Vorlesungsmodul Programmieren 2
- VorlMod Prog2 -

Matthias Ansorg

14. März 2002 bis 23. Mai 2003

Zusammenfassung

Studentische Mitschrift zur Vorlesung Programmieren 2 bei Prof. Hoffmann (Sommersemester 2002) im Studiengang Informatik an der Fachhochschule Gießen-Friedberg.

- **Bezugsquelle:** Die vorliegende studentische Mitschrift steht im Internet zum Download bereit: http://matthias.ansorgs.de/InformatikDiplom/Modul.Prog2.Hoffmann/Prog2.pdf.
- **Lizenz:** Diese studentische Mitschrift ist public domain, darf also ohne Einschränkungen oder Quellenangabe für jeden beliebigen Zweck benutzt werden, kommerziell und nichtkommerziell; jedoch enthält sie keinerlei Garantien für Richtigkeit oder Eignung oder sonst irgendetwas, weder explizit noch implizit. Das Risiko der Nutzung dieser studentischen Mitschrift liegt allein beim Nutzer selbst. Einschränkend sind außerdem die Urheberrechte der verwendeten Quellen zu beachten.
- **Korrekturen:** Fehler zur Verbesserung in zukünftigen Versionen, sonstige Verbesserungsvorschläge und Wünsche bitte dem Autor per e-mail mitteilen: Matthias Ansorg, ansis@gmx.de.
- **Format:** Die vorliegende studentische Mitschrift wurde mit dem Programm LyX (graphisches Frontend zu LaTeX) unter Linux erstellt und als pdf-Datei exportiert. Grafiken wurden mit dem Programm xfig unter Linux erstellt und als eps-Datei exportiert.
- **Dozent:** Prof. Hoffmann.
- **Verwendete Quellen:** .
- **Klausur:**

Inhaltsverzeichnis

1 ToDo

Mindestens die Kapitel 8 und folgende (d.i. S. 206 ff.) in [5] waren nicht Stoff der Veranstaltung Programmieren II bei Prof.Hoffmann.

2 Schreibtischtests

Die folgenden Tipps dienen der sicheren Durchführung von Schreibtischtests bei unübersichtlichen und komplizierten Ausdrücken.

Um Ausdrücke auszuwerten, verwende man grundsätzlich »Äquivalenzen« in quasimathematischer Notation. Das heißt, man formt die Ausdrücke schrittweise in immer einfachere Ausdrücke um. Dabei ist jeder Variablenbezeichner äquivalent zu seinem Inhalt. Dies ist vor allem bei Ausdrücken mit Pointern hilfreich, z.B.:

Gegeben sind folgende Initialisierungen:

```
int x[] = {1,3,5,7,8,6,4,2,0};
int *p = &x[3];
int **q = &p;
int i = -3, j = 2;
```

Dann gelten folgende Äquivalenzen bei der Auswertung von Ausdrücken:

```
q ⇔ &p
*q⇔ *(&p) ⇔ p
**q ⇔ **(&p) ⇔ *p ⇔ *(&x[3]) ⇔ x[3] ⇔ 7
```

Es ist empfehlenswert, bei der Auswertung eines Ausdrucks zuerst in einen solchen äquivalenten Ausdruck umzuformen, in dem man alle Prioritäten von Operatoren durch eine entsprechende Klammerung ausgedrückt hat. Um das zu erreichen, beginnt man mit den enthaltenen Operatoren mit der höchsten Priorität und fasst sie und all ihre Argumente zu einem Ausdruck zusammen; so fährt man fort bis zu den Operatoren mit der niedrigsten Priorität, deren Argumente jetzt auch Klammerausdrücke sein können.

Treffen zwei Operatoren mit der gleichen Priorität zusammen, entscheidet die Assoziativität: Fast alle Operatoren sind linksbindend (man rechnet normalerweise »linear, von links nach rechts«), nur einstellige Operatoren und Zuweisungsoperatoren sind rechtsbindend (man rechnet »rekursiv, von rechts nach links«). Ein Ausdruck »`a-b-c-d`« ist äquivalent zu »`((a-b)-c)-d`« und nicht zu »`a-(b-(c-d))`«, denn der Subtraktionsoperator ist linksassoziativ, d.h. die Elemente am weitesten links werden zuerst verknüpft. Der Ausdruck »`a=b=c`« ist jedoch äquivalent zu »`a=(b=c)`«, denn der Zuweisungsoperator ist rechtsassoziativ.

Für Ausdrücke mit dem ternären Operator: ein »`:`« gehört immer zum unmittelbar vorhergehenden »`?`«. Das heißt: der ternäre Operator ist rechtsassoziativ. Beginnt man beim ersten ternären Operator, so kann dessen then-Fall und else-Fall wiederum einen solchen ternären Operator enthalten, d.h. es ergibt sich eine Schachtelung.

3 Deklaration von Objekten aus Variablen, Funktionen, Feldern und Zeigern auf all diese.

Siehe hierzu aus die Datei `Test.Dekl.cc`.

Es gibt folgende, durch Semikola getrennte, Grundelemente von Deklarationen:

`<typ> name` Einfachen Variablen.

`<typ> name (<liste>)` Funktion mit Rückgabewert und Argumenten.

`<typ> (*name) (<liste>)` Zeiger auf eine Funktion mit Rückgabewert und Argumenten.

`<typ> (*name[x]) (<liste>)` Feld von Zeigern auf Funktionen mit Rückgabewert und Argumenten.

`<typ> (*)(<liste>) name[x]` Wie oben. Wird von g++ nicht akzeptiert, daher wohl fehlerhaft, im Gegensatz zur Aufführung in [5].

Es gibt folgende, durch Kommata getrennte, Grundelemente von Argumenten:

`<typ> name` Wertparameter von einem Typ. `name` kann auch fehlen.

`<typ> name[x]` Feld als Wertparameter mit Elementen von einem Typ. `name` kann auch fehlen.

`<typ> &name` Referenzparameter von einem Typ. `name` kann auch fehlen.

`<typ> (&name)[x]` Feld als Referenzparameter mit Elementen von einem Typ. `name` kann auch fehlen.

`<typ> &name[x]` Feld von Referenzen auf Variablen von einem Typ. Das Feld ist ein Wertparameter! `name` kann auch fehlen.

`<typ> *name` Zeiger auf eine Variable von einem Typ als Wertparameter. `name` kann auch fehlen.

`<typ> (*name)[x]` Zeiger auf ein Feld von Elementen eines Typs. `name` kann auch fehlen.

`<typ> *name[x]` Feld von Zeigern auf eine Variable von einem Typ. `name` kann auch fehlen.

Es gibt folgende Grundelemente von Rückgabewerten:

`<typ> f()` Wert als Ergebnis.

`<typ> & f()` Referenz als Ergebnis.

`<typ> *f()` Zeiger auf einen Typ als Ergebnis.

`<typ> **f()` Zeiger auf einen Zeiger auf einen Typ als Ergebnis.

`<typ> f()[x]` Feld mit Elementen von einem Typ als Ergebnis.

`<typ> *f()[x]` Feld von Zeigern auf Variablen eines Typ als Ergebnis.

`<typ> (*f())[x]` Zeiger auf ein Feld von Variablen eines Typs als Ergebnis. Im Gegensatz zu »Zeiger auf eine Funktion« ist hier der Funktionsbezeichner mit Argumenten eingeklammert!

4 Lösungen zu Übungsaufgaben

Die folgenden Lösungen beziehen sich auf [4], wobei die Nummerierung der Aufgaben identisch übernommen wurde.

4.1 Laufende Nummer eines Tages

4.2 Ackermann-Funktion

Siehe beigefügte Datei `Aufg.2.cc`.

4.3 Verschlüsselungsalgorithmus

Die Maske `MSK` hat den Wert `0x55555555`, dem entspricht das Bitmuster `0101`, achtmal wiederholt. Der Ausdruck `x>>1&MSK|x<<1&~MSK` entspricht nach der Operatorenrangfolge dem Ausdruck:

```
((x>>1)&MSK)|((x<<1)&(~MSK))
```

Damit ist der Algorithmus: Vertausche die Bits der Binärdarstellung des Integers in jedem Paar. Da Integerzahlen immer eine gerade Anzahl von Bits haben, ist es für das Ergebnis egal, ob bei dem führenden oder letzten Paar Bits mit Vertauschen begonnen wird.

Das paarweise Vertauschen wird hier wie folgt erreicht (die Bits werden nummeriert, beginnend mit dem führenden Bit, das Nummer 1 erhält und damit ein »ungerades Bit« ist)

1. `x>>1`: Verschiebe alle Bits von `x` um 1 nach rechts. Das letzte (gerade) Bit verschwindet.
2. `(x>>1)&MSK`: Setze alle geraden Bits von `x` (die ja jetzt auf ungeraden Stellen stehen) auf 0. Teilergebnis 1.
3. `x<<1`: Verschiebe alle Bits von `x` um 1 nach links. Das führende (ungerade) Bit verschwindet.
4. `&(~MSK)`: Setze alle ungeraden Bits von `x` (die ja jetzt auf geraden Stellen stehen) auf 0. Teilergebnis 2.
5. Kombiniere die beiden Teilergebnisse. Nun stehen alle ungeraden Bits eine Stelle weiter rechts, alle geraden Bits eine Stelle weiter links. Das entspricht einer Vertauschung der Bits in jedem Paar.

4.4 Ägyptische Bauernmultiplikation

»Warum kann man behaupten, daß es sich bei der ägyptischen Bauernmultiplikation um ein Rechenverfahren nach dem neuesten Stand der Technik handelt?« Vermutung: Dieses Verfahren wird zur Ganzzahlmultiplikation in heutigen Prozessoren eingesetzt, denn es eignet sich hervorragend für binäre Zahldarstellungen: Verdopplung ist gleich dem Verschieben der Bitfolge um eine Position nach links, ganzzahliges Halbieren ist gleich dem Verschieben der Bitfolge um eine Position nach rechts, und am letzten Bit kann man feststellen, ob eine Zahl gerade oder ungerade ist. Additionen können schnell ausgeführt werden.

4.5 Matrixverarbeitung

4.6 Effekt der Funktion `fkt`

Es handelt sich um eine Implementierung des straightinsert-Sortieralgorithmus, der die Elemente `0` bis `n-1` des Feldes `x[]` zu einer absteigenden Folge sortiert.

```
void fkt(int x[], int n) {
  for (int i=1; i<n; i++)
  //INV: Einfuegeschritt fuer x[i] in absteigend sortierte
  //Folge x[0] .. x[i-1]
    if (x[i]>x[i-1]) {
      int v=x[i]; //Zwischenspeicher, x[i] jetzt belegbar
      int j=i;
      //alle Elemente >v eins nach rechts schieben:
      while (j>0 && v>x[j-1]) {
        //INV: x[j] ist belegbar, da sein Inhalt an
        //andere Stelle kopiert wurde
        x[j]=x[j-1]; j--;
      }
      x[j]=v;
    }
}
```

4.7 Magisches Quadrat

Siehe beigefügte Datei `Aufg.7.cc`.

4.8 Ausgabe des Programms

Die Ausgabe kann mit dem Sourcecode in der begefügten Datei `Aufg.8.cc` leicht selbst erstellt werden. Sie ist:

```
20
8
6
18
10
8
```

4.9 Auswirkung der Funktionen `xyz_a`, `xyz_b`

a) Die Funktion erhält mit `float *v` einen Zeiger auf ein Array von `float` und mit `n` die Anzahl der Elemente dieses Arrays. Die Funktion bewirkt die Vertauschung des ersten und letzten, des zweiten und vorletzten usw. Elements im Array. Da nach einem Schleifendurchlauf beide Indizes verändert werden, können sie nur für das mittlere Element bei ungeradem `n` gleich werden, bei geradem `n` nie. Sind `i` und `j` gleich, so muss keine Vertauschung durchgeführt werden, denn die Vertauschung eines Elements mit sich selbst ist überflüssig.

b) Die Funktion hat dieselbe Auswirkung wie `xyz_a`, zusätzlich aber das Abschneiden des Gleitkommaanteils der `float`-Werte durch Konversion von `float` nach `int` (Anweisungen `vj=*(v+j), vi=*(v+i);`) und wieder zurück nach `float` (Anweisungen `*(v+i)=vj; *(v+j)=vi;`).

c) Durch die Schleifenbedingung `i<=j` wird die Schleife auch dann durchlaufen, wenn die Werte für `i` und `j` für das mittlere Element bei ungeradem `n` identisch sind. In `xyz_a` war das unnötig, weil die Vertauschung eines Elementes mit sich selbst unnötig ist; in `xyz_b` ist das nötig, weil zusätzlich der Gleitkommateil der `float`-Werte abgeschnitten wird.

4.10 Funktion `int ue_to_umlaut(string &s)`

4.11 Werte von Ausdrücken

Hier wird eine quasimathematische Notation für die Werte der Ausdrücke verwendet, wobei $\Leftrightarrow$ jeweils »Gleichwertigkeit« bedeutet. Die Ergebnisse können mit dem Sourcecode in der begefügten Datei `Aufg.11.cc` leicht selbst erstellt werden. Sie sind:

a) 5

b) 4

c) 5

d) $*(z-k) \Leftrightarrow *(\&x\,[3]) \Leftrightarrow 2$

e) $7-4 \Leftrightarrow 3$

f) $7-(3+2) \Leftrightarrow 2$

g)

$$\begin{aligned} & x\,[k+1] \ll *z \\ \Leftrightarrow\quad & x\,[3] \ll 3 \\ \Leftrightarrow\quad & 2 \ll 3 \\ \Leftrightarrow\quad & 2^4 \\ \Leftrightarrow\quad & 16 \end{aligned}$$

h)

$$\begin{aligned} & x\,[k+2] < *z \\ \Leftrightarrow\quad & x\,[3] < 3 \\ \Leftrightarrow\quad & 2 < 3 \\ \Leftrightarrow\quad & \text{true} \end{aligned}$$

i)

$$\begin{aligned} & x\,[k**z] \\ \Leftrightarrow\quad & x\,[2*(*z)] \\ \Leftrightarrow\quad & x\,[2*3] \\ \Leftrightarrow\quad & 5 \end{aligned}$$

j)

$$\begin{aligned} & x\,[k*(*z)+3]*(*z)+3 \\ \Leftrightarrow\quad & x\,[6+3]*3+3 \\ \Leftrightarrow\quad & -1*3+3 \\ \Leftrightarrow\quad & -3+3 \\ \Leftrightarrow\quad & 0 \end{aligned}$$

k) $x\,[x\,[k]] \Leftrightarrow x\,[7] \Leftrightarrow 4$

l) $x\,[*z+k] \Leftrightarrow x\,[3+2] \Leftrightarrow 3$

4.12 Funktion `items`

Siehe beigefügte Datei `Aufg.12.cc`.

4.13 Korrektur von `stringop`

Korrigierter Sourcecode mit Testbett und Kommentaren zu den Operationen von `stringop` in der Datei `Aufg.13.cc`. Die korrigierte Version macht aus `Fachhochschule Gießen` den String `FACHOSULEGIN` - sie macht alle Kleinbuchstaben zu Großbuchstaben, löscht alle Zeichen, die keine Buchstaben sind, ebenso Umlaute und ß, und entfernt alle Wiederholungen von Zeichen.

4.14 Funktion `linpol`

4.15 Werte von Ausdrücken

4.16 Numerische Berechnung der n-ten Ableitung

4.17 Deklaration von Objekten erkennen

a) Eine Funktion `g`, die einen Zeiger auf `int` und ein Feld von Zeilen mit 6 Elementen aus Zeigern auf `double` als Argumente erwartet und einen Zeiger auf `double` zurückgibt.

b) `fkt` ist ein Feld mit 5 Elementen vom Typ »Zeiger auf eine Funktion, die zwei Zeiger auf `double` als Argumente erwartet und einen Wert vom Typ »Zeiger auf ein Feld mit 15 `floats`« zurückgibt«.

c) `h` ist ein Zeiger auf eine Funktion, die ei Feld von Zeigern auf `int` als Argument erwartet und keinen Rückgabewert zurückgibt.

d) `u` ist ein Zeiger auf eine Funktion, die einen Zeiger auf ein Feld von 8 `ints` als Argument erwartet und einen Zeiger auf einen `unsigned int` zurückgibt.

e) `fc` ist eine Funktion. Sie erwartet einen Zeiger auf `int` und eine Funktion, die keine Argumente hat und `double` zurückgibt, als Argumente. Sie gibt einen Zeiger auf `double` zurück.

f) `f` ist ein Zeiger auf eine Funktion. Die Funtion erwartet einen Zeiger auf `double` als Argument und gibt einen Zeiger auf ein Feld von 8 `char` zurück.

g) `y` ist ein Feld von 8 Zeigern auf Funktionen. Die Funktionen erwarten einen `int` und eine Funktion vom Typ »erwartet 2 Zeiger auf float als Argument, gibt Zeiger auf int zurück« als Argument. Die Funktionen geben einen Zeiger auf `long int` zurück.

4.18 Deklaration von Objekten erstellen

4.19 Numerische Approximation der Kurvenlänge

4.20 Werte von Ausdrücken

4.21 Programm zur Verwaltung von Kundendaten

4.22 DBF-Dateien einlesen und Inhalt ausgeben

4.23 Deklarationen von Objekten erstellen

4.24 Geeignete Ausgaben zu einem Programm entwerfen

Siehe `Aufg.25.cc`.

4.25 Klasse `codierer` definieren

4.26 Ausgabe des Programms

Die Ausgabe des Programms ist:

```
7
8
11
5
-9 -8 -7 6 5 4
```

Dies kann mit dem Sourcecode in `Aufg.26.cc` auf einfache Weise selbst überprüft werden.

4.27 Klasse `datumsrechner` definieren

4.28 Operationen der Funktion `matrixop`

Die Funktion `matrixop` quadriert die übergebene Matrix dreimal: $\left(\left(a^2\right)^2\right)^2 = a^8$. Sie errechnet also die achte Potenz der übergebenen Matrix.

```
void matrixop(long a[][8]) {
  int i, j, k, m;
  long s, *c;
  c=new long[64];
  for (m=0; m<3; m++) {     //3mal: Quadriere die Matrix a
    for (i=0; i<8; i++) {   //fuer jede Zeile i von a
      for (j=0; j<8; j++) { //fuer jede Spalte j der Zeile i von a
        for (k=0, s=0; k<8; k++) //multipliziere Spalte i mit Zeile j von a
          s+=a[i][k]*a[k][j];
        c[8*i+j]=s; //Ergebnis in ebensolches Feld
        }
    }
    for (i=0; i<64; i++) //kopiere c nach a
      a[0][i]=c[i];
  }
  delete [] c;
}
```

4.29 Werte von Ausdrücken

4.30 Ergänzung der Klasse `datumsrechner`

4.31 Ausgabe des Programms

5 Lösungen zur Klausur vom WS 1999/00

Quelle: [8]

5.1 Aufgabe 1

a) Zulässig. `z` ist eine Funktion, die einen Zeiger auf einen Zeiger auf `char` zurückgibt und als Argument einen Zeiger auf eine Funktion erwartet, die einen Zeiger auf `int` als Argument braucht und char zurückgibt.

b) Zulässig. `z` ist eine Funktion, die ein Feld von 8 Zeigern auf `char` zurückgibt. Ihr Argument muss ein Zeiger auf eine Funktion sein, die einen `int` als Argument erwartet und einen Zeiger auf `char` zurückgibt.

c) Zulässig. `z` ist ein Zeiger auf eine Funktion, die einen Zeiger auf `char` zurückgibt. Ihr Argument ist ein Zeiger auf eine Funktion, die einen Zeiger auf einen Zeiger auf `int` als Argument erwartet und einen Zeiger auf `char` zurückgibt.

d) Zulässig. `z` ist ein Feld von 8 Zeigern auf Funktionen, die ein `char` zurückgeben. Das Argument dieser Funktionen ist ein Zeiger auf eine Funktion, die ein `int` als Argument erwartet und ein `char` zurückgibt.

e) Zulässig. `z` ist eine Funktion, die einen Zeiger auf ein Feld von 8 `char` zurückgibt. Ihr Argument muss ein Zeiger auf eine Funktion sein, die einen `int` als Argument erwartet und einen Zeiger auf `char` zurückgibt.

5.2 Aufgabe 2

Die Ausgabe des Programms ist:

```
11
26
7
```

Dies kann mit dem beiligenden Sourcecode `Aufg.WS99-00.2.cc` leicht selbst nachgeprüft werden. Zur Begründung dieses Verhaltens des Programms siehe z.B. [9, Kap. 7.8, S.172].

5.3 Aufgabe 3

a)

$$
\begin{aligned}
& *(z+i+k) \\
\Leftrightarrow\ & *(\&x\,[6]+2+-4) \\
\Leftrightarrow\ & *(\&x\,[4]) \\
\Leftrightarrow\ & 3
\end{aligned}
$$

b)

$$
\begin{aligned}
& x\,[i**z+1]-k \\
\Leftrightarrow\ & x\,[2*4+1]+4 \\
\Leftrightarrow\ & x\,[9]+4 \\
\Leftrightarrow\ & 4
\end{aligned}
$$

c)

$$
\begin{aligned}
& p\,[(p-x)+k]-i \\
\Leftrightarrow\ & p\,[(\&x\,[4]-\&x\,[0])-4]-2 \\
\Leftrightarrow\ & p\,[4-4]-2 \\
\Leftrightarrow\ & 1
\end{aligned}
$$

d)

$$
\begin{aligned}
& *(p-i)-z\,[k] \\
\Leftrightarrow\ & *(\&x\,[4]-2)-x\,[2] \\
\Leftrightarrow\ & *(\&x\,[2])-2 \\
\Leftrightarrow\ & 0
\end{aligned}
$$

e)

$$
\begin{aligned}
& z\,[z-p]+\ **zz-*(p+*p) \\
\Leftrightarrow\ & z\,[\&x\,[6]-\&x\,[4]]+x\,[6]-*(\&x\,[4]+3) \\
\Leftrightarrow\ & z\,[2]+4-9 \\
\Leftrightarrow\ & 0
\end{aligned}
$$

f)

$$
\begin{aligned}
& ((z\,[i]<4)+(z\,[i]<5)+(z\,[k<6]))*z\,[k] \\
\Leftrightarrow\ & ((x\,[8]<4)+(x\,[8]<5)+(z\,[\text{true}]))*x\,[2] \\
\Leftrightarrow\ & (\text{false}+\text{false}+x\,[7])*2 \\
\Leftrightarrow\ & (0+0+9)*2 \\
\Leftrightarrow\ & 18
\end{aligned}
$$

g)

$$
\begin{aligned}
& *(p-2*(*zz-p)) \\
\Leftrightarrow\ & *(\&x\,[4]-2*(\&x\,[6]-\&x\,[4])) \\
\Leftrightarrow\ & *(\&x\,[4]-4) \\
\Leftrightarrow\ & 1
\end{aligned}
$$

h)

$$
\begin{aligned}
& *(z - *(x + i - k) - i) \\
\Leftrightarrow \quad & *(\&x\,[6] - *(\&x\,[0] + 2 + 4) - 2) \\
\Leftrightarrow \quad & *(\&x\,[6] - 4 - 2) \\
\Leftrightarrow \quad & *(\&x\,[0]) \\
\Leftrightarrow \quad & 1
\end{aligned}
$$

5.4 Aufgabe 4

Die Richtung aller Verkettungen der einfach verketteten Liste wird herumgedreht. Der Kopf der Liste `kopf` enthält danach die Adresse des vorher letzten Elements, Nachfolger des vorher letzten Elementes ist das vorher zweitletzte ELement usw.. Das vorher erste Element ist nun das letzte Element.

5.5 Aufgabe 5

```
typedef struct {
  long kundennr;
  char name[25];
  int lb_tag;
  int lb_mon;
  int lb_jahr;
} KDAT;
// entferne alle Kunden, die zum letzten mal ein halbes Jahr
// oder laenger vor dem uebergebenen Datum bestellt haben.
void del_kunden(ELEMENT **kopf, int tag, int mon, int jahr) {
  ELEMENT *element = *kopf, //Zeiger auf aktuelles Element
    *vorelement = 0, //Vorgaenger von element
    *folgeelement = *kopf->nptr; //Nachfolger von element
  //Liste traversieren
  while (element) {
    //6 Monate zum Datum der letzten Bestellung addieren
    int lbplus6_tag = element->daten.lb_tag;
    int lbplus6_mon = element->daten.lb_mon + 6;
    int lbplus6_jahr = element->daten.lb_jahr;
    if (lbplus6_mon > 12) {
      lbplus6_mon %= 12;
      lbplus6_jahr++;
    }
    //pruefen, ob Datum kleiner oder gleich uebergebenem Datum
    if (lbplus6_tag <= tag &&
        lbplus6_mon <= mon &&
        lbplus6_jahr <= jahr) {
      //ggf. Zeiger auf Startelement anpassen
      if (*kopf == element)
        *kopf = folgeelement;
      //Kundendatensatz element loeschen
      delete element;
      if (vorelement != 0)
        vorelement->nptr = folgeelement;
      element = folgeelement;
    }
    else //naechstes Listenelement waehlen
      vorelement = element;
      element = folgeelement;
  }
```

```
  folgeelement = element->nptr;
}
```

6 Lösungen zur Klausur vom SS 1998

Quelle: [8]

6.1 Aufgabe 1

a) $x\,[i+4] \Leftrightarrow x\,[1] \Leftrightarrow 3$

b) $*p - j \Leftrightarrow 7 - 2 \Leftrightarrow 5$

c) $*\,(p-j) \Leftrightarrow *\,(\&x\,[1]) \Leftrightarrow 3$

d) $(*q)\,[-i] \Leftrightarrow 4$

e) $*\,(q\,[0]+j) \Leftrightarrow *\,((*q)+2) \Leftrightarrow *\,(p+2) \Leftrightarrow *\,(\&x\,[5]) \Leftrightarrow 6$

f) $x\,[4*(*q - \&x\,[j])] \Leftrightarrow x\,[4*(*\,(\&p) - \&x\,[2])] \Leftrightarrow x\,[4*(\&x\,[3] - \&x\,[2])] \Leftrightarrow x\,[4*1] \Leftrightarrow 8$

g) $p\,[-5 < i < 0] \Leftrightarrow p\,[(-5 < -3) < 0] \Leftrightarrow p\,[1 < 0] \Leftrightarrow p\,[0] \Leftrightarrow 7$

h) $x\left[x\,[j]\hat{}\,x\,[j-i]\ \&\ 0x00F2 \mid 1\right] \Leftrightarrow x\,[5\hat{}6\,\&\,0x00F2 \mid 1] \Leftrightarrow x\,[7] \Leftrightarrow 2$

6.2 Aufgabe 2

Es handelt sich um den bekannten straight insertion-Algorithmus. Damit wird hier ein Feld `x` (übergeben als Pointer auf seine Anfangsadresse) von `n` Elementen absteigend sortiert, indem: Prüfe für jedes Element `x[i]` aus `x` ab dem zweiten Element: Ist es größer als das vorangehende Element? Wenn ja, schiebe alle vorangehenden Elemente, die kleiner oder gleich sind, um einen Platz nach rechts und füge das vorherige `x[i]` an den freien Platz ein. Entgegen dem normalen straight insertion-Algorithmus ist diese nicht stabil.

6.3 Aufgabe 3

Müssen bei solchen Aufgaben unübersichtliche Ausdrücke ausgewertet werden, so hilft es, die zunächst entsprechend der Prioritätenrangfolge zu klammern. Wichtig: Bei Operatoren

```
printf("\n1. %d",PDIF(b-a,c-d));
⇔printf("\n1. %d",b-a>c-d ? b-a-c-d : c-d-b-a);
⇔printf("\n1. %d",5-3>6-1 ? 5-3-6-1 : 6-1-5-3);
⇔printf("\n1. %d",2>5 ? -5 : -3);
⇔printf("\n1. %d",-3);
⇔printf("\n1. -3");

printf("\n2. %d",PDIF(a,b)-NDIF(c,d));
⇔printf("\n2. %d",
  ((a>b) ? (a-b) : (((b-a)-c)<d)) ? (c-d) : (d-c));
⇔printf("\n2. %d",
  ((3>5) ? (a-b) : (((5-3)-6)<1)) ? (c-d) : (d-c));
⇔printf("\n2. %d", c-d);
⇔printf("\n2. %d", 5);

printf("\n3. %d",NDIF(c,d)-PDIF(a,b));
⇔printf("\n3. %d",NDIF(c,d)-PDIF(a,b));
⇔printf("\n3. %d",
```

```
  ((c<d) ? (c-d) : (((d-c)-a)>b)) ? (a-b) : (b-a));
⇔printf("\n3. %d",
  ((6<1) ? (c-d) : (((1-6)-3)>5)) ? (a-b) : (b-a));
⇔printf("\n3. %d", false ? (a-b) : (5-3));
⇔printf("\n3. %d", 2);

printf("\n4. %d", NDIF(c,NDIF(d,a)));
⇔printf("\n4. %d", c<NDIF(d,a) ? c - NDIF(d,a) : NDIF(d,a) - c);
⇔printf("\n4. %d",
  c< d<a ? d-a : a-d ? c - d<a ? d-a : a-d : d<a ? d-a : a-d - c);
⇔printf("\n4. %d",
  ((c<d)<a) ? (d-a) : ( (a-d) ? ( ((c-d)<a) ? (d-a) : (a-d) ) : ( (d<a) ? (d-a) : ((a-d)-c)))
⇔printf("\n4. %d",-2);
```

6.4 Aufgabe 4

a)

```
typedef struct e {
  WINPAR window;
  e *pre, *post;
} ELEMENT;
```

b) Die Musterlösung verwendet hier die Funktion `free(char *)`. Dies ist die Art, in C Speicher an den Heap zurückzugeben. In C++ dagegen verwendet man die Operatoren `delete` und `delete[]`.

```
void delwindow(ELEMENT *pointer) {
  //Zeiger pre des Nachfolgers richtig setzen
  (pointer->post)->pre = pointer->pre;
  //Zeiger post des Vorgaengers richtig setzen
  (pointer->pre)->post = pointer->post;
  //Element loeschen
  delete[] pointer->window.title;
  delete pointer;
}
```

c)

```
#include<stdio.h>
void deldouble(ELEMENT *kopf) {
  //fuer jedes Element der Liste:
  for (ELEMENT *element=kopf; element != 0; element = element->post) {
    char *title= element->window->title; //speichere Titel
    bool del_it = false; //aktuelles Element loeschen?
    //fuer alle Vorgaenger des Elements
    for (ELEMENT *vor=kopf; vor != element && !kill_it; vor = vor->post) {
      //merke Element zum Loeschen vor, wenn Titel gefunden
      if (strcmp(title, vor->window->title) == 0)
        kill_it = true;
    }
    if (kill_it) {
      ELEMENT vorelement=element->pre;
      delwindow(element);
      element=vorelement; //for-Schleife waehlt den Nachfolger!
    }
  }
}
```

6.5 Aufgabe 5

Bei solchen Aufgaben wähle man (einfache) Namen für die deklarierten Elemente.

a) Bei unvollständiger Feldinitialisierung darf nur die Anzahl der äußersten (»größten«) Elemente unbekannt sein, hier also der Zeilen:

```
int f(char *, float* [][4]);
```

b)

```
char (*g)(int* (*[])(char*) ,double *);
```

c)

```
typedef int* MATRIXTYP[4][4];
typedef MATRIXTYP* FKTTYP(int);
FKTTYP* m[3][6];
```

7 Errata und Ergänzungen

Die folgenden Anmerkungen beziehen sich auf [5], verwendet wurde die logische Seitennummerierung in diesem Dokument selbst.

S.28 »Variablendefinition mit Const«: Ersetze »und nur ihre statischen Methoden dürfen aktiviert werden« durch »und nur ihre konstanten Methoden dürfen aktiviert werden«.

S.47: Kapitel 2.4.9 Sprachgebrauch: »Referenz« meint eine Referenz auf eine Variable, d.h. ein anderer Name für diese Variable. Zum Beispiel das Referenzergebnis der Funktion:

```
Vektor & operator= (const Vektor &v) { ... return *this; }
```

»Verweis« meint einen Pointer, d.h. eine Variable, die eine Adresse einer anderen Variablen als Wert hat. `this` ist ein Pointer, dessen Wert stets die Startadresse des aktuellen Objektes ist. `*this` ist das aktuelle Objekt selbst, denn * ist der Dereferenzoperator. Die Funktion gibt also mit `return *this;` eine Referenz auf das aktuelle Objekt zurück, d.h. einen anderen Namen für das aktuelle Objekt.

S.114, Beispiel

```
*pc = 'X' ;
```

`pc` ist eine Referenz auf eine Variable vom Typ »Zeiger auf `char`«. `*pc` dereferenziert nicht `pc` selbst (Referenzen müssen nicht dereferenziert werden), sondern seinen Inhalt: einen Zeiger auf `char`.

```
pc = &b;
```

Hier wurde versucht, den Inhalt von `pc` (einen Zeiger auf `char`) zu verändern, nicht das Gezeigte. Das ist nicht möglich, denn `pc` ist konstant.

S.128: Tiefe Listenkopie

```
// Kopierkonstruktor: initialisiere mich
// mit einer tiefen Kopie von p_l
List::List (const List &p_l) {
  head = 0; //Pointer auf den ersten Knoten der Liste
  Node **l = &head, //in *l muss der Pointer auf einen neu kopierten Knoten eingesetzt werden
             *n; //Pointer auf den jeweils neu kopierten Knoten
  for(Node * p = p_l.head; p != 0; p = p->next){
    //INV: l ist der Pointer auf das Feld next des letzten
    //Knotens der Liste. *l ist 0.
    n = new Node (p->v, 0);
    *l = n; //neuen Knoten anhaengen
    l = &(n->next); //l auf das Feld next des neuen letzten Knotens setzen
  }
}
```

S.147: Kapitel 5.7.4 Ersetze »Der Leseoperator wird problemlos rekursiv definiert.« durch »Der Schreiboperator wird problemlos rekursiv definiert.«.

S.165, Kap. 6.3.4 Ersetze die Zeile »`template<class TT> friend class C; // Alle Instanzen von C`« durch »`friend class C; // Alle Instanzen von C`«.

S.166, Kap. 6.3.4 Ersetze die Zeile »`template<class TT> friend class C; // Alle Instanzen von C`« durch »`friend class C; // Alle Instanzen von C`«.

S.167, Kap. 6.3.5 Ersetze »`friend Vektor operator+ <T> (const Vektor &, const Vektor &);`« durch »`friend Vektor<T> operator+ <T> (const Vektor<T> &, const Vektor<T> &);`«. Oder ist das Auslassen der Templateparameter bei der Angabe eines Funktionstemplates als Freund erlaubt?

S.190, Kap. 7.4.2 Ersetze »`void h (B &b)`« durch »`void h (Basis &b)`«.

Literatur

[1] Drum D.: »Prog 2 Mitschrift zur Vorlesung bei Prof. Hoffmann«; vor 2002-03-04; Quelle eigentlich http://homepages.fh-giessen.de/~hg11604/dls/Prog2SS01.pdf, zur Zeit jedoch nur http://www.rolfhub.de. Diese Mitschrift entspricht zumindest zu Beginn der Vorlesung genau dem behandelten Stoff.

[2] »Hoffmann: Programmierung II (C) - Kompakt«. Skript und Lösung zu mehreren nicht mehr aktuellen Übungsaufgaben als ASCII-Text. Quelle: Enthalten in der Skriptsammlung der Fachschaft MNI der FH Gießen-Friedberg http://www.fh-giessen.de/FACHSCHAFT/Informatik/cgi-bin/navi01.cgi?skripte als Skript zu Programmierung II bei Prof. Hoffmann, direkte Adresse http://www.fh-giessen.de/FACHSCHAFT/Informatik/data/skripte/prog2.zip. Dieses Dokument ist eine Art Kurzreferenz zu C, also im Wesentlichen für die heutige Veranstaltung »Programmierung II« bei Prof. Hoffmann unbrauchbar.

[3] Prof. Hoffmann: »Programmierung II«. Dieses Skript besteht hauptsächlich aus Ergänzungen zu [5], das Prof. Hoffmann für zu umfangreich hält; er wird also nicht alles daraus behandeln und hat teilweise eigene Kapitel geschrieben. Quelle: ALABAMA-Server, in Form von einzelnen Kapiteln.

[4] Prof. Hoffmann: »Übungsaufgaben Programmierung II (C++)«. Sie sind inkl. Lösungen auf dem ALABAMA-Server vorhanden. Die Übungsaufgaben sind in 11 Dateien `Prgiia01.pdf` bis `Prgiia11.pdf` enthalten. Weitere Quelle für die Übungsaufgaben: http://www.rolfhub.de/de/study.ss02.phtml.

[5] Prof. Dr. Thomas Letschert: »Programmierung II C++«; FH Gießen-Friedberg; Version vom 21. Januar 2002. Quelle: http://velociraptor.mni.fh-giessen.de/Programmierung/progII.pdf.
Professor Hoffmann geht in seiner Vorlesung »Programmierung II« nicht genau nach diesem Skript vor, sondern behandelt weniger Stoff, tw. aber mit eigenen Ergänzungen. Vergleiche dazu [3].

[6] Prof. Dr. Thomas Letschert: »Programmierung II; Erweitertes Kapitel Vererbung und Polymorphismus«; FH Gießen-Friedberg; Version vom 21. Januar 2002. Quelle: http://velociraptor.mni.fh-giessen.de/Programmierung/erganz.pdf. Dieses Dokument ist bereits vollständig in [5] (Version vom 21. Januar 2002) enthalten und wird also nicht benötigt; es ist eine Erweiterung zur alten Skriptversion.

[7] Prof. Dr. Thomas Letschert: »Programmierung II; Aufgaben und Lösungshinweise«; FH Gießen-Friedberg; Version vom 21. Januar 2002. Die Aufgaben sind bereits völlig identisch in [5] enthalten, die Lösungshinweise jedoch nicht. Quelle: http://velociraptor.mni.fh-giessen.de/Programmierung/uebloes.pdf.

[8] In der Fachschaft MNI der Fachhochschule Gießen-Friedberg sind folgende Klausuren von Prof. Hoffmann zum Kopieren erhältlich:

- Geisse / Hoffmann: »Extra-Klausur Programmierung II/Ib«; 1996-01-22; mit Lösungsblatt.
- Hoffmann: »Klausur Programmierung II«; SS 1997; Gruppe A; mit Lösungsblatt.

- Hoffmann: »Klausur Programmierung II«; SS 1997; Gruppe B; ohne Lösungsblatt.
- Hoffmann: »Klausur Programmierung II«; WS 1997/1998; mit Lösungsblatt.
- Hoffmann: »Klausur Programmierung II«; SS 1998; mit Lösungsblatt.
- Hoffmann: »Klausur Programmierung II«; WS 1999/2000; ohne Lösungsblatt.

[9] Bjarne Stroustrup: »Die C++ Programmiersprache«; 4., aktualisierte Auflage; Deutsche Ausgabe der Special Edition; ©2000 Addison-Wesley Verlag; ISBN 3-8273-1756-8.