Technische Hochschule Mittelhessen

Prof. Dr.-Ing. A. Christidis

**Fachbereich Mathematik,
Naturwissenschaften und Informatik**

15. Februar 2013

Klausur
Konzepte Systemnaher Programmierung
WS 2012 / 13

– Lösungshilfe –

Personalien:

Name, Vorname: .....

Matrikelnummer: .....

Hinweise:

- **Die Bearbeitungszeit beträgt 90 Minuten.**
- **Alle schriftlichen Hilfsmittel sind zugelassen; andere Hilfsmittel, insb. elektr. Rechen- und Kommunikationsapparate dürfen nicht verwendet werden.**
- **Ausgesprochene Folgefehler (durch Übertragung falscher Zwischenergebnisse) werden in Folgerechnungen als richtig gewertet.**
- **Die Aufgaben sollen nur auf diesen Blättern (inkl. Rückseite) bearbeitet werden. Bei Bedarf kann zusätzliches Papier zur Verfügung gestellt werden.**
- **Zur sicheren Zuordnung aller Lösungen wird um eine persönliche Kennung (Name u./o. Matrikelnr.) auf allen Blättern gebeten.**
- **Auf Wunsch darf auch Bleistift verwendet werden.**

Zur leichteren Lesbarkeit werden Substantive nur in einem Geschlecht („Nutzerin“) verwendet.

1. **Aufgabe** (15 Punkte)

a) Was ist eine Software-Plattform?

Die ‚tiefste' (Hw-nächste) genutzte Sw-Schicht – i.d.R. das Betriebssystem.

b) Was ist eine Software- Umgebung?

Ein Satz von Programmen, die für eine bestimmte Nutzung die Kommunikation mit der Plattform übernehmen.

c) Von einer Entwicklergruppe, die Sie sehr schätzen, hören Sie, daß ihre Programme am besten auf ihrer eigenen Software-Plattform und Software-Umgebung laufen. Nur eine der beiden soll in einer Hochsprache programmiert sein. Welche ist es am wahrscheinlichsten? (Bitte begründen Sie Ihre Aussage!)

Die Software-Umgebung ist vermutlich in einer Hochsprache codiert; denn sie ist von der Hardware weiter entfernt („höher") als die Plattform.

d) Was ist eine Kollision?

Kollision ist die gleichzeitige Nutzung exklusiv (seriell) nutzbarer Ressourcen durch mehr als einen Prozeß.

e) Was verstehen Sie unter einer Blockierung?

Blockierung heißt das Warten auf Ereignisse, die nicht eintreten.

f) Welche der folgenden Aussagen können Sie bestätigen?

X	Eine Blockade ist immer auch eine Blockierung.
	Eine Blockierung ist immer auch eine Blockade.
	Eine Blockade ist nicht immer eine Blockierung.
X	Eine Blockierung ist nicht immer eine Blockade.

g) Kann allgemein die Vermeidung von Kollisionen zu einer Blockierung führen?

Wenn ja: Zu welcher/n Blockierungsart/en? (Aufzählung genügt)

Wenn nein: Welcher Umstand sorgt dafür, daß dies nicht eintritt? (Nennung genügt)

Ja. Die Vermeidung von Kollisionen kann zu allen Blockierungsarten führen: Verhungern, Blockade, Verklemmung.

2. **Aufgabe** (15 Punkte)

Freunde, die Ihre Erfolge in KSP schätzen, bitten Sie um Unterstützung in einem Softwareprojekt:

Das Programm `NotenPad.c` (s.u.) soll später in Hochschulen eingesetzt werden. Erste Schritte konnte die Entwicklergruppe mit Programm-Fragmenten aus dem Internet tun. Es ergaben sich aber Fragen, zu deren Beantwortung Ihr Fachwissen benötigt wird.

Eine zentrale Rolle nehmen die Strukturen `Lernfach` und `Lehrfach` ein (s. Code-Anfang); sie sollen in späteren Phasen dieses Projektes ergänzt werden und die Daten von Lernenden und Lehrenden aufnehmen. Exemplarisch werden schon heute Inhalte mit gleicher Bedeutung (z.B. die Nummer eines Faches in der Modul-Liste) für die beiden Personengruppen unterschiedlich codiert (hier: als `char Nummer[LANG]` bzw. als `int ListNr`).

Beantworten Sie bitte folgende Fragen:

a) Was ist in der C-Notation eine Struktur?

Die Sammlung von Variablen („Strukturelementen“) unter einem Namen.

b) Was bewirkt das „Data Alignment“ bei Strukturen?

Daß ihr Speicherbedarf ein ganzzahliges Vielfaches des größten darin enthaltenen Datentyps ist.

c) Haben die beiden o.a. Strukturen in `NotenPad.c` eigene Namen?

Wenn ja: Wie lauten die Strukturnamen?

Wenn nein: Woran erkennen Sie das?

Nein. Die Strukturen haben keine eigenen Namen. Dieser müßte hinter dem Bezeichner `struct` erscheinen.

d) Definieren die beiden Strukturen neue Datentypen?

Wenn ja: Wie heißen diese Datentypen?

Wenn nein: Welche andere Bedeutung haben sie für das Programm?

Ja. Die beiden Datentypen heißen `Lernfach` und `Lehrfach` .

e) Welche Konsequenz hat die Einrichtung der beiden Strukturen für das Programm und seine Ausführung?

Zutreffende Aussage(n) bitte ankreuzen:

	Es werden die globalen Variablen `Lernfach` und `Lehrfach` eingerichtet.
X	Dem Compiler wird mitgeteilt, daß im nun folgenden Code evtl. auch Variablen des jeweiligen Typs benötigt werden.
	Die Elemente der beiden Strukturen erhalten eine Initialisierung.
	Der Compiler wird angewiesen, Speicherplatz für mindestens je eine Kopie der angelegten Strukturen bereitzustellen.

f) `main()` enthält zwei Vereinbarungen mit den beiden Strukturen:

```
Lernfach *lern=NULL;
Lehrfach *lehr=NULL;
```

Was bewirken diese Zeilen? (Zutreffendes bitte ankreuzen.)

X	Sie richten je eine Zeigervariable vom Typ `Lernfach` und vom Typ `Lehrfach` ein.
	Sie richten je eine Variable vom Typ `Lernfach` und vom Typ `Lehrfach` ein.
	Sie löschen die beiden Strukturen `Lernfach` und `Lehrfach`, indem sie ihnen die Adresse `NULL` zuordnen.
	Es werden zwei Strukturen vereinbart, die zwar dieselben Namen `Lernfach` und `Lehrfach` tragen, aber nicht die o.a. Elemente, sondern Zeiger auf solche Elemente enthalten.

3. **Aufgabe** (40 Punkte)

a) Der Versuch, `NotenPad.c` in der vorliegenden Form zu compilieren, scheitert. Die Entwicklungsumgebung meldet, der Bezeichner `next` in der Funktion `main()` sei nicht definiert und markiert als fehlerhaft die zwei Stellen, in denen er vorkommt, nämlich

die Wertzuweisung: `next = set;`

und den Aufruf: `next (lern, lehr);`

Was schließen Sie daraus? (Zutreffende Aussage/n bitte ankreuzen)

	`next` ist eine C-Funktion.
X	`next` ist eine Variable.
	`next` ist eine Direktive.
X	`next` kann die Adresse einer Funktion zugewiesen bekommen.
	`next` ist ein flexibler `void`-Zeiger, der mehrere Rollen spielt.

b) Geben Sie bitte eine passende Deklaration für `next` an und initialisieren Sie den Bezeichner mit einem Wert, der ein Debugging erleichtert:

```
int (*next)(Lernfach*, Lehrfach*)=NULL;
```

c) `NotenPad.c` läßt sich nun compilieren. In der Parameterliste der ersten aufgerufenen Funktion `checkNget()` fällt auf, daß die beiden übergebenen Argumente jeweils zwei vorangestellte Sternchen (`**`) haben.

Bitte kreuzen Sie die korrekt/en unter den folgenden Aussagen an:

	Die C-Notation „`**Variable`“ kann gelesen werden als: „Inhalt vom Inhalt der Variablen“. Bei Strukturen bedeutet das, daß nur Werte (keine Adressen) der Strukturelemente übergeben werden.
X	Die Adresse der Adresse einer Variablen übergibt man einer Funktion typischerweise dann, wenn in der Funktion die Adresse der Variablen geändert werden soll.
	Adressen von Adressen kommen in der C-Programmierung äußerst selten vor und zeugen von schlechtem Programmierstil, sofern sie nicht der Lehre dienen.
	Die Verwendung von „`**Variable`“ ist hier nur ein Vorgriff auf den geplanten Ausbau des Programms mit mehreren, indizierten Strukturvariablen. Für die hier betrachtete Version des Programms hätte es dieselbe Wirkung, wenn der Prototyp heißen würde: `void checkNget (Lernfach *lern, Lehrfach *lehr);`

d) Der Start des Programms in der aktuellen Fassung beginnt mit der Ausführung der Funktion `checkNget()` und erzeugt die Ausgabe nach Abb. 1:

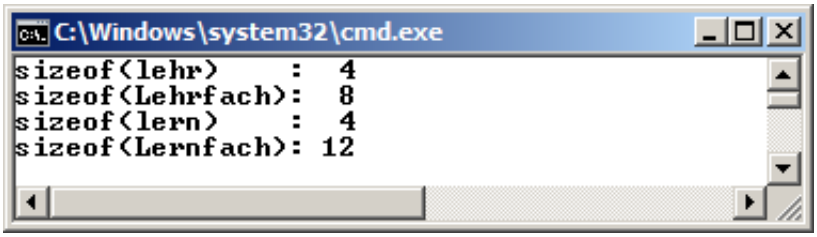

Abb. 1

Erörtern Sie bitte mit wenigen Worten, wie die vier Zahlenwerte zu erklären sind:

(i) ***`lehr` ist eine Zeigervariable (Adresse einer Adresse) und ist immer 4-Byte-groß.***

(ii) ***`Lehrfach` enthält als Struktur zwei `int`-Variablen mit je 4 Byte; die Summe beträgt 8 Byte.***

(iii) ***lern ist (wie `lehr`) eine Zeigervariable (Adresse einer Adresse) und ist ebenfalls 4-Byte-groß.***

(iv) ***`Lernfach` enthält als Struktur eine `float`-Variable, die 4-Byte-groß ist. Aufgrund des Data-Alignment kann die Strukturgröße nur ein ganzzahliges Vielfaches von 4 sein. Die Summe mit den 6 Byte (Makro-Konstante `LANG`) ergäbe 6+4=10; sie wird auf 12 erhöht.***

e) Nach Bearbeitung von `checkNget()` beendet sich das Programm. Das ist nicht beabsichtigt, aber auch die schrittweise Ausführung im Debugger bestätigt, daß die letzte ausgeführte Anweisung jene in `main()` ist:

```
if (lern && lehr)        next = set; else exit(0);
```

Erklären Sie bitte kurz die `if`-Bedingung in der Klammer, indem Sie die Fragen beantworten:

Worauf werden die Variable `lern` und die Variable `lehr` geprüft, damit entschieden wird, welche der Anweisungen (vor bzw. hinter `else`) ausgeführt wird?

Sie werden darauf geprüft, ob die Adressen (noch) den Wert `NULL` haben.

Entscheidet schon das Überprüfungsergebnis einer der Variablen, oder müssen grundsätzlich beide Variablen überprüft werden?

Es genügt, wenn eine der Variablen den Wert NULL hat; dann wird der `else`-Zweig ausgeführt.

f) Nach weiterer Befassung mit dem Problem erkennen Sie, daß der Code-Abschnitt von `checkNget()` mit der Überschrift `/*Speicherplatz-Reservierung:*/` fehlerhaft sein muß.

Erläutern Sie bitte kurz, wieso hier Speicherplatz reserviert (und offenbar benötigt) wird, indem Sie auf folgende Fragen antworten:

(i) Wieso reichen nicht die übergebenen Variablen `lern` und `lehr`, um die Werte der Struktur-Elemente unterzubringen? Was hat es mit der Differenz zwischen den o.a. Größen von `Lernfach` und `lern` bzw. `Lehrfach` und `lehr` zu tun?

Was übergeben wird, sind nicht Strukturen, sondern nur je 4 Byte mit der Adresse, unter welcher die Adresse der jeweiligen Struktur gespeichert ist. Mit den o.a. Größen-unterschieden hat dies nichts zu tun: Der Speicherplatz für Strukturelemente ist unabhängig vom Speicherplatz für die Adresse einer Struktur.

(ii) Wozu braucht eine Funktion Speicherplatz, unmittelbar, bevor sie verlassen wird?

Der Speicherplatz wird nicht in der Funktion benötigt; er wird reserviert und dem gesamten Programm zur Speicherung der beiden Strukturen zur Verfügung gestellt.

g) Geben Sie bitte die korrekte Formulierung der beiden `calloc`-Anweisungen an, unabhängig davon, ob nur eine oder beide geändert werden müssen:

```
*lehr = (Lehrfach *)calloc (NFACH, sizeof(Lehrfach));

*lern = (Lernfach *)calloc (NFACH, sizeof(Lernfach));
```

4. **Aufgabe** (10 Punkte)

Sie befassen sich mit der Funktion `set()` des Programms `NotenPad.c`.

Über die Parameterliste erhält `set()` Zugriff auf die eingerichteten Strukturen und ihre Elemente (s. Datei-Anfang). In dieser Funktion wird das Gewicht jedes Faches eingestellt. Während für Lernende (`Lernfach`) der Faktor (`float Faktor`) maßgeblich ist, mit dem die Fach-Note in das Abschlußzeugnis eingeht, zählt für Lehrende (`Lehrfach`) die Anzahl der Lehrstunden (`int LehrWS`), die sie wöchentlich für das jeweilige Fach ableisten.

In der aktuellen Erprobungsphase werden in einer `for`-Schleife belanglose Werte zugewiesen, die mit der Zählvariablen `j1` gebildet werden. Die Ausgabe auf Konsole (Abb. 2) zeigt, daß dies korrekt geschieht.

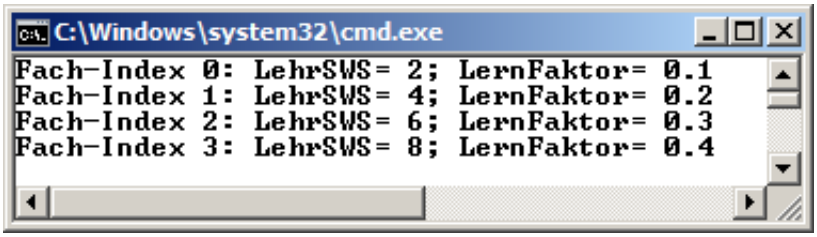

Abb. 2

a) Dennoch fällt in `set()` auf, daß auf die Elemente der beiden Strukturen unterschiedlich zugegriffen wird, obwohl die Parameterliste den Zugriff darauf in gleicher Form herstellt (allgemein: `Typ *Variable`). Zur Schaffung von Klarheit wollen Sie bitte in der folgenden Liste jene Ausdrücke ankreuzen, die Adressen (ggf. auch: Adressen von Adressen) kennzeichnen:

X	`lern`
	`lern[j1]`
X	`lern+j1`
X	`lehr`
	`lehr[j1]`
X	`lehr+j1`

b) Kreuzen Sie bitte die zutreffenden Voraussetzungen für die Verwendung von '`.`' bzw. '`->`' an, und setzen Sie das Kreuz jeweils in die Spalte des richtigen Operators:

`.`	`->`	Der Zugriff auf Elemente einer Struktur erfolgt mit …
X		… wenn der Zugriff über die Struktur selbst geschieht.
		… ausschließlich in Funktionen, die die Definition der Struktur als lokale Größe enthalten.
		… nur in Ausdrücken, in denen die Elemente links vom Gleichheitszeichen stehen (sog. „L values“).
	X	… dort, wo er über einen Zeiger auf die Struktur stattfindet.
		… wenn nicht auf den Wert des Elementes, sondern auf seine Adresse zugegriffen werden soll.
		… wenn die Struktur global verfügbar ist.

c) Beim Versuch, in `set()` die Ausdrücke mit den Variablen `lern` und `lehr` zu vereinheitlichen, stellen Sie fest, daß der verwendete Ausdruck `(lehr+j1)->LehrWS` ebensogut ersetzt werden kann durch (korrekte Ausdrücke bitte ankreuzen):

X	`(&(lehr[j1]))->LehrWS`
	`lehr->(LehrWS+j1)`
	`lehr.LehrWS[j1]`
X	`lehr[j1].LehrWS`

5. **Aufgabe** (20 Punkte)

Die Belegung weiterer Strukturelemente gibt Anlaß zur Erweiterung des Programms um die Funktion `put()`; sie soll die Daten in eine Datei übertragen. Dabei werden Fragen aufgeworfen, die Sie bitte beantworten wollen.

a) `put()` soll in `main()` aufgerufen werden. Dazu wird ihr Code um eine Zeile erweitert zu:

```
/*Initialisierungen, Wertzuweisungen:*/
if (lern && lehr)      next = set; else exit(0);
if (!next(lern, lehr)) next = put; else exit(0); //neu
next (lern, lehr);
```

Beim Testen stellen Ihre Freunde fest, daß die neue Funktion `put()` überhaupt nicht aufgerufen wird. Nach kurzer Suche können Sie darauf hinweisen, daß man ständig eine Compiler-Warnung übersehen hatte, daß die Funktion `set()` keinen Wert zurückgab.

Wie lautet die fehlende C-Anweisung in `set()`, damit `put()` im obigen Code-Fragment aufgerufen wird?

`return(0);`

b) Bei erneuter Überprüfung des Quellcodes fällt auf, daß auch `checkNget()` keine solche Anweisung enthielt, ohne daß der Compiler eine Warnung abgegeben hätte.

Wie ist dieses Verhalten des Compilers zu erklären?

Eine `return`-Anweisung ist bei `checkNget()` entbehrlich, weil die Funktion vom Typ `void` ist: Sie gibt keinen Wert zurück.

(Der Rücksprung als solcher wird implizit bei der schließenden Klammer am Ende des Funktion-Codes ausgeführt.)

c) In der Funktion `put()` wird eine Datei eröffnet. Kreuzen Sie bitte die Aussagen, die für diese Operation zutreffen:

	Die Datei wird zum Lesen und Schreiben eröffnet.
X	Die Datei wird nur zum Schreiben eröffnet.
	Das eröffnende Programm darf nur Einträge lesen, die es bei der aktuellen Ausführung selbst erzeugt hat.
X	Das eröffnende Programm darf überhaupt keine Einträge aus der Datei lesen.
X	Bei der Eröffnung der Datei werden evtl. vorhandene Einträge gelöscht.
	Nach Eröffnung der Datei werden Einträge hinter die vorhandenen geschrieben.
	Nach Eröffnung der Datei werden Einträge vor die vorhandenen geschrieben.
	Die Datei wird im übergeordneten Verzeichnis eröffnet.
	Die Datei wird im Desktop-Verzeichnis eröffnet.

In dieser Version funktioniert `put()` wie gewünscht und gibt die erwarteten Meldungen aus (Abb. 3).

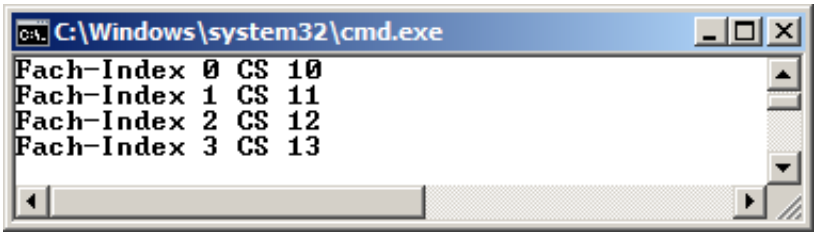

Abb. 3

Beantworten Sie bitte folgende Fragen zur Codierung von `put()`:

d) In der `for`-Schleife, in der die Meldungen nach Abb. 3 entstehen, wird je nach dem Wert der Zählvariablen `j1` entschieden, ob die Daten aus der Struktur `Lernfach`, oder jene aus `Lehrfach` verwendet werden.

Bei welchen Werten von `j1` wird die Struktur zur Variablen `lern`, bei welchen jene zu `lehr` eingesetzt, und woran erkennen Sie das?

(Stichwortartige Begründung genügt.)

Die Struktur zu `lern` wird bei geradezahligen `j1`-Werten (genauer: 0 und 2), jene zu `lehr` bei ungeradezahligen (1 und 3) ausgegeben. Das bewirkt die bitweise UND-Verknüpfung (`j1 & 1`).

e) Erläutern Sie bitte kurz, wie die Inkrementierung der „Lernfächer“ erfolgt. Gibt es insb. einen höchsten Wert für die Makro-Konstante `NFACH`, ab der die `for`-Schleife keine sinnvollen Fach-Indizes mehr liefert?

Die Inkrementierung der „Lehrfächer“ erfolgt über die Erhöhung des ASCII-Wertes der letzten Ziffer, beginnend bei '0'; d.h., nach der '9' kommen nicht-numerische Zeichen (konkret: der Doppelpunkt ':').

Der höchste sinnvolle Wert für `NFACH` ist somit 10.

f) Sie starten das Programm im Debugger, halten es am Ende der `for`-Schleife an (Anweisung `printf("\n");`) und öffnen die erzeugte Datei im Editor.

Welches der u.a. Erscheinungsbilder erwarten Sie zu diesem Zeitpunkt? (Bitte ankreuzen!)

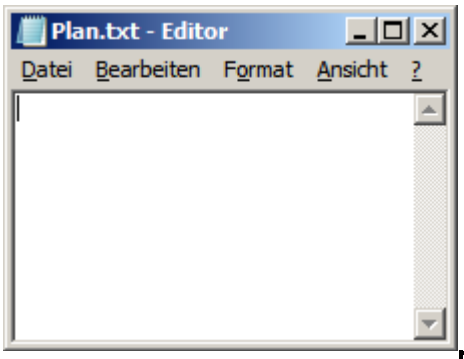

Abb. 4 [X]

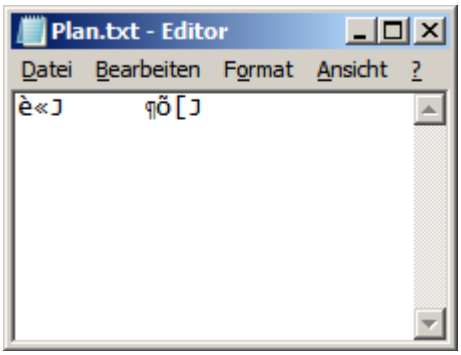

Abb. 5 [ ]

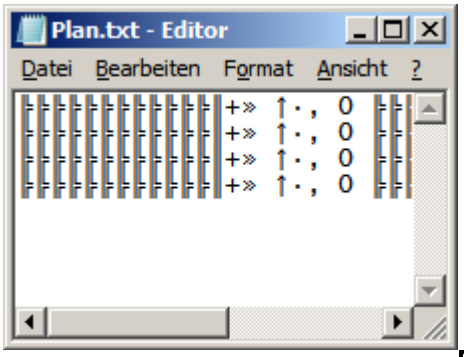

Abb. 6 [ ]

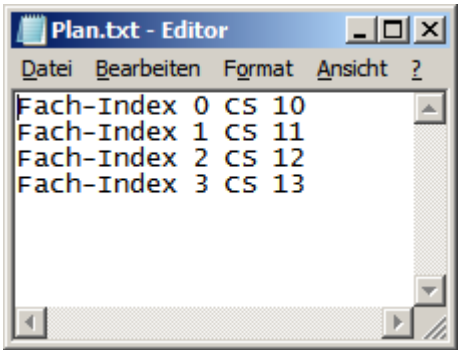

Abb. 7 [ ]

Bitte begründen Sie kurz Ihre Erwartung:

Das Programm ist noch nicht beendet, und es gab keinen Aufruf von `fflush()` oder `fclose()`. Die Datei ist leer. (Sie hat wegen "w" in `fopen()` auch keinen älteren Inhalt.)

```
/* NotenPad.c */
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>

#define LANG    6
#define NFACH   4
#define STRLEN  7

/*Strukturen und Datentypen:*/
typedef struct
{ char  Nummer[LANG];
  float Faktor;
} Lernfach;

typedef struct
{ int   ListNr;
  int   LehrWS;
} Lehrfach;


/*************************************************************************/
  void checkNget (Lernfach **lern, Lehrfach **lehr)
/*************************************************************************/
{ /*Groessen:*/
  printf ("sizeof(lehr)    :%3d\n", sizeof(lehr));
  printf ("sizeof(Lehrfach):%3d\n", sizeof(Lehrfach));
  printf ("sizeof(lern)    :%3d\n", sizeof(lern));
  printf ("sizeof(Lernfach):%3d\n", sizeof(Lernfach));  printf ("\n");

  /*Speicherplatz-Reservierung:*/
  lehr = calloc (NFACH, sizeof(Lehrfach));
  lern = calloc (NFACH, sizeof(Lernfach));
}


/*************************************************************************/
  int set (Lernfach *lern, Lehrfach *lehr)
/*************************************************************************/
{ int j1=0;

  /*Werte fuer LehrWS und Faktor:*/
  for (j1=0; j1<NFACH; j1++)
  { lehr[j1].LehrWS= 2*(j1+1);
    printf ("Fach-Index %d: LehrSWS=%2d; ", j1, (lehr+j1)->LehrWS);

   (lern+j1)->Faktor = .1f*(j1+1);
    printf ("LernFaktor=%4.1f\n", lern[j1].Faktor);
  } printf ("\n");

}
```

```
/*************************************************************************/
  int put (Lernfach *lern, Lehrfach *lehr)
/*************************************************************************/
{ int j1=0;
  FILE *memo;

  if (memo=fopen("Plan.txt","w"));  else return(0);

  /*Werte fuer Fach-Nummer:*/
  for (j1=0; j1<NFACH; j1++)
  {(lehr+j1)->ListNr = 10 + j1;
    sprintf (((lern+j1)->Nummer), "CS 1%c", '0'+j1);
    if (j1 & 1)
    { printf ("Fach-Index %d CS %2d\n", j1, (lehr+j1)->ListNr);
      fprintf(memo, "Fach-Index %d CS %2d\n", j1, (lehr+j1)->ListNr);
    }
    else
    { printf ("Fach-Index %d %s\n", j1, (lern+j1)->Nummer);
      fprintf (memo, "Fach-Index %d %s\n", j1, (lern+j1)->Nummer);
    }
  } printf ("\n");

  fclose (memo);
  return (0);
}


/*************************************************************************/
  int main (void)
/*************************************************************************/
{ int j1=0;
  Lernfach *lern=NULL;
  Lehrfach *lehr=NULL;

  int (*next)(Lernfach*, Lehrfach*)=NULL;

  /*Ueberpruefung, Speicherplatz-Reservierung:*/
  checkNget (&lern, &lehr);

  /*Initialisierungen, Wertzuweisungen:*/
  if (lern && lehr)        next = set; else exit(0);
  next (lern, lehr);

 _getch();
  return (0);
}
```

Platz für Notizen: