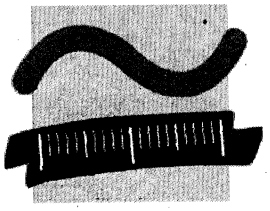
Dipl.Phys. Gerald Kempfer

Beuth Hochschule für Technik Berlin
- University of Applied Sciences -
Fachbereich VI – Informatik und Medien
Studiengang Technische Informatik Bachelor



Klausur zur Lehrveranstaltung IN2 TI-B (Informatik II) 24. Juli 2009

Hinweise:

Die Teilnahme an der Klausur ist nur bei Bestehen der Übungsaufgaben zulässig!

Zum Bestehen der Klausur benötigen Sie 50 Punkte von 100 möglichen Punkten.

Die Bearbeitungszeit der Klausur beginnt pünktlich um 08.00 Uhr und endet – ebenfalls pünktlich – um 09.30 Uhr. Sie haben also 90 Minuten Zeit.

Folgende Hilfsmittel in Papierform sind zugelassen: Skripte, Mitschriften und Bücher. **Nicht** zugelassen sind elektronische Geräte (Handys, Notebooks, PDAs, usw.).

Schreiben Sie mit Kugelschreiber oder Füller, **nicht** mit Bleistift! Verwenden Sie **nicht** die Farbe Rot. Schreiben Sie leserlich – was ich nicht lesen kann, ist grundsätzlich falsch! Beschreiben Sie nur die Vorderseiten!

Jeder Austausch mit anderen Personen wird als Täuschungsversuch gewertet und führt dazu, dass die Klausuren aller Beteiligten als „nicht bestanden" gewertet werden.

Erläuterungen sollten kurz, aber dennoch präzise und vollständig sein. Wenn möglich ist eine stichpunktartige Beantwortung zu wählen, sofern die Verständlichkeit gegeben ist. Im Zweifelsfall können ganze Sätze Klarheit schaffen.

Kennzeichnen Sie jedes Blatt, das Sie abgeben, mit Ihrem Namen und / oder Ihrer Matrikelnummer.

Viel Erfolg!

Name: [redacted]

Matrikelnummer: [redacted]

letzter Prüfungsversuch: ☐

Bewertung:

Aufgabe	Mögliche Punktzahl	Erreichte Punktzahl
1	10	10
2	15	13
3	25	25
4	25	25
5	25	25
Summe	**100**	98

Note Klausur 1,0

Aufgabe 1: Finden Sie im folgenden Programm alle Syntaxfehler. (10 / 10)
Markieren Sie die Fehler und schreiben Sie hinter bzw. unter die Zeile, wie die Zeile richtig aussehen muss.

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

int main()
{
   struct TArtikel
   {
      int ArtNr;
      int Status;
      struct TArtikel *NextArtikel;
   } *Artikel, *Start;
   int i;

   Artikel = malloc(siseof(struct TArtikel));

   Start = Artikel;
   for (i = 0; i < 10. )
   {
      Artikel->ArtNr = ++i;
      Artikel->Status = 0xr2d2 | i;
      if (i < 10)
         Artikel->NextArtikel = malloc(sizeof(struct_Artikel));
      else
         Artikel->NextArtikel = Null;
      Artikel = Artikel->NextArtikel;
   }

   Artikel = Start;
   for (i = 20; i => 11; i--)
   {
      printf("ArtNr : %i\n", Artikel->ArtNr);
      printf("Status: %i\n", Artikel.Status);
      Artikel = (*Artikel).NextArtikel;
   }

   while (Start <> NULL)
   {
      Artikel = Start->NextArtikel;
      free(Start);
      Start = Artikel;
   }

   return 0;
   }
}
```

Aufgabe 2: Multiple-Choice-Fragen. (13/ 15)
Kreuzen Sie an, wie der Satz richtig heißen muss. Es gibt immer nur eine richtige Antwort.

Eine andere Schreibweise für `Ptr->Fld` (`Ptr` ist ein Zeiger auf eine Struktur, die das Feld `Fld` beinhaltet) ist
- [x] `(*Ptr).Fld`
- [ ] `*(Ptr).Fld`
- [ ] `*(Ptr.Fld)`

Eine make-Datei
- [ ] kann wie eine Header-Datei in ein C-Programm eingebunden werden.
- [x] beschreibt Abhängigkeiten.
- [ ] kann vom C-Compiler kompiliert werden.

Mit dem Schlüsselwort `struct` wird eine
- [ ] Programmstruktur definiert.
- [ ] Computerstruktur definiert.
- [x] Datenstruktur definiert.

Mit dem Präprozessorbefehl `#define` wird eine
- [ ] eine Variable definiert.
- [x] eine Konstante definiert.
- [ ] eine Funktion definiert.

Reservierte Speicherbereiche werden freigegeben mit der Funktion
- [x] `free`
- [ ] `erase`
- [ ] `remove`

Der Unterschied zwischen Text- und Binär-Modus bei der Datei-Ein- und Ausgabe liegt u.a.
- [x] in der Erkennung der deutschen Umlaute.
- [x] in der Erkennung des Dateiendes.
- [ ] in der Schreib- und Lesegeschwindigkeit.

Eine Variable vom Typ `int **` belegt
- [ ] viermal so viel Speicher wie eine Variable vom Typ `int *`.
- [ ] doppelt so viel Speicher wie eine Variable vom Typ `int *`.
- [x] genau so viel Speicher wie eine Variable vom Typ `int *`.

Eine Struktur darf
- [x] keine Unterstruktur enthalten.
- [x] beliebig viele Unterstrukturen enthalten.
- [ ] nur eine Unterstruktur enthalten.

und was haben wir in der Übung gemacht?

Wenn mit der `fopen`-Funktion eine Datei nicht geöffnet werden kann, gibt die Funktion

- ☒ einen `NULL`-Zeiger zurück.
- ☐ den Wert `EOF` zurück.
- ☐ gar nichts zurück.

Eine Variable namens `FktPointer` vom Typ Zeiger auf eine Funktion, die zwei `int`-Parameter erhält und einen `int`-Zeiger zurückgibt, wird folgendermaßen definiert:

- ☐ `int **(FktPointer(int, int))`
- ☒ `int * (*FktPoiner)(int, int)`
- ☐ `int (*FktPointer*)(int, int)`

Wird ein Speicherbereich mit `calloc` reserviert, ist der Inhalt dieses Speicherbereiches

- ☐ undefiniert.
- ☐ mit 255 initialisiert.
- ☒ mit 0 initialisiert.

Kann ein dynamisch reservierter Speicherbereich nicht mehr freigegeben werden, weil z.B. kein Zeiger mehr auf diesen Speicherbereich verweist, wird dieses

- ☐ Computerleck genannt.
- ☒ Speicherleck genannt.
- ☐ Programmleck genannt.

Ein mit `#define` definiertes Makro wird wieder entfernt mit dem Befehl

- ☐ `#undefine`
- ☒ `#undef`
- ☐ `#delete`

Die `fclose`-Funktion

- ☐ gibt nichts zurück.
- ☒ gibt eine 0 zurück, wenn die Datei geschlossen werden konnte.
- ☐ gibt den Parameter zurück.

Mit `int * const Zeiger;` wird folgendes definiert:

- ☐ Ein unveränderbarer Zeiger auf eine unveränderbare Variable.
- ☐ Ein veränderbarer Zeiger auf eine unveränderbare Variable.
- ☒ Ein unveränderbarer Zeiger auf eine veränderbare Variable.

Aufgabe 3: Was gibt das folgende Programm aus? (25 / 25)
Schreiben Sie die Ergebnisse der vorgegebenen Ausdrücke sowie die Bildschirm-Ausgaben auf!

```
#include <stdio.h>

char Fkt1(char);
char Fkt2(char);
char Fkt3(char);

int main()
{
   char (*FktArray[3])(char) = {Fkt1, Fkt2, Fkt3};

   printf("%c", FktArray[ Fkt2( Fkt1( 5 ) )  ]('N') - 2 );
   printf("%c", FktArray[ Fkt2( 32 ) + 1     ]('M')     );
   printf("%c", FktArray[ Fkt1( 0 )          ]('B')     );
   printf("%c", FktArray[ Fkt1( 1 )  - 4     ]('U') + 1 );
   printf("%c", FktArray[ Fkt2( 42 )         ]('R') + 2 );
   printf("%c", FktArray[ Fkt2( 12 ) + 3     ]('V')     );
   printf("%c", FktArray[ Fkt3( Fkt2( 52 ) ) ]('T') - 1 );
   printf("\n");
}

char Fkt1(char Wert)
{
   return (Wert + 1) * (Wert + 2);
}

char Fkt2(char Wert)
{
   return (Wert - 2) / 10 - 2;
}

char Fkt3(char Wert)
{
   return Wert - 1;
}
```

Zwischenergebnisse:

`Fkt2( Fkt1( 5 ) )` = 2

`FktArray [ Fkt2( Fkt1( 5 ) ) ]` zeigt auf Funktion Fkt3

`FktArray [ Fkt2( Fkt1( 5 ) ) ]('N') - 2` = 'K' ✓

`Fkt2( 32 ) + 1` = 2

`FktArray[ Fkt2( 32 ) + 1 ]` zeigt auf Funktion Fkt3

`FktArray[ Fkt2( 32 ) + 1 ]('M')` = 'L'

`Fkt1( 0 )` = 2

`FktArray[ Fkt1( 0 ) ]` zeigt auf Funktion Fkt3

`FktArray[ Fkt1( 0 ) ]('B')` = 'A'

`Fkt1( 1 ) - 4` = 2

`FktArray[ Fkt1( 1 ) - 4 ]` zeigt auf Funktion Fkt3

`FktArray[ Fkt1( 1 ) - 4 ]('U') + 1` = 'V'

`Fkt2( 42 )` = 2

`FktArray[ Fkt2( 42 ) ]` zeigt auf Funktion Fkt3

`FktArray[ Fkt2( 42 ) ]('R') + 2` = 'S'

`Fkt2( 12 ) + 3` = 2

`FktArray[ Fkt2( 12 ) + 3 ]` zeigt auf Funktion Fkt3

`FktArray[ Fkt2( 12 ) + 3 ]('V')` = 'U'

`Fkt3( Fkt2( 52) )` = 2

`FktArray[Fkt3( Fkt2( 52) ) ]` zeigt auf Funktion Fkt3

`FktArray[Fkt3( Fkt2( 52) ) ]('T') - 1` = 'R'

Bildschirm-Ausgabe des oben stehenden Programms:

KLAUSUR

Aufgabe 4: Schreiben Sie die zwei fehlenden Funktionen zum vorgegebenen Hauptprogramm.
Die Funktion `compareDate` soll zwei Datumsstrukturen miteinander vergleichen. Als Parameter werden Zeiger auf die beiden Datumsstrukturen übergeben. Als Ergebnis wird eine ganze Zahl zurückgegeben (0, wenn beiden Daten gleich sind, < 0, wenn das erste Datum kleiner als das zweite Datum ist und > 0, wenn das erste Datum größer als das zweite Datum ist).
Die Funktion `writeFile` soll alle Daten aus dem Datumsarray in eine Textdatei schreiben. Dazu soll zuerst die Anzahl der Daten und dann pro Zeile ein Datum geschrieben werden. Dabei soll jedes Datum im Format zweistelliger Tag, Punkt, Leerzeichen, ausgeschriebener Monat, Leerzeichen und das vierstellige Jahr geschrieben werden, z.B. 08. Februar 2008. Als Parameter werden der Dateiname, das Array mit den Daten, die Anzahl der Daten und ein Array mit den Monatsnamen übergeben. (25 / 25)

```
#include <stdio.h>

#define MAX 10

typedef struct
{  int Day;
   int Month;
   int Year;
} TDate;

void BubbleSort(TDate *, int, int (*Vergleich)(TDate *, TDate *));
void fclearBuffer(FILE *);
int readFile(char *, TDate *);

// Platz für Ihre Funktionsdeklarationen (2 Punkte):
   int compareDate(TDate * date1, TDate * date2);
   void writeFile(char * Filename, TDate * A, int Anz, char ** Monthnames);

int main()
{  TDate Array[MAX];
   char *Monthnames[12] = { "Januar"   , "Februar"  , "März"     ,
                            "April"    , "Mai"      , "Juni"     ,
                            "Juli"     , "August"   , "September",
                            "Oktober"  , "November" , "Dezember" };
   int CountDate;

   CountDate = readFile("datum.txt", Array);
   BubbleSort(Array, CountDate, compareDate);
   writeFile("datumsortiert.txt", Array, CountDate, Monthnames);
}

void BubbleSort(TDate *A, int Anz, int (*Vergl)(TDate *, TDate *))
{  int i, j;
   TDate Temp;

   for (i = 1; i < Anz; i++)
      for (j = Anz - 1; j >= 1; j--)
         if (Vergl(A + j, A + j - 1) < 0)
         {  Temp = *(A + j);
            *(A + j) = *(A + j - 1);
            *(A + j - 1) = Temp;
         }
}
```

```
void fclearBuffer(FILE *D)
{  char Dummy;

   do
      fscanf(D, "%c", &Dummy);
   while (Dummy != '\n');
}

int readFile(char *Filename, TDate *A)
{  FILE *D;
   int CountDate = 0;
   int i;

   D = fopen(Filename, "r");
   if (D != NULL)
   {
      fscanf(D, "%i", &CountDate);
      fclearBuffer(D);
      if (CountDate > MAX)
         CountDate = MAX;
      for (i = 0; i < CountDate; i++)
      {
         fscanf(D, "%i %i %i", &((A + i)->Day),
                               &((A + i)->Month),
                               &((A + i)->Year));
         fclearBuffer(D);
      }
      fclose(D);
   }
   return CountDate;
}
```

// Platz für Ihre Funktionsdefinition (23 Punkte):

6

```
int compareDate(TDate * date1, TDate * date2)
{
   int i;
   i = date1->Year - date2->Year;
   if(i)
      return i;
   else
   { i = date1->Month - date2->Month;
     if(i)
        return i;
     else
        return date1->Day - date2->Day;
   }
}
```

Funktionsdefinitionen für Aufgabe 4 (Fortsetzung): 17

```
void writeFile (char * Filename, TDat *A, int Anz,
                                    char **Monthnames);
{
    FILE * D = fopen(Filename, "w");
    if (!D)
    {   fprintf(stderr, "failed to open file :s \n", Filename);
        return;
    }
    else
    {   int i;
        fprintf( D, "%i\n", Anz);
        for ( i = 0; i < Anz; i++)
        {
            fprintf( D, "%2i. %s %4i\n",
                (A+i)->Day, Monthnames[(A+i)->Month)-1],
                                    (A+i)->Year);
        }
        fclose (D);
    }
```

Aufgabe 5: Schreiben Sie die zwei fehlenden Funktionen zum vorgegebenen Hauptprogramm. Die Funktion `showOrdner` soll alle Akten eines Ordners tabellarisch auf den Bildschirm ausgeben. Dazu soll erst der Titel mit Angabe der Schrank- und der Ordnernr. ausgegeben werden. Dann soll jede Akte mit Belegnr. und Betreff ausgegeben werden. Schrank und Ordner werden als Parameter übergeben.
Die Funktion `appendAkte` soll die Daten einer Akte (Belegnr. und Betreff) in einer verketteten Liste speichern (anhängen!). Anfang und Ende der Liste sind in der Struktur `Ordner` gespeichert. Der entsprechende Ordner ist Teil eines Ordner-Arrays, das wiederrum in der Struktur `Schrank` liegt. (25 / 25)

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>

#define MAXORDNER 100

typedef struct A
{
   int BelegNr;
   char *Betreff;
   struct A *Next;
} Akte;

typedef struct
{
   Akte *E; // Erster
   Akte *L; // Letzter;
} Ordner;

typedef struct
{
   int SchrankNr;
   Ordner OA[MAXORDNER]; // OA für OrdnerArray
} Schrank;

void resetSchrank(Schrank *S, int SNr);        // SNr: Schranknr.
void deleteAkte(Schrank *S, int ONr, int BNr); // ONr: Ordnernr.
                                               // BNr: Belegnr.

// Platz für Ihre Funktionsdeklarationen (2 Punkte):
void showOrdner(Schrank *S, int O);
void appendAkte(Schrank *S, int o, int bn, char *bet);

int main()
{
   Schrank S1, S2;
   int i;

   resetSchrank(&S1, 1); // alle Zeiger in den Ordnern
   resetSchrank(&S2, 2); // auf NULL setzen

   // Akten im ersten Ordner von Schrank S1 ablegen:
   appendAkte(&S1, 1, 11, "Angebot");   // Belegnr. 11: Angebot
   appendAkte(&S1, 1, 17, "Rechnung");  // Belegnr. 17: Rechnung
```

2 ✓

```
   // Akten im 35. Ordner von Schrank S2 ablegen:
   appendAkte(&S2, 35, 15, "Angebot");  // Belegnr. 15: Angebot
   appendAkte(&S2, 35, 33, "Rechnung"); // Belegnr. 33: Rechnung
   appendAkte(&S2, 35, 57, "Mahnung");  // Belegnr. 57: Mahnung

   showOrdner(&S1, 1);  // alle Akten vom  1.Ordner von Schrank 1 zeigen
   showOrdner(&S2, 35); // alle Akten vom 35.Ordner von Schrank 2 zeigen

   deleteAkte(&S1, 1, 11);  // lösche Akte (Belegnr. 11) aus  1. Ordner
   deleteAkte(&S1, 1, 17);  // lösche Akte (Belegnr. 17) aus  1. Ordner
   deleteAkte(&S2, 35, 15); // lösche Akte (Belegnr. 15) aus 35. Ordner
   deleteAkte(&S2, 35, 33); // lösche Akte (Belegnr. 33) aus 35. Ordner
   deleteAkte(&S2, 35, 57); // lösche Akte (Belegnr. 57) aus 35. Ordner
}


void resetSchrank(Schrank *S, int SNr) // SNr: Schranknr.
{
   S->SchrankNr = SNr;
   for (int i = 0; i < MAXORDNER; i++)
      S->OA[i].E = S->OA[i].L = NULL;
}


void deleteAkte(Schrank *S, int ONr, int BNr)
{
   Akte *Temp = NULL, *Voriger = NULL;

   if ((S != NULL) && (ONr >= 1) && (ONr <= MAXORDNER))
   {
      if (S->OA[ONr - 1].First == NULL)
         return;
      if (S->OA[ONr - 1].First->BelegNr == BNr)
      {
         Temp = S->OA[ONr - 1].First;
         if (S->OA[ONr - 1].First == S->OA[ONr - 1].Last)
            S->OA[ONr - 1].Last = NULL;
         S->OA[ONr - 1].First = S->OA[ONr - 1].First->Next;
         free(Temp);
         return;
      }
      Voriger = S->OA[ONr - 1].First;
      Temp = Voriger->Next;
      while (Temp != NULL)
      {
         if (Temp->BelegNr == BNr)
         {
            Voriger->Next = Temp->Next;
            free(Temp);
            return;
         }
         Voriger = Temp;
         Temp = Temp->Next;
      }
   }
}

// Platz für Ihre Funktionsdefinitionen auf der nächsten Seite
```

Funktionsdefinitionen für Aufgabe 5 (23 Punkte):

8

```
void showOrdner( Schrank *s, int o)
{   Akte *a;
    printf("Schrank %4i: Ordner %4i\n", s->SchrankNr, o);
    a = s->OA[o-1].E;
    while(a)
    {
        printf("Belegnr. %3i: %.s\n", a->BelegNr, a->Betreff);
        a = a->Next;
    }
}
```

15 ✓.

```
void appendAkte( Schrank *s, int o, int bn, char * bet)
{
    Akte * a = malloc(sizeof(Akte));
    Akte * aptr = s->OA[o-1].E;
    if (!aptr)
        s->OA[o-1].E = s->OA[o-1].L = a;
    else if (aptr->Next != NULL)   // falls .E == .L
    {   while( aptr->Next != s->OA[o-1].L)
            aptr = aptr->Next;
        aptr->Next = a;
        s->OA[o-1].L = a;
    }
    a->Next = NULL;
    a->BelegNr = bn;
    a->Betreff = malloc(sizeof(char)*strlen(bet));
    strcpy(bet, a->Betreff);
}
```

✓