum Beispiel den Wert 65, ‚B‘ ist gleich 66 etc... Somit sind Größenvergleiche zwischen den Buchstaben möglich.

Bei Zeichenketten ist wird zunächst der erste Buchstabe der ersten Zeichenkette mit dem ersten Buchstaben der zweiten verglichen und die Zeichenkette entsprechend eingeordnet. Sind die Buchstaben gleich, wird die Überprüfung am nächsten Buchstaben fortgesetzt, bis sich die Zeichenketten um einen Buchstaben unterscheiden. (keine Angabe im Skript Kapitel 7 gefunden)

8. Datenstrukturen

1. Was verstehen Sie unter einem **Stack**? Erläutern Sie die für seine Benutzung geeigneten **Grundalgorithmen**! Was ist seine wichtigste **Anwendung**?

Jedes Programm benutzt einen eigenen Stack. Der Stack ist ein Bereich im Arbeitsspeicher, ein Zwischenspeicher für Programme. Auf den Stack können Daten abgelegt an einem Ende abgelegt werden und es dürfen auch nur die oben aufliegenden Daten wieder entfernt werden (Last-In-First-Out). Jedes Funktion in einem Programm, welche Daten auf den Stack ablegt, muß diese vor verlassen der Funktion wieder entfernen, um den Stack nicht aus dem Gleichgewicht zu bringen.

Grundalgorithmen sind vorrangig Ablegen von Daten auf den Stack (,Push') und das Lesen (und freigeben vom Stack) von Daten (,Pop').

Normalerweise werden auf dem Stack nur Adressen oder Werte gespeichert.

Dabei funktioniert das Schreiben so, daß die es einen Zeiger(Stackpointer) gibt, der auf die erste freie Position im Stack-Speicherbereich zeigt. Zunächst wird der Stackpointer entsprechend der Größe der Daten (bei einer 16-Bit Adresse entsprechend 2 Bytes) und die Daten werden auf der Adresse, auf die der Stackpointer nun zeigt, abgelegt. Beim Lesen wird umgekehrt vorgegangen: Die Daten werden aus dem Stack zurückkopiert und schließlich wird der Stackpointer entsprechend der Größe der Daten wieder zurückgeschoben, so daß die nächsten Daten, die auf den Stack geschrieben werden, die grade kopierten Daten überschreiben werden.

Zu beachten ist, daß der Stack üblicherweise rückwärts aufgebaut wird (daher auch die Reihenfolge von Stackpointer verschieben/Schreiben bzw.

Lesen/Stackpointer verschieben.) Das bedeutet, daß Ende vom Stack im Speicher vor dem Anfang angeordnet ist.

Wichtigste Anwendung des Stacks ist die Kontrolle von internen Programmabläufen. Wird in ein Unterprogramm verzweigt, so wird im Stack die Adresse gespeichert, von der das Unterprogramm aufgerufen wird. Wenn das Unterprogramm fertig ist, liest es die Adresse aus dem Stack aus und springt dorthin zurück. (vgl. Gleichgewicht des Stacks).

(vgl. 8-1ff)

2. Was verstehen Sie unter einer **Queue**? Erläutern Sie die für ihre Benutzung geeigneten **Grundalgorithmen**! Was ist die wichtigste **Anwendung**?

Ein Queue (Ringspeicher auf Deutsch(Warteschlange)) ist ein Speicherbereich, welcher jeweils ein Zeiger für Eingabe und Ausgabe besitzt. Daten werden entsprechend an den Eingabezeiger geschrieben. Wird das Ende erreicht, wird vorne weitergeschrieben. Erreicht man den Outputzeiger, sollen mehr Daten im Queue gespeichert werden, als Platz ist. Werden Daten gelesen, beginnt man am Outputzeiger (Output aus Sicht des Ringspeichers) und verschiebt den Outputzeiger entsprechend der Datenmenge. Daten werden im Gegensatz zum Stack in der Reihenfolge gelesen, wie sie auch hineingeschrieben werden. Grundalgorithmen: Schreiben: Daten werden geschrieben, der Inputzeiger entsprechend der Datenmenge (eine Adresse im 16-Bit-Systemen = 2 Byte) verschoben. Wird das Rand des Ringspeichers (Queue) erreicht, so wird der Inputzeiger auf den Anfang zurückgesetzt. Erreicht der Inputzeiger den Outputzeiger, so ist der Ringspeicher voll.

Lesen: Daten werden gelesen, der Outputzeiger entsprechend der Datenmenge verschoben. Wird der Rand des Ringspeichers erreicht, so wird der Outputzeiger auf den Anfang zurückgesetzt. Erreicht der Outputzeiger den Inputzeiger, so ist der Ringspeicher leer.

Hauptaufgabe ist das Puffern von Daten (Cache). So werden zum Beispiel die Daten, die an der seriellen Schnittstelle ankommen in einen Ringspeicher geladen und dem Betriebssystem gemeldet, daß etwas angekommen ist. Bis das Betriebssystem reagiert verbleiben die Daten im Ringspeicher, neue Daten werden vom Interfacebaustein entsprechend dem Inputzeiger angefügt.(vgl 8-6ff)

3. Erläutern Sie das Konzept einer **verketteten Liste**! Begründen Sie deren Vorteil gegenüber einem Array von Datensätzen! Wie funktioniert der Algorithmus zur Anzeige einer solchen Liste?

Zum Konzept einer verketteten Liste: Eine verkettete Liste ist eine Datenblock, der neben den eigentlichen Daten einen Zeiger auf den nachfolgenden Datenblock besitzt. Die Reihenfolge der Elemente der Liste ist nicht von der Position im Speicher abhängig, sie wird dadurch geklärt, daß jedes Datenelement auf das nachfolgende Element zeigt (bzw. bei einer doppelt verketteten Liste auf Nachfolger und Vorgänger).

(Aus dem Skript geht nicht hervor, was ein Array ist)

Ein Array ist eine Hintereinanderreihung von Daten gleichen Typs. So ist ein Wort zum Beispiel eine Hintereinanderreihung von Buchstaben. Vorteil der verketteten Liste ist, daß beim Zufügen oder Entfernen von Daten aus dem Datenbestand, die Daten nicht kopiert werden müssen, um die notwendige Lücke für die neuen Daten zu schaffen, bzw. die durch das Löschen entstandene Lücke zu schließen. Es wird lediglich der Zeiger auf das Nachfolgende Element korrigiert.

Die Ausgabe einer Liste funktioniert nach folgenden Prinzip: Es wird eine Laufvariable benötigt, die zunächst auf erste Element der Liste zeigt. Die Daten werden ausgegeben und die Laufvariable auf den Nachfolger gesetzt. Dies wird sooft wiederholt, bis es kein Nachfolgeelement mehr gibt. Dies ist dadurch gekennzeichnet, daß das letzte Element als Zeiger auf den Nachfolger keine Adresse, sondern den Wert Null besitzt (NIL in Pascal/Delphi). (vgl. 8-10)

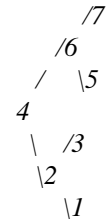
4. Erklären Sie den Aufbau und die wesentlichen Eigenschaften eines **binären Baumes!**

Ein binärer Baum ist gewissermaßen eine Liste mit jeweils zwei Nachfolgern. Somit läßt sich die Suchgeschwindigkeit beschleunigen, da die Nachfolger im Baum einfach sortiert werden können: ein Nachfolger für Elemente, die kleiner oder gleich dem eigenen Datenelement sind und der zweite Nachfolger für Elemente, welche größer als das eigene Datenelement sind. Bei einem ausbalancierten Baum (d.h. daß der Baum so aufgebaut wird, daß möglichst wenig Verzweigungen notwendig sind, um ein Element zu finden:

Nicht balancierter Baum: (um an das Element 7 heranzukommen muß sechsmal von der Wurzel (1) aus verzweigt werden.



Balancierter Baum: (maximale Anzahl der Verzweigungen bei einem balancierten Baum mit 7 Elementen ist 2: Das Element 7 ist bereits nach zwei Verzweigungen gefunden).



(vgl. 8-4)

5. Worin besteht das Prinzip **rekursiver Algorithmen**? Welche Vorteile und welche Probleme sind damit verbunden?

(keine Quelle gefunden)

Rekursive Algorithmen sind Unterprogramme, die so geschrieben wurden, daß sie sich selbst aufrufen können. Dies vereinfacht Probleme oftmals sehr, die laufend wieder auftreten. Das Teilproblem wird in einer Prozedur beschrieben und diese ruft sich sooft wie notwendig selbst auf, damit ist das Problem oft übersichtlicher und einfacher zu lösen als durch Schleifen-Konstrukte.

Probleme können durch mangelnden Stackspeicher auftreten, wenn die Verschachtelung der Prozeduren so tief geht, daß die Daten, die bei jedem Aufruf der Prozedur auf den Stack geschrieben werden, über die Grenzen des Stacks hinausgehen. Weiterhin sind rekursive Prozeduren teilweise fehleranfällig und kompliziert zu korrigieren.

Respekt!

Wer diesen Text solange ertragen konnte, kann davon ausgehen, daß er das schwierigste am Studium schon hinter sich hat: Mich :->