Dipl.Phys. Gerald Kempfer

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

Klausur zur Lehrveranstaltung "Algorithmen und Datenstrukturen" 21. Juli 2023

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 14.15 Uhr und endet – ebenfalls pünktlich – um 15.45 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, oben links mit Ihrer Matrikelnummer.

Viel Erfolg!

Name: [redacted]

Matrikelnummer: [redacted] letzter Prüfungsversuch: ☐

Bewertung:

Aufgabe	Mögliche Punktzahl	Erreichte Punktzahl
1	10	10
2	15	14
3	25	25
4	25	25
5	25	23
Summe	**100**	97

Note Klausur 1,0

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

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

taipedef struct x                    typedef
{
   int value;
   struct *next;                     struct x * next;
} Structure;

int main()                                        malloc
{
   struct x *tmp = (Structure *) memalloc(sizeof(Structure));
   Structure *start = tmp;
   FILE file;                        FILE * file;
   int i = 0;

   // Liste erzeugen
   for (; i < 10;)
   {
      tmp->value = ++i;
      if (i < 10)                                 sizeof
         tmp->next = malloc(seizeof(Structure));
      else
         tmp->next = NULL;
      tmp = tmp->nekst;                           next
   }

   // Liste in Datei schreiben
   tmp = start;
   file = fopen("zahlen.txt", "wt");
   if (file)
   {
      while (tmp)
      {
         fprintf(file, "%i\n", tmp->value);
         tmp = tmp.next;                          tmp->next
      }
      fileclose(file);                            fclose (file);
   }

   // Liste löschen
   tmp = start;
   while (temp)                                   while (tmp)
   {
      start = start->next;
      free((void *) tmp);
      tmp = start;
   }

   retörn 0;                                      return 0;
}
```

Aufgabe 2: Multiple-Choice-Fragen. (14 / 15)
Kreuzen Sie an, wie der Satz richtig heißen muss. Es gibt immer nur eine richtige Antwort.

Reservierte Speicherbereiche werden freigegeben mit der Funktion
- ☐ `erase`
- ☐ `remove`
- ☒ `free`

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

Die `fclose`-Funktion
- ☐ gibt nichts zurück.
- ☐ gibt immer eine 1 zurück.
- ☒ gibt eine 0 zurück, wenn die Datei geschlossen werden konnte.

Wird ein Speicherbereich mit `malloc` reserviert, ist der Inhalt dieses Speicherbereiches
- ☐ mit 0 initialisiert.
- ☐ mit 1 initialisiert.
- ☒ undefiniert.

Wenn mit der `fopen`-Funktion eine Datei nicht geöffnet werden kann, gibt die Funktion
- ☐ gar nichts zurück.
- ☐ den Wert `EOF` zurück.
- ☒ einen `NULL`-Zeiger zurück.

Mit dem Schlüsselwort `struct` wird eine
- ☐ Programmstruktur definiert.
- ☒ Datenstruktur definiert.
- ☐ Computerstruktur definiert.

Mit `const float * Zeiger;` wird folgendes definiert:
- ☐ Ein unveränderbarer Zeiger auf eine unveränderbare Variable.
- ☐ Ein unveränderbarer Zeiger auf eine veränderbare Variable.
- ☒ Ein veränderbarer Zeiger auf eine unveränderbare Variable.

Eine Variable namens `FktPointer` vom Typ Zeiger auf eine Funktion, die einen `int`-Zeiger-Parameter erhält und einen `char`-Zeiger zurückgibt, wird folgendermaßen definiert:
- ☒ `char * (*FktPointer)(int *);`
- ☐ `char * *FktPointer ( int *);`
- ☐ `char * (*FktPointer ( int *));`

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

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

Mit dem Präprozessorbefehl `#define` wird eine

- ☐ eine Variable definiert.
- ☒ eine Konstante definiert.
- ☐ eine Funktion definiert.

Eine Struktur darf

- ☐ keine Unterstruktur enthalten.
- ☐ nur eine Unterstruktur enthalten.
- ☒ beliebig viele Unterstrukturen enthalten.

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

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

Eine Variable vom Typ `unsigned float *` belegt

- ☐ weniger Speicher als eine Variable vom Typ `unsigned float **`.
- ☐ mehr Speicher als eine Variable vom Typ `unsigned float **`.
- ☒ genau so viel Speicher wie eine Variable vom Typ `unsigned int **`.

Eine doppelt-verkettete Liste heißt so, weil

- ☐ in den Listenelementen jeder Zeiger doppelt vorhanden sein muss.
- ☒ die Listenelemente mit zwei Zeigern miteinander verbunden sind.
- ☐ die Liste doppelt so viel Speicher belegt.

Eine andere Schreibweise für `ptr->Fld` (`ptr` ist ein Zeiger auf eine Struktur, die das Feld `Fld` beinhaltet) ist

- ☐ `(ptr).Fld`
- ☒ `*(ptr.Fld)`
- ☒ `(*ptr).Fld`

Aufgabe 3: Was gibt das folgende Programm aus? (25 / 25)
Schreiben Sie die Ergebnisse der vorgegebenen Ausdrücke (Zwischenergebnisse) sowie die Bildschirm-Ausgabe des Programms auf das folgende Blatt! Die Zwischenergebnisse für die erste printf-Zeile werden als Beispiel vorgegeben.

(1 Punkt je Zwischenergebnis + 4 Punkte für das Endergebnis)

```
#include <stdio.h>

typedef unsigned char Zeichen;

Zeichen Halbiere(Zeichen);
Zeichen Verdopple(Zeichen);
Zeichen Minus1(Zeichen);
Zeichen Plus1(Zeichen);

int main()
{
   Zeichen (*FktArray[4])(Zeichen) = {Halbiere,
                                      Verdopple,
                                      Minus1,
                                      Plus1};

   printf("%c", FktArray[ Minus1( Plus1( 3 ) )  ]('Q') + 1 );
   printf("%c", FktArray[ Verdopple( 2 ) - 1    ]('d')     );
   printf("%c", FktArray[ Plus1( 1 )            ]('p') - 2 );
   printf("%c", FktArray[ Minus1( 2 ) + 2       ]('g') - 3 );
   printf("%c", FktArray[ Verdopple( 3 ) - 4    ]('r') + 2 );
   printf("%c", FktArray[ Halbiere( 5 ) + 1     ]('p') + 3 );
   printf("%c", FktArray[ Verdopple( 2 ) - 2    ]('b') + 4 );
   printf("%c", FktArray[ Minus1( Minus1( 5 ) ) ]('t') - 3 );

   return 0;
}

Zeichen Halbiere(Zeichen c)
{  return (c / 2);  }

Zeichen Verdopple(Zeichen c)
{  return (c * 2);  }

Zeichen Minus1(Zeichen c)
{  return (c - 1);  }

Zeichen Plus1(Zeichen c)
{  return (c + 1);  }
```

Zwischenergebnisse:

`Minus1( Plus1( 3 ) )`	=	*3*
`FktArray[ Minus1( Plus1( 1 ) ) ]` zeigt auf Funktion	=	*Plus1*
`FktArray[ Minus1( Plus1( 1 ) ) ]('Q') + 1`	=	*S*

```
Verdopple( 2 ) - 1                                  = 3
```
FktArray[Verdopple(2) - 1] zeigt auf Funktion = Plus1

```
FktArray[ Verdopple( 2 ) - 1 ]('d')                 = e
```
d ≙ 100

```
Plus1( 1 )                                          = 2
```
FktArray[Plus1(1)] zeigt auf Funktion = Minus1

```
FktArray[ Plus1( 1 ) ]('p') - 2                     = m
```

```
Minus1( 2 ) + 2                                     = 3
```
FktArray[Minus1(2) + 2] zeigt auf Funktion = Plus1

```
FktArray[ Minus1( 2 ) + 2 ]('g') - 3                = e
```

```
Verdopple( 3 ) - 4                                  = 2
```
FktArray[Verdopple(3) - 4] zeigt auf Funktion = Minus1

```
FktArray[ Verdopple( 3 ) - 4 ]('r') + 2             = s
```

```
Halbiere( 5 ) + 1                                   = 3
```
FktArray[Halbiere(5) + 1] zeigt auf Funktion = Plus1

```
FktArray[ Halbiere( 5 ) + 1 ]('p') + 3              = t
```

```
Verdopple( 2 ) - 2                                  = 2
```
FktArray[Verdopple(2) - 4] zeigt auf Funktion = Minus1

```
FktArray[ Verdopple( 2 ) - 4 ]('b') + 4             = e
```

```
Minus1( Minus1( 5 ) )                               = 3
```
FktArray[Minus1(Minus1(5))] zeigt auf Funktion = Plus1

```
FktArray[ Minus1( Minus1( 5 ) ) ]('t') - 3 )        = r
```

Bildschirm-Ausgabe des oben stehenden Programms:
(Beachten Sie dabei auch die Formatierungsangaben sowie die Groß-/Kleinschreibung!)

Semester

Aufgabe 4: Schreiben Sie die zwei fehlenden Funktionen zum vorgegebenen Hauptprogramm.
Die Funktion `createPerson` soll einen Zeiger auf eine neue Person zurückgeben. Als Parameter werden Name (Zeichenkette) sowie das Geburtsdatum in Form von Tag, Monat und Jahr (jeweils ganze Zahlen) übergeben. Zuerst muss Speicherplatz für die Person reserviert werden, anschließend muss Speicherplatz für den Namen der Person entsprechend der Namenslänge reserviert werden. Dann werden die Daten der Person (also die Parameter) in dem Personendatensatz gespeichert. Der `next`-Zeiger muss auf `NULL` gesetzt werden. Zurückgegeben wird ein Zeiger auf die Person. Wichtig: Es soll bei *jedem* Speicherplatz-Reservieren abgefragt werden, ob das Reservieren erfolgreich war.
Die Funktion `printPersons` soll die Personen entsprechend der unten angegebenen Vorgabe auf dem Bildschirm ausgeben, d.h. der Name wird linksbündig mit einer Breite von 20 Zeichen ausgegeben, dann kommt ein senkrechter Strich und dahinter kommt das Geburtsdatum (2stelliger Tag, Punkt, 2stelliger Monat, Punkt und 4stelliges Jahr; jeweils mit führenden Nullen). Es gibt weder Parameter noch Rückgabewert. (25 / 25)

strlen+1
strcpy

Vorgabe für die Ausgabe:

```
Geburtstagsliste:
Beate Bunte          | 30.01.1993
Alfons Albrecht      | 03.05.1991
Susi Sorglos         | 21.08.1989
Max Mustermann       | 17.11.1988

Geburtstagsliste:
Keine Geburtstagskinder eingetragen
```

Hauptprogramm:

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

typedef struct
{
   int day;
   int month;
   int year;
} sDate;

typedef struct sp
{
   char *name;
   sDate birthday;
   struct sp *next;
} sPerson;

sPerson *first = NULL,
        *last  = NULL;

int appendPerson(sPerson *);
sPerson *removeFirstPerson();
void freePerson(sPerson *);

// Platz für Ihre Funktionsdeklarationen
// 6 Punkte (3 Punkte pro Deklaration):
```

sPerson * createPerson (char * name, ~~int~~ int day, int month, int year);

void printPersons();

```
int main()
{
   appendPerson( createPerson( "Beate Bunte"    , 30,  1, 1993 ) );
   appendPerson( createPerson( "Alfons Albrecht",  3,  5, 1991 ) );
   appendPerson( createPerson( "Susi Sorglos"   , 21,  8, 1989 ) );
   appendPerson( createPerson( "Max Mustermann" , 17, 11, 1988 ) );

   printPersons();

   while (first)
      freePerson(removeFirstPerson());

   printPersons();

   return 0;
}

int appendPerson(sPerson *newPerson)
{
   if (newPerson == NULL)
      return 0;

   newPerson->next = NULL;

   if (first == NULL)
   {
      first = last = newPerson;
      return 1;
   }

   last = last->next = newPerson;
   return 1;
}

sPerson *removeFirstPerson()
{
   sPerson *tmp = first;

   if (first)
   {
      first = first->next;
      if (first == NULL)
         last = NULL;
   }

   return tmp;
}

void freePerson(sPerson *Person)
{
   free(Person->name);
   free(Person);
}

// Platz für Ihre Funktionsdefinitionen
// (createPerson: 11 Punkte, printPerson: 8 Punkte):
```

```
sPerson * createPerson (char * name, int day,
                        int month, int year) {
   sPerson * ret = (sPerson *)malloc (sizeof(sPerson));
   if (sPerson == NULL) return NULL;
   if (ret == NULL) return NULL;
   ret->name = (char *) malloc ((strlen(name)+1)*sizeof(char));
   if (ret->name == NULL) {return NULL;
      free(ret);
      return NULL;
   }
```

```
// Fortsetzung Funktionsdefinitionen Aufgabe 4:
    ret->day = day;          ret->birthday.day = day;
    ret->month = month;      ret->birthday.month = month;
    ret->year = year;        ret->birthday.year = year;
    ret->next = NULL;
    strcpy(ret->name,
    strcpy(ret->name, name);
    return ret;
}

void printPersons() {
    sPerson * tmp = first;
    while (tmp) {
    printf("Geburtstagsliste:\n");
    if (tmp == NULL) {
        printf("Keine Geburtstagskinder eingetragen\n");
        return;
    }
    while (tmp)
    while (tmp != NULL) {
        printf("%20s | %02d.%02d.%04d\n",
            tmp->name,
            tmp->day,   tmp->birthday.day,
            tmp->month, tmp->birthday.month,
            tmp->year); tmp->birthday.year);
        tmp = tmp->next;
    }
    printf("\n");
}
```

Aufgabe 5: Schreiben Sie die zwei fehlenden Funktionen zum vorgegebenen Hauptprogramm.
Die Funktion `compareMeasurements` soll zwei Daten miteinander vergleichen. Als Parameter werden Zeiger auf die zu vergleichenden Messungen übergeben. Als Ergebnis wird eine ganze Zahl zurückgegeben (0, wenn beiden Daten gleich sind, < 0, wenn die erste Uhrzeit kleiner als die zweite Uhrzeit ist und > 0, wenn die erste Uhrzeit größer als die zweite Uhrzeit ist). Beim Vergleich der Uhrzeit werden zuerst die Stunden verglichen, wenn dieses bei beiden Uhrzeiten identisch ist, dann werden die Minuten vergleichen und wenn diese auch gleich ist, werden noch die Sekunden verglichen.
Die Funktion `writeFile` soll alle Messdaten aus dem Array in eine Textdatei schreiben. Dazu soll zuerst die Anzahl der Daten und dann pro Zeile ein Datensatz geschrieben werden: Zuerst wird die Uhrzeit (zweistellige Stunde, Doppelpunkt, zweistellige Minuten, Doppelpunkt und zweistellige Sekunden), dann ein Doppelpunkt und Leerzeichen und schließlich die gemessene Temperatur mit 5 Zeichen Mindestbreite, 2 Nachkommastellen und einem angehangenen "°C".
D.h. eine Zeile in der neuen Datei sieht z.B. wie folgt aus:
`11:55:17: 31.28 °C`
Als Parameter werden der Dateiname, das Array mit den Daten und die Anzahl der Daten übergeben. (23 / 25)

0 1=2
-1 1<2
1 1>2

Hauptprogramm:

```
#include <stdio.h>

#define MAX 10

typedef struct
{
   short hour;
   short minute;
   short second;
} sTime;

typedef struct
{
   sTime time;
   float temperature;
} sMeasurement;

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

// Platz für Ihre Funktionsdeklarationen:
// 6 Punkte (3 Punkte pro Deklaration):
int compareMeasurements (sMeasurement * first, sMeasurement
                         * second);
void writeFile (char * fName, sMeasurement * array, int cnt);

int main()
{
   sMeasurement measurements[MAX];
   int countMeasurements = readFile("measurements.txt", measurements);

   BubbleSort(measurements, countMeasurements, compareMeasurements);
   writeFile("measurements_sorted.txt", measurements, countMeasurements);

   return 0;
}
```

```
void BubbleSort(sMeasurement *array, int count, int (*cmpfct)(sMeasurement *, sMeasurement *))
{
   int i, j;
   sMeasurement tmp;

   for (i = 1; i < count; i++)
      for (j = count - 1; j >= 1; j--)
         if (cmpfct(array + j, array + j - 1) < 0)
         {

            tmp = *(array + j);
            *(array + j) = *(array + j - 1);
            *(array + j - 1) = tmp;
         }
}

void fclearBuffer(FILE *d)
{
   char dummy;

   do
   {
      fscanf(d, "%c", &dummy);
      if (feof(d))
         break;
   } while (dummy != '\n');
}

int readFile(char *filename, sMeasurement *array)
{
   int i, count = 0;
   FILE *d = fopen(filename, "r");

   if (d != NULL)
   {
      fscanf(d, "%i", &count);
      fclearBuffer(d);
      if (count > MAX)
         count = MAX;
      for (i = 0; i < count; i++)
      {
         fscanf(d, "%hi %hi %hi %f", &((array + i)->time.hour   ),
                                     &((array + i)->time.minute ),
                                     &((array + i)->time.second ),
                                     &((array + i)->temperature ));
         fclearBuffer(d);
      }
      fclose(d);
   }

   return count;
}
// Platz für Ihre Funktionsdefinitionen (19 Punkte)
// (compareDate: 7 Punkte; writeFile: 12 Punkte):
```

```
int compareMeasurements (sMeasurement * first,
                         sMeasurement * second) {
   if (first->hour - second->hour) {
      return first->hour - second->hour;
   } else if (first->minute - second->minute) {
      return first->minute - second->minute;
   } else {
      return first->second - second->second;
   }
}
```

```
// Fortsetzung Funktionsdefinitionen Aufgabe 5:
int compareMeasurements (sMeasurement *first
                         sMeasurement *second) {
    sTime *a = first.time;
    sTime *b = second.time;
    if (a->hour - b->hour) {
        return a->hour - b->hour;
    } else if (a->minute - b->minute) {
        return a->minute - b->minute;
    } else {
        return a->second - b->second;
    }
}
void writeFile (char *fName, sMeasurement *array, int cnt) {
    FILE* file = fopen(fName, "w");
    if (file == NULL) {
        printf("Fehler beim Schreiben der Datei!");
        return;
    }
    fprintf(file, "Anzahl der Daten: %d\n", cnt);
    int i=0;
    for(i=0; i<cnt; i++) { sMeasurement *tmp = array+i;
        fprintf(file, "%02d:%02d:%02d: %5.2f °C\n",
                tmp->time.day, tmp->time.month, tmp->time.
                tmp->time.hour, tmp->time.minute, tmp->time.second,
                tmp->temperature);
    }
    fclose(file);
}
```

time ist kein Zeiger!

-2

✓