Repetitorium zur Prozeduralen Programmierung

1 Grundlagen

1.1. Welche Werte haben die Variablen a und b nach Ausführung des folgenden Programms:

```
int a = 5;
int b = -3;
a = b = a;
```

1.2. Finden Sie die Fehler in folgendem Programm:

```
float a = 0;
int c, i;
for {i = 0; i < 5; i++}
  a = a + i;
```

1.3. Welche Ausgabe produziert folgendes Programm:

```
int i = 5;
for (i = 0; i < 3; i--) {
  i -= i;
  printf("i = %3d", i);
}
```

1.4. Schreiben Sie eine Funktion `isprime`, welche für gegebenen integer $n > 0$ den Wert 1 ausgibt, falls n prim, und 0 sonst. Hinweis: 1 ist keine Primzahl.

1.5. Schreiben Sie eine Funktion `squaresum`, die $\sum_{i=1}^{n} i^2$ berechnet. Berechnen Sie mit Hilfe der Funktion `isprime` aus Aufgabe 4, wie viele der Quadratsummen für n zwischen 1 und 10000 prim sind. Welcher Datentyp sollte für n bzw. für die Funktionsrückgabe verwendet werden?

1.6. Was sollte in folgenden Zeilen verbessert werden für ein lauffähiges Programm?

```
int i;
double x, *y;
scanf("%d%f", i, &x, y);
printf('i = %d%d%lf', i, x, y);
```

1.7. Für die Zahlen von 1 bis 2500 soll in einem 50×50 Quadrat jeweils ein bestimmter `char` gedruckt werden, falls die entsprechende Zahl prim ist, und sonst ein anderer `char`. Gehen Sie davon aus, dass eine Funktion `isprime` aus Aufgabe 4 zur Verfügung steht.

Zusatz: Führen Sie einen weiteren Parameter N ein, so dass ein $N \times N$ Quadrat gedruckt wird.

1.8. Schreiben Sie ein Programm, welches für gegebene Werte N und D feststellt, wieviele der Zahlen von 1 bis N durch D teilbar sind.

1.9. Prüfen Sie für gegebenes N, für wieviele der Paare (a, b) für a und b zwischen 1 und N die Zahl $a^2 + b^2$ wiederum ein Quadrat ist.

1.10. Schreiben Sie ein Programm, das zunächst den Wert N und anschließend N Zahlen einliest. Die eingegeben Zahlen sollen aufsteigend sortiert ausgegeben werden.

2 Zeiger

2.1. Vervollständigen Sie das folgende Programm und führen Sie es aus.

```
#include <stdio.h>
int main() {
  char x = 'G';
  char *p, **q;
  p=___x;
  q=&p;

  printf("Der Wert von x ist:\t %___ \n", *___);
  printf("Die Adresse von x ist:\t %p \n\n", (void*)___);

  printf("Der Wert von p ist: \t %___ \n", *___);
  printf("Die Adresse von p ist: \t %___ \n", (void*)___);

  return 0;
}
```

Beantworten Sie folgende Verständnisfragen:

a) Was ist der Unterschied zwischen einem Zeiger und einer Adresse ?

b) Wozu dienen `&` und `*` ? Unterscheiden Sie dabei für `*`, ob das Symbol in einer Deklaration auftritt oder nicht.

c) Was bedeutet `%p` und wieso schreibt man `(void*)q` ?

d) Welche Datentypen haben x, p und q?

e) Worum handelt es sich bei `0x7fff297417c8` ? Was bedeutet dabei `0x` ? Wofür stehen die Buchstaben `f` und `c` ? Wie kann man die Zahl ins Dezimalsystem umrechnen ?

2.2. Was ist der Unterschied zwischen folgenden Definitionen? Welche Notationen sind dabei eher irreführend (eigentlich falsch) und welche halten Sie für am besten geeignet?

a) `int* zeiger1, zeiger2;`

b) `int *zeiger1, *zeiger2;`

c) `int* zeiger1, *zeiger2;`

d) `int *zeiger1; int *zeiger2;`

e) `int* zeiger1; int *zeiger2;`

Vier der angegeben fünf Definitionen sind semantisch äquivalent. Welche hat eine andere Bedeutung?

2.3. Vervollständigen Sie das folgende Programm:

```
#include <stdio.h>

int main() {
  char   c, *p_chr;
  int    x, *p_int;
  float  y, *p_flt;
  double z, *p_dbl;

  p_chr = __c;
  p_int = __x;
  p_flt = __y;
  p_dbl = __z;

  printf("Geben Sie ein Zeichen ein:          c = ");
  scanf("%__", p_chr);
  printf("Geben Sie eine ganze Zahl ein:      x = ");
  scanf("%d", _______);
  printf("Geben Sie eine Fließkommazahl ein:  y = ");
  scanf("%f", _______);
  printf("                  ... und noch eine:  z = ");
  scanf("%__",_______);

  printf("\n");
  printf(" Objekt | Bytes | Wert      \n");
  printf("--------+-------+----------\n");
  printf(" *p_chr |   %d  | %c \n", (int) sizeof(*p_chr), _______);
  printf(" *p_int |   %d  | %d \n", (int) sizeof(_______), _______);
  printf(" *p_flt |   %d  | %f \n", (int) sizeof(_______), _______);
  printf(" *p_dbl |   %d  | %f \n", (int) sizeof(_______), _______);
  printf("  p_chr |   %d  | %p \n", (int) sizeof(_______), (void*)_______);
  printf("  p_int |   %d  | %p \n", (int) sizeof(_______), (void*)_______);
  printf("  p_flt |   %d  | %p \n", (int) sizeof(_______), (void*)_______);
  printf("  p_dbl |   %d  | %p \n", (int) sizeof(_______), (void*)_______);

  return 0;
}
```

Warum haben beispielsweise `*p_chr` und `*p_int` unterschiedliche Byte-Längen während hingegen `p_chr` und `p_int` gleiche Byte-Längen haben?

2.4. Schreiben Sie ein Programm, das die Werte zweier integer Variablen x und y von der Tastatur einliest und dann eine Funktion `swap` aufruft, welche deren Werte mittels "call-by-reference" vertauscht. Abschließend sollen im Hauptprogramm die Werte von x und y ausgegeben werden.

2.5. Führen Sie das folgende Programm aus und erklären Sie die Ausgabe im Detail!

```
#include <stdio.h>
int main() {
  short int x = 333;
  char *p_chr;
  int i;
  p_chr = (char*) &x;
  printf("x = %d\n", x);
  printf("x = ");
  for(i = 32768; i > 0; i >>= 1) {
    printf("%d", (x & i) > 0);
  }
  printf("\n\n");
  printf("x_chr = %c \n",*p_chr);
  printf("x_chr = %d \n",*p_chr);
  printf("x_chr = ");
  for(i = 128; i > 0; i >>= 1) {
    printf("%d", ((*p_chr) &i) > 0);
  }
  return 0;
}
```

2.6. Führen Sie das folgende Programm aus:

```
#include <stdio.h>
int main() {
  int x=7, y=17, z=27;
  int *p1 = &x, *p2 = &y, *p3 = &z;

  printf("%p  %p  %p\n\n",(void*)p1,(void*)p2,(void*)p3);

  printf("%p  %d  %d\n",  (void*)(p1+1), *(p1+1), *p1+1);
  printf("%p  %d  %d\n\n",(void*)(p1+2), *(p1+2), *p1+2);

  printf("%p  %d  %d\n",  (void*)(p1-1), *(p1-1), *p1-1);
  printf("%p  %d  %d\n\n",(void*)(p1-2), *(p1-2), *p1-2);

  return 0;
}
```

a) Erklären Sie die Ausgabe im Detail!

b) Welche Ausgaben ändern sich bei wiederholter Programmausführung und welche bleiben gleich? Welche der letzteren könnten auf einem anderen Rechner zu anderen Ergebnissen führen und welche nicht? Geben Sie jeweils genaue Gründe an!

c) Bei welchen Anweisungen handelt es sich um sogenannte “Zeigerarithmetik”?

2.7. Schreiben Sie ein Programm, das drei `double`-Variablen `x,y,z` von der Tastatur einliest und deren Adressen durch call-by-reference an eine Funktion `median` übergibt. Diese soll das arithmetische Mittel $a := \frac{x+y+z}{3}$, das geometrische Mittel $b := \sqrt[3]{x \cdot y \cdot z}$ und das harmonische Mittel $c := \frac{3}{\frac{1}{x}+\frac{1}{y}+\frac{1}{z}}$ berechnen. Die Ergebnisse a, b, c sollen in dieser Reihenfolge in `x,y,z` gespeichert und im Hauptprogramm ausgegeben werden.

Hinweise: Sie können die `math.h`-Funktion `cbrt(z)` zur Berechnung von $\sqrt[3]{z}$ nutzen (C99).

Für positive `x,y,z` gilt stets $a \geq b \geq c$.

2.8. Vervollständigen Sie das folgende Programm:

```
#include <stdio.h>
int main() {
  char ch0, *ch1, **ch2, ***ch3, ****ch4;
  ch1 = &ch0;
  ch2 = &ch1;
  ch3 = &ch2;
  ch4 = &ch3;
  printf("Geben Sie ein Zeichen ein ch = ");
  scanf("%c",______ch3);
  printf("\n Ihre Eingabe war ch = %c \n",_____ch4);
  return 0;
}
```

1. Welche Datentypen haben `ch0,ch1,ch2,ch3,ch4` ?
2. Ersetzen Sie in den Zeilen 10 und 11, `(ch3,ch4)` durch andere Kombinationen, beispielsweise durch `(ch1,ch3)`, `(ch0,ch2)` und `(ch4,ch1)` !

2.9. Schreiben Sie ein Programm, das drei Fließkommazahlen von der Tastatur einliest, diese an eine Funktion `maximum` übergibt, welche ihr Maximum bestimmt und einen Zeiger auf dieses an das aufrufende Hauptprogramm zurück gibt, wo es ausgegeben wird. Überlegen Sie sich dabei zunächst die Signatur der Funktion `maximun`.

2.10. Vervollständigen Sie das folgende Programm:

```
#include <stdio.h>
int main() {
  char ch = 'W';
  int x = 7;
  float y = 3.141;
  void *p;

  p = &ch;
  printf("ch = %c\n",________p);
  p = &x;
  printf("x  = %d\n",________p);
  p = &y;
  printf("y  = %f\n",________p);
  return 0;
}
```

a) Welchen Datentyp hat p?

b) Warum genügt es in den Zeilen 9, 11 und 13 nicht einfach nur p zu schreiben ?

c) Was bedeutet "typecasting" ?

d) Was ist der Unterschied zwischen explizitem und implizitem typecasting ?

3 Arrays

3.1. a) Schreiben Sie eine Funktion `scalar_mult`, welche die Skalarmultiplikation $w = a \cdot v$ für gegebenen Vektor $v \in \mathbb{R}^n$ und gegebene reelle Zahl $a \in \mathbb{R}$ implementiert. Überlegen Sie dabei zunächst geeignete Datentypen für die Funktionsparameter.

b) Schreiben Sie analog zu a) eine Funktion `vec_add`, welche die Vektorsumme $w = u+v$ für gegebene Vektoren $u, v \in \mathbb{R}^n$ berechnet.

c) Schreiben Sie ein Hauptprogramm, das eine Dimension n, zwei Vektoren $u, v \in \mathbb{R}^n$ und ein $a \in \mathbb{R}$ von der Tastatur einliest und dann sowohl $a \cdot v$ als auch $u + v$ mit den Funktionen aus a) und b) berechnet und beide Ergebnisse auf dem Bildschirm ausgibt. Die nötigen Arrays der variablen Länge n legen Sie dynamisch mit `malloc` an.

3.2. a) Schreiben Sie eine Funktion `scalar_prod`, welche das Skalarprodukt $a := u^T v = \sum_{i=1}^{n} u_i v_i$ zweier gegebener reeller Vektoren $u, v \in \mathbb{R}^n$ der Länge n berechnet und als Funktionswert zurückgibt. Überlegen Sie sich zunächst eine geeignete Signatur der Funktion. Die Dimension n soll dabei als Parameter übergeben werden.

b) Benutzen Sie die Funktion `scalar_prod` aus a), um eine Funktion `two_norm` zu implementieren, welche die euklidische Norm $\|v\|_2 := \sqrt{v^t v} = \sqrt{\sum_{i=1}^{n} v_i^2}$ eines gegebenen Vektors $v \in \mathbb{R}^n$ berechnet und zurück gibt. (Binden Sie `math.h` zur Verwendung von `sqrt` ein.)

c) Schreiben Sie ein Hauptprogramm, das eine Dimension n und zwei Vektoren $u, v \in \mathbb{R}^n$ von der Tastatur einliest und dann sowohl $u^t v$ als auch $\|v\|_2$ berechnet und ausgibt.

d) Erweitern Sie das Hauptprogramm abschließend um die Berechnung des Winkels α (in Grad), den die beiden Vektoren u, v einschließen. Testen Sie ihr Programm für $n = 2$ mit $u = (\sqrt{3}, 1)$ und $v = (1/2, \sqrt{3}/2)$. Ergebnis: $\alpha = 30°$.

Hinweise: $\cos(\alpha \cdot \frac{\pi}{180}) = \dfrac{u^T v}{\|u\|_2 \|v\|_2}$, `math.h`, `acos()`.

3.3. a) Schreiben Sie ein Programm, das `N=5` ganze Zahlen von der Tastatur einliest und sie aufsteigend sortiert wieder ausgibt. Die Sortierung soll in einer Funktion `sort` erfolgen, die beispielsweise einen Bubblesort-Algorithmus implementiert. Definieren Sie dabei `N` als Makro-Konstante und verwenden Sie ausschließlich diese bei der anschließenden Programmierung. (Warum?)

b) Ändern Sie Ihr Programm so, dass beliebig viele zu sortierende Zahlen eingelesen werden können, indem als erstes deren Anzahl von der Tastatur eingelesen wird.

3.4. Führen Sie das folgende Programm für verschiedene Werte von `n` mehrfach aus und beantworten Sie anschließende Fragen.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  int n;
  short *d;
  short a[5]= {4,7,-5};
  printf(" sizeof(a) = %d",(int)sizeof(a)/(int)sizeof(*a));
  printf("\n a[4] = %hd",a[4]);
  printf("\n a[5] = %hd",a[5]);
  short b[n];
  printf("\n sizeof(b) = %d",(int)sizeof(b)/(int)sizeof(*b));
  printf("\n n = ");
  scanf("%d",&n);
  short c[n];
  printf(" sizeof(c) = %d \n",(int)sizeof(c)/(int)sizeof(*c));
  d=(short*)malloc(n*sizeof(short));
  printf(" sizeof(d) = %d \n",(int)sizeof(d)/(int)sizeof(*d));
  return 0;
}
```

a) Wozu dient der `sizeof`-Operator und was wird durch den Term `(int)sizeof(a)/(int)sizeof(*a)` berechnet?

b) Hat `a[4]` bei mehrfacher Ausführung immer denselben Wert? Wenn ja, welchen? Ist das Ergebnis rechnerunabhängig?

c) Hat `a[5]` bei mehrfacher Ausführung immer denselben Wert? Wenn ja, welchen? Ist das Ergebnis rechnerunabhängig?

d) Warum kann sich `sizeof(b)` bei mehrfacher Ausführung ändern, aber `sizeof(c)` nicht?

e) Warum unterscheiden sich `sizeof(c)` und `sizeof(d)`? Erklären Sie beide ausgegebenen Werte im Detail. Welcher dieser Werte ist rechnerunabhängig und welcher nicht?

3.5. a) Wie werden (im Allgemeinen) Arrays an Funktionen übergeben, mit "call-by-value" oder "call-by-reference" ?

b) Im folgenden Programm sollen mit der Funktion `increment` die Inhalte des `int`-Arrays `a` des Hauptprogramms um 1 erhöht werden. Vervollständigen Sie den Funktionskopf und überprüfen Sie Ihre Antwort auf Frage a).

```
#include <stdio.h>
__________ increment(____________) {
  b[0]++;
  b[1]++;
  return;
}
int main() {
  int a[2]= {4,7};
  printf("a[0] = %d , a[1] = %d \n",a[0],a[1]);
  increment(a);
  printf("a[0] = %d , a[1] = %d \n",a[0],a[1]);
  return 0;
}
```

c) Das folgende Programm bindet das Array `a` als Komponente in eine Struktur ein. Führen Sie das Programm aus und erklären Sie die neue, von b) abweichende Ausgabe. Wie erfolgt jetzt die Übergabe des Arrays an die Funktion `increment`, mit "call-by-value" oder "call-by-reference"?

```
#include <stdio.h>
typedef struct {
  int c[2];
} arr;
void increment(arr b) {
  b.c[0]++;
  b.c[1]++;
  return;
}
int main() {
  arr a= {{4,7}};
  printf("a[0] = %d , a[1] = %d \n",a.c[0],a.c[1]);
  increment(a);
  printf("a[0] = %d , a[1] = %d \n",a.c[0],a.c[1]);
  return 0;
}
```

Ändern Sie den Funktionsaufruf und die Funktion so ab, dass die gleiche Ausgabe wie in b) erfolgt.

d) Ein Programmieranfänger möchte eine Funktion `addiere` schreiben, mit der zwei dreidimensionale Vektoren $a, b \in \mathbb{R}^3$ addiert werden. Der Ergebnisvektor $c := a + b$ soll von der Funktion zurückgegeben werden. Er überlegt sich folgenden Funktionskopf:

```
double c[3] addiere(double a[3],double b[3])
```

Was ist daran alles falsch. Schlagen Sie einen alternativen Funktionskopf vor und implementieren Sie die Funktion.

3.6. In einem Programm wird `float a[5],b[9],*p=NULL;` definiert. Welche der folgenden Anweisungen sind syntaktisch falsch und welche sind semantisch falsch. Begründen Sie Ihre Entscheidungen. Erläutern Sie auch, was bei den syntaktisch richtigen Anweisungen passiert.

a) `scanf("%f",a+3);`

b) `scanf("%f",&a);`

c) `scanf("%f",&a[4]);`

d) `a = b;`

e) `a = p;`

f) `p = a;`

g) `&a[3] = p;`

h) `p = b+5;`

i) `a[4] = b[9];`

4 Strings

4.1. Schreiben Sie eine `void`-Funktion, die als Parameter ein `char`-Array und einen Integer-Wert, der die Länge des `char`-Arrays angibt, besitzt. Die Funktion soll Inhalt und Adresse jedes Array-Elementes übersichtlich ausgeben. Testen Sie die Funktion mit einem Hauptprogramm, das einen String (max. 100 Zeichen) von der Tastatur einliest und diesen zusammen mit seiner Länge an die Funktion übergibt. Erklären Sie die Abstände der einzelnen ausgegebenen Adressen.

Hinweis: `string.h`, `strlen`

4.2. Führen Sie folgendes Programm aus und erläutern Sie die Ausgabe im Detail.

```
#include <stdio.h>
int main(void) {
  char str[50] = "hello\0 worl\bd";
  printf("\n %s ",str);
  printf("%s \n",str+str[4]-*str);
  return 0;
}
```

4.3. Schreiben Sie ein "Suche-und-Ersetze"-Programm, das einen String `str` der Länge 100 und zwei Zeichen `ch_1` und `ch_2` von der Tastatur einliest und das alle Vorkommen von `ch_1` in `str` durch `ch_2` ersetzt. Benutzen Sie eine while-Schleife, um die Buchstaben des Strings einzeln (von links nach rechts) durchzugehen. Abschließend soll der neue String auf dem Bildschirm ausgegeben werden.

Beispiel: Wenn `str` das Wort "Kernkraftwerk" enthält und `ch_1='e'` und `ch_2='o'` sind, dann lautet das Ergebnis "Kornkraftwork".

4.4. Schreiben Sie ein Programm, das einen String (max. 20 Zeichen) von der Tastatur einliest und alle in ihm versteckten Ziffern hintereinander ausgibt. Abschließend soll auch die Summe all dieser Ziffern ausgegeben werden.

Beispiele: Der String `"abc-0.19xy4"` enthält die Ziffern `0194`, deren Summe `14` ist.

4.5. Schreiben Sie ein Programm, das mit (beliebig vielen) rationalen Zahlen als Programmparameter aufgerufen werden kann und deren Minimum berechnet und ausgibt.

4.6. Beantworten Sie folgende Fragen:

a) Was ist ein String?

b) Was ist der Unterschied zwischen einem `char`-Array und einem String.

c) Was bedeutet `'\0'` ? Welchen ASCII-Code hat dieses Zeichen?

d) Ist folgende Anweisung syntaktisch und semantisch richtig? `char str[3]="ABC";`

e) Das folgende Programm enthält genau einen semantischen Fehler. Führen Sie es aus. Kommentieren Sie anschließend Zeile 4 aus und starten Sie das Programm erneut. Nehmen Sie dann die Zeile wieder hinzu. Erläutern Sie das Problem und beheben Sie es.

```
#include <stdio.h>
int main(void) {
  char *s = "Luft";
  s[1] = 'o';
  printf("\n%s\n", s);
  return 0;
}
```

f) Wie heißt die Standard-Header-Datei mit Funktionen zur String-Verarbeitung? Nennen Sie (mindestens) drei Funktionen dieser Header-Datei und schreiben Sie ein kurzes Programm, das diese Funktionen (sinnvoll) aufruft.

g) Wie kann man den Zahlwert des numerischen Strings `"-12.54"` einer `double`-Variablen x zuweisen? Geben Sie eine entsprechende Codezeile an.

5 Dynamische Speicherverwaltung

5.1. Beantworten Sie folgende Fragen:

a) Was versteht man unter dynamischer Speicherreservierung?

b) Welche Funktionen stellt die Standardbibliothek hierfür zur Verfügung und in welcher Header-Datei befinden sich diese?

c) Wozu dient die `sizeof`-Funktion im Zusammenhang mit dynamischer Speicherreservierung?

d) Wie gibt man dynamisch reservierten Speicher wieder frei.

e) Was geschieht mit nicht explizit freigegebenem dynamisch alloziertem Speicher? Welche Probleme mit welchen Folgen können entstehen?

f) Wie erkennt man eine fehlgeschlagene dynamische Speicherallokation und wie sollte ein Programm darauf reagieren?

5.2. Schreiben Sie ein Programm, das eine natürliche Zahl n von der Tatstatur einliest, anschließend dynamisch Speicher für ein `double`-Array A der Länge n anlegt und dann die Werte $A[i]$, $i = 0, .., n$, von der Tastatur einliest. Schreiben Sie nun eine Funktion `reflect`, welche die Array-Werte "spiegelt", d.h. die Werte `A[i]` und `A[n-i]` sollen vertauscht werden. Überlegen Sie sich dabei zunächst eine geeignete Signatur für die Funktion. Rufen Sie diese dann im Hauptprogramm auf und geben Sie das Array danach zur Kontrolle aus. Abschließend soll der Speicherbereich des Arrays wieder freigegeben werden.

Beispiel: Das Array `A={2,-4,7,8,-1}` wird zu `A'={-1,8,7,-4,2}` gespiegelt.

5.3. Definieren Sie eine Struktur `person` mit den zwei Komponenten `surname` (Nachname) und `age` (Alter). Überlegen Sie sich dabei zunächst geeignete Datentypen für die beiden Komponenten. Reservieren Sie anschließend dynamisch Speicher für eine einzige Person und legen Sie die Adresse dieses Speicherbereichs in einem Zeiger ab. Fangen Sie die Situation ab, dass kein Speicher reserviert werden konnte. Lesen Sie nun den Nachnamen und das Alter von der Tastatur in die entsprechenden Komponenten ein und geben Sie diese direkt anschließend zur Kontrolle wieder aus. Benutzen Sie dabei die Notation "`... -> ...`", um auf die Komponenten zuzugreifen. Geben Sie abschließend den Speicher wieder frei.

5.4. In dieser Aufgabe soll dynamisch Speicher für ein String-Array erzeugt werden, das variabel viele Strings speichern kann, die ihrerseits eine variable Länge ≤ 100 Zeichen haben können. Dabei soll kein Speicher nutzlos verschwendet werden.

Lesen Sie zunächst eine ganze Zahl n für die Anzahl der einzugebenden Strings ein. Erzeugen Sie dann dynamisch ein entsprechendes Array S zu deren Speicherung. Welchen Datentyp hat S? Lesen Sie anschließend die n Strings von der Tastatur ein. Bestimmen Sie dafür zunächst die jeweilige Stringlänge, reservieren Sie dann dynamisch entsprechend viel Speicher, kopieren Sie den eingelesen String in diesen Speicherbereich und legen Sie die Anfangsadresse in dem entsprechenden Feld von S ab. Geben Sie zur Kontrolle das String-Array auf dem Bildschirm aus und geben Sie abschließend den gesamten dynamisch reservierten Speicher wieder frei.

Hinweis: Die $n = 3$ Strings "Haus", "Hof", "As" haben insgesamt $4+3+2 = 9$ Buchstaben. Nach obigem Vorgehen sollten daher inklusive `\0`-Terminierungszeichen insgesamt auch nur $9 + n = 12$ Byte Speicher für `char`-Werte dynamisch reserviert werden.

5.5. Schreiben Sie eine Funktion `memory`, die berechnet, wieviel Gigabyte zusammenhängender Speicher auf Ihrem Rechner aktuell dynamisch reserviert werden kann. Schreiben Sie ein kurzes Hauptprogramm, das diese Funktion aufruft und das Ergebnis auf dem Bildschirm ausgibt.

Hinweise: `1 GB = 1024^3 Byte`. Das Ergebnis soll ganzzahlig sein, z.B.: `20 GB`.

6 Matrizen (mehrdimensionale Arrays)

6.1. Beantworten Sie folgende Fragen:

a) Was ist ein mehrdimensionales Array und wie wird es in C definiert?

b) Wie kann das zweidimensionale Array (die Matrix) `double A[2][3]` als Parameter an eine Funktion übergeben werden?

c) Wie lässt sich eine Matrix $A \in \mathbb{R}^{4,17}$ in einem eindimensionalen Array speichern? Wie lässt sich dann ein Makro `#define A(i,j) ...` vervollständigen, sodass eine Anweisung `A(3,7) = -11;` den Matrixeintrag $A_{3,7} := -11$ vornimmt. Testen Sie dies notfalls.

d) Welchen Vorteil bietet es, Array-Dimensionen als Makros zu definieren anstatt feste Werte einzusetzen?

e) Welche Möglichkeiten kennen Sie, eine $m \times n$-Matrix $A \in \mathbb{R}^{m,n}$ in C zu implementieren, deren Zeilen- und Spaltenzahl m und n erst zur Laufzeit bekannt sind.

6.2. a) Definieren Sie drei 2×2-Matrizen $A, B, C \in \mathbb{R}^{2,2}$. Lesen Sie A und B von der Tastatur ein. Schreiben Sie eine Funktion `mat_mult`, welche die Matrizen `A` und `B` miteinander multipliziert und das Ergebnis in `C` ablegt. Rufen Sie diese Funktion im Hauptprogramm auf und geben Sie `C` dort aus.

b) Verallgemeinern Sie Ihr Programm auf beliebige $(n \times n)$-Matrizen (Makrodefinition).

c) Ändern Sie Ihr Programm auf allgemeine Matrizen $A \in \mathbb{R}^{m,n}$, $B \in \mathbb{R}^{n,k}$ und $C \in \mathbb{R}^{m,k}$ ab. Dabei sollen m, n, k von der Tastatur eingelesen und A, B, C dynamisch erzeugt werden. Überlegen Sie sich ggf. geeignete Funktionen für folgende Teilaufgaben:

i) Speicher für eine Matrix dynamisch reservieren

ii) Matrix einlesen

iii) Matrix ausgeben

iv) dynamisch reservierten Speicher einer Matrix wieder freigeben

6.3. a) Schreiben Sie eine Funktion `tmat_alloc`, welche zu gegebenem $n \in \mathbb{N}$ Speicher für eine reelle obere Dreiecksmatrix $A \in \mathbb{R}^{n,n}$ reserviert und einen Zeiger auf diesen zurückgibt. Dabei soll kein Speicher ungenutzt verschwendet werden, d.h., es soll nur Speicher für die Einträge oberhalb der Diagonalen reserviert werden.

b) Schreiben Sie eine Funktion `tmat_read`, welche die Einträge einer gemäß a) erzeugten oberen Dreiecksmatrix von der Tatstatur einliest.

c) Schreiben Sie eine Funktion `tmat_mult`, die für gegebenes $n \in \mathbb{N}$ und gegebene obere Dreiecksmatrizen $A, B \in \mathbb{R}^{n,n}$ zunächst Speicher für eine dritte obere Dreiecksmatrix $C \in \mathbb{R}^{n,n}$ erzeugt, dann $C := AB$ setzt und den Zeiger auf C zurückgibt.

d) Schreiben Sie eine Funktion `tmat_print`, die eine obere Dreiecksmatrix übersichtlich auf dem Bildschirm ausgibt. Jede Matrixzeile soll dabei in einer neuen Bildschirmzeile erscheinen.

e) Schreiben Sie eine Funktion `tmat_free`, welche den gesamten Speicher einer oberen Dreiecksmatrix wieder freigibt.

f) Schreiben Sie nun mit Hilfe der Funktionen aus a) - e) ein Programm, das $n \in \mathbb{N}$ und zwei obere Dreiecksmatrizen $A, B \in \mathbb{R}^{n,n}$ von der Tastatur einliest, $C := AB$ berechnet und ausgibt.

Überlegen Sie sich für die Funktionen in a) - e) zunächst geeignete Signaturen.

6.4. Jeder der folgenden Anweisungsblöcke enthält höchstens einen syntaktischen oder semantischen Fehler. Finden Sie diesen und begründen Sie Ihre Antwort. Geben Sie auch die Bedeutung der syntaktisch richtigen Anweisungen an.

a)
```
char S[3][20] = {"Rep.", "Prozedurale", "Programmierung"};
printf("%s %s %s", S[0], S[1], S[2]);
```

b)
```
short** s = malloc(3*sizeof(long*));
s[2] = malloc(4*sizeof(short));
s[2][3] = -7;
printf("\n %hi ", s[2][3]);
```

c)
```
double B[3] = {1, 2, 3};
double C[3] = {4, 5, 6};
double A[2][3] = {B, C};
```

d)
```
int A[1][3] = {{1, 2, 4}};
printf("\n %d, %d, %d",*A[0],*A[0]+1,*A[0]+2);
```

e)
```
float B[2][3];
printf("\n Größe der Matrix in Byte : %ld \n",sizeof(B));
```

7 Funktionen

7.1. Schreiben Sie ein einfaches Programm, welches eine Textdatei erzeugt und den über die Tastatur eingegeben Text hineinschreibt.

7.2. Schreiben Sie ein Programm, welches zwei Dateipfade zu unterschiedlichen Textdateien von der Tastatur einliest und anschließend den Text der zweiten Datei an den der ersten anfügt.

7.3. Nennen Sie drei verschiedene Funktionen, die man dazu verwenden kann einen String auf dem Bildschirm auszugeben. Schreiben Sie ein Programm, in welchem die von Ihnen aufgezählten Möglichkeiten genutzt werden, um den Satz "Ich bereite mich auf die Klausur vor" auszugeben. Worin unterscheiden sich die verwendeten Standardfunktionen zur Ausgabe von Zeichenketten?

7.4. Schreiben Sie ein Programm, welches die Ausbreitung einer Sinuswelle auf dem Bildschirm darstellt. Zur Vereinfachung der Aufgabe können Sie davon ausgehen, dass die Welle auf Ihrer Konsole in Richtung des unteren Fensterrandes verläuft (in negative y-Richtung).

7.5. Implementieren Sie eine Funktion, die einen Pfad zu einer Textdatei übergeben bekommt und für den enthaltenen Text die durchschnittliche Satzlänge (Wörter pro Satz) bestimmt.

7.6. Schreiben Sie ein Programm, welches Zahlenwerte in Form von Kommandozeilenargumenten übergeben bekommt und die Summe dieser auf dem Bildschirm ausgibt. Testen Sie Ihr Programm.

7.7. Implementieren Sie eine Funktion, die zum Berechnen einer dritten Seite eines rechtwinkligen Dreiecks dient. Die Funktion soll drei Argumente übergeben bekommen, für jede Seitenlänge genau ein Argument, wobei die Reihenfolge der Katheten und der Hypotenuse festgelegt sei. Die zu bestimmende Seitenlänge soll durch einen negative Wert des Eingabearguments angezeigt werden. Wie muss der Funktionskopf gestaltet werden, wenn der negative Eintrag durch den neu berechneten Wert ersetzt werden soll?

7.8. Realisieren Sie eine Funktion, welche ein Feld von Fließkommazahlen übergeben bekommt und ein Feld gleicher Länge zurückgibt. Die Elemente des Ausgabefeldes sollen mit den Sinuswerten zu den Einträgen des Eingangsfeldes gefüllt sein. Wie muss die Signatur der Funktion aussehen, wenn die Einträge des übergebenen Feldes nicht verändert werden sollen?

7.9. Nehmen Sie für die folgende Aufgabe an, dass Sie ein umfangreiches Programm mit einer Vielzahl möglicher Kommandozeilenargumente schreiben wollen. Die jeweiligen Parameter sollen durch ein Minuszeichen am Anfang jedes Arguments getrennt werden, da sie aus mehreren Wörtern bestehen können. Der Programmaufruf

```
prog.exe -i input.txt -o output.txt -mode wcount b2q
```

teilt dem Programm demnach 3 Einstellungsoptionen mit: `-i input.txt`, `-o output.txt` und `-mode wcount b2q`. Beginnen Sie mit der Implementierung Ihres Programms, indem Sie ein Feld erzeugen, welches aus den Zeichenketten zu den entsprechenden Eingabeoptionen besteht. Überprüfen Sie Ihr Programm mit der Ausgabe dieser Parameter auf dem Bildschirm.

7.10. Schreiben Sie ein Programm, welches einen Pfad zu einer Textdatei und ein Wort als Kommandozeilenargumente übergeben bekommt. Das Programm soll in der Textdatei nach allen Vorkommen des angegeben Wortes suchen und die Position dieser auf dem Bildschirm ausgeben. Wie die genaue Positionsangabe aussieht, zum Beispiel i-te Zeile, j-te Spalte oder auch k-tes Wort, ist Ihnen überlassen.

7.11. Überlegen Sie sich, wie die Signatur einer Funktion zur Erweiterung eines zuvor dynamisch allozierten Speichers aussehen sollte, wenn der Anwender lediglich den Zeiger auf den entsprechenden Speicherbereich sowie die neue Speichergröße übergeben möchte, ohne einen Rückgabewert der Funktion auswerten zu müssen.

a) Implementieren Sie eine solche Funktion und erläutern Sie worauf der Benutzer bei der Parameterübergabe zu achten hat.

b) Erweitern Sie die Funktion um ein zusätzliches Argument der Länge des ursprünglichen, bereits beschriebenen Speicherbereichs. Der zugehörige Inhalt soll nun auch im expandierten Speicherbereich erhalten bleiben. Schreiben Sie eine Funktion, die diese

Anforderung umsetzt. Ist dies auch ohne die Angabe der ursprünglichen Arraylänge realisierbar?

7.12. Nachfolgend geht es um die Erweiterung einer Sortierfunktion für den Umgang mit beliebigen Feldtypen und Sortierkriterien. Für die Umsetzung der Aufgabe sind Grundkenntnisse über die Verwendung von Funktionszeigern erforderlich.

a) Schreiben Sie eine Funktion zum aufsteigenden Sortieren eines Integer-Feldes beliebiger Länge. Implementieren und verwenden Sie hierfür Hilfsfunktionen, sowohl für die Feststellung, ob ein Element vor oder hinter einem Vergleichselement anzuordnen ist, als auch für das Vertauschen der entsprechenden Elemente. Der verwendete Sortieralgorithmus ist an dieser Stelle nachrangig.

b) Erweitern Sie die Funktion für Felder beliebiger Datentypen, indem Sie `void`-Zeiger für die Übergabe der entsprechenden Parameter verwenden. Ändern Sie zu Testzwecken den Datentypen von `int` zu `float` und die Sortierrichtung von aufsteigend zu absteigend (durch Anpassung des Kriteriums in der entsprechenden Hilfsfunktion).

c) Um die Anwendbarkeit der Funktion noch genereller zu gestalten, bietet es sich an, die Vergleichs- und die Vertauschungsfunktion selbst in Form von Funktionszeigern zu übergeben. Passen Sie Ihre Sortierfunktion entsprechend an und testen Sie Ihre Implementierung.

d) Nach der Realisierung aller angesprochenen Modifikationen hat die Sortierfunktion bereits ein sehr breites Anwendungsspektrum. Welche weiterführenden Änderungen wären notwendig, damit die Funktion auch auf Listen von Strukturelementen anwendbar ist?

Rekursive Funktionen

7.1. Schreiben Sie eine rekursive Funktion zur Berechnung der Summe alternierender Fakultäten:

$$\mathrm{f}(n) := \sum_{k=1}^{n} (-1)^k k!, \qquad n \in \mathbb{N}.$$

Erweiterte Aufgabenstellung: Realisieren Sie die Funktion nun ohne Verwendung von Hilfsfunktionen oder Schleifenkonstrukten.

7.2. Implementieren Sie eine rekursive Funktion zur Bestimmung von

$$\mathrm{f}(n) := \sum_{k=0}^{n} \prod_{j=1}^{k} \frac{-2j}{n}$$

für beliebige Werte $n \in \mathbb{N}$.

7.3. In den nachfolgenden Teilaufgaben geht es um verschiedene Implementierungsansätze zur Auswertung von:

$$\mathrm{f}(n) := \sum_{k=1}^{n} \frac{(-1)^k}{2k} \; - \; \frac{1}{(2n)!} \prod_{k=1}^{n} (2k-1), \qquad n \in \mathbb{N}.$$

a) Schreiben Sie eine Funktion zur Berechnung von $\mathrm{f}(n)$, welche ohne Rekursionen auskommt.

b) Realisieren Sie die entsprechende Funktion, indem Sie für die Teilterme $\sum_{k=1}^{n} \frac{(-1)^k}{2k}$, $\frac{1}{(2n)!}$ und $\prod_{k=1}^{n}(2k-1)$ jeweils eigene rekursive Hilfsfunktionen implementieren.

c) Realisieren Sie die entsprechende Funktion in Form **einer einzigen** linearen Rekursion. Die Realisierung soll ohne den Gebrauch von Schleifen oder die Verwendung von `C`-Standardfunktionen aus entsprechenden Bibliotheken erfolgen.

7.4. Neben den Randbedingungsgleichungen

$$\binom{n}{0} = \binom{n}{n} = 1, \qquad n \in \mathbb{N},$$

genügen Binomialkoeffizienten unter anderem auch den folgenden Gleichungen

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} = \frac{n}{k} \binom{n-1}{k-1} = \frac{n+1-k}{k} \binom{n}{k-1} = \prod_{i=1}^{k} \frac{n+1-i}{i},$$

wobei $n, k \in \mathbb{N}$ und $1 \leq k < n$.

a) Schreiben Sie für jede der vier Identitäten eine entsprechende Funktion zur Berechnung des Binomialkoeffizienten.

b) Welche Methode ist am wenigsten effizient, benötigt daher die meisten Berechnungen und verursacht die höchste Speicherbelegung? Ist die effizienteste Methode, unabhängig von n und k, eindeutig? Begründen Sie Ihre Antwort.

c) Realisieren Sie eine rekursive Funktion zur Berechnung des Binomialkoeffizienten, deren Rekursionstiefe gleich der kleineren der beiden Zahlen k und $n - k$ ist.

7.5. Schreiben Sie eine rekursive Funktion, welche die ersten n Zahlen der Fibonacci-Folge $(1, 1, 2, 3, 5, \ldots, \mid a_i = a_{i-2}+a_{i-1})$ in absteigender Reihenfolge auf dem Bildschirm ausgibt.

Erweiterte Aufgabenstellung: Realisieren Sie die Aufgabe mit einer einzigen rekursiven Funktion, ohne die Verwendung von globalen Variablen, Feldern oder Schleifen-Konstrukten.

7.6. Schreiben Sie eine Funktion `multi_alloc`, welche dynamisch Speicher für beliebige mehrdimensionale Felder von Elementen vorgegebener Bytelänge alloziert. Um Speicher für ein vierdimensionales `float`-Feld der Dimensionen $3 \times 5 \times 3 \times 2$ bereitzustellen würde man beispielsweise den folgenden Code verwenden:

```
int dims[4] = { 3, 5, 3, 2 };
float ****fl4array = multi_alloc( dims, 4, sizeof( float ) );
```

Hinweis: Für die Implementierung der Funktion ist es nicht bedeutsam, welcher Zeigertyp gerade behandelt wird, da alle berücksichtigten Zeigertypen den gleichen Speicher beanspruchen. Verwenden Sie daher nur `void`-Zeiger. Die entsprechenden Typenkonvertierungen werden in der aufrufenden Funktion vorgenommen.

Bemerkung: Eine einfache rekursive Umsetzung dieser Funktion kommt mit weniger als 10 Zeilen Code (darunter zwei `malloc`-Aufrufe) aus.

7.7. Für die folgende Betrachtung sei ein Labyrinth durch ein zweidimensionales Feld beschrieben. Passierbare Positionen seien durch positive Zahlen gekennzeichnet, Mauern und andere Hindernisse hingegen durch negative Einträge identifiziert.

a) Schreiben Sie eine rekursive Funktion zur Bestimmung eines Weges durch ein solches Labyrinth. Die Startposition soll der Funktion übergeben werden. Die Zielposition darf gegebenenfalls auch global festgelegt sein.

Die Wegbeschreibung soll in Form der zugehörigen Schrittreihenfolge (rechts, links, oben, unten) auf dem Bildschirm ausgegeben werden.

b) Modifizieren Sie die Funktion dahingehend, dass diese den kürzesten Weg durch das Labyrinth bestimmt.

c) Zusätzlich zu der vorangegangenen Labyrinthsbeschreibung soll nun davon ausgegangen werden, dass die positiven Werte der passierbaren Positionen im 2D Feld als Kosten für die Begehung eben dieser anzusehen sind. Adaptieren Sie Ihre Funktion für den Fall der Minimierung dieser Kosten.

8 Lineare Listen

8.1. Erstellen Sie eine Struktur, die von einem Mitarbeiter folgende Daten speichern kann:

- Vorname (max. 50 Zeichen ohne Leerzeichen)
- Nachname (max. 50 Zeichen ohne Leerzeichen)
- Personal-Nr. (4 Ziffern ohne Vorzeichen und Führende 0)

Erstellen Sie weiter eine Struktur für eine lineare einfach verkette Liste von Aufgaben. Einer Aufgabe speichert folgende Daten:

- ID-Nr. (8 Ziffern ohne Vorzeichen und Führende 0)
- Kurzbeschreibung (max. 50 Zeichen ohne Leerzeichen)
- Priorität $(0 \ldots 5)$
- Zeiger auf den zugewiesenen Mitarbeiter
- Zeiger auf die nächste Aufgabe

8.2. Schreiben Sie eine Funktion `erstelle_daten`, die keine Eingabeargumente besitzt und einen Zeiger auf eine Aufgaben-Datenstruktur zurück gibt. Erstellten Sie in dieser Funktion Datenstrukturen für 2 Mitarbeiter (`mitarbeiter1` und `mitarbeiter2`). Anschließend erstellen Sie eine verkette Liste von 3 Aufgaben, von denen die erste Aufgabe `mitarbeiter1` und die letzte Aufgabe `mitarbeiter2` zugeordnet sind. Aufgabe 2 ist keinem Mitarbeiter zugeordnet (`NULL`). Geben Sie die erzeugte Struktur als `return` Parameter zurück. Rufen Sie diese Funktion in Ihrer `main`-Funktion auf.

8.3. Schreiben Sie eine Funktion `datenausgabe`, welche die in Teilaufgabe 2 erzeugten Daten auf der Standardausgabe (Bildschirm, `stdout`) oder einer beliebigen anderen Ausgabe ausgeben kann (`FILE*`). Rufen Sie diese Funktion in Ihrer `main`-Funktion auf.

8.4. Schreiben Sie eine Funktion, die eine neue Aufgabe erstellt und am Anfang einer verketteten Aufgabenliste hinzufügt. Die Funktion soll den neuen Anfang der Liste zurück geben. Rufen Sie diese Funktion in Ihrer `main`-Funktion auf.

8.5. Schreiben Sie eine Funktion, die eine neue Aufgabe erstellt und am Ende einer verketteten Aufgabenliste hinzufügt. Die Funktion soll den ggf. neuen Anfang der Liste zurück geben. Berücksichtigen Sie auch den Fall einer leeren Liste. Rufen Sie diese Funktion in Ihrer `main`-Funktion auf.

8.6. Schreiben Sie eine Funktion, die eine neue Aufgabe erstellt und an die n-te Position einer verketteten Aufgabenliste einfügt. Ist die Liste leer oder $n == 0$, soll das Element der neue Anfang der Liste sein. Wenn n größer als die Anzahl der Listenelemente ist, soll die neue Aufgabe am Ende der Liste hinzugefügt werden. Die Funktion soll den neuen Anfang der Liste zurück geben. Rufen Sie diese Funktion in Ihrer `main`-Funktion auf.

8.7. Schreiben Sie eine Funktion `swap_aufgaben`, die zwei aufeinander folgende Elemente einer verketteten Aufgabenliste vertauscht.

8.8. Sortieren Sie die verketteten Aufgabenliste absteigend nach Priorität (oder ID) mit einem Sortierverfahren Ihrer Wahl. Nutzen Sie dafür die Funktion `swap_aufgaben` aus Teilaufgabe 7.

8.9. Sortieren Sie die verketteten Aufgabenliste aufsteigend alphabetisch nach der Kurzbeschreibung mit einem Sortierverfahren Ihrer Wahl. Nutzen Sie dafür aus Teilaufgabe 7 die Funktion `swap_aufgaben`.

8.10. Geben Sie die Daten der Aufgabe mit der höchsten Priorität aus. Sollte es mehrere Aufgaben mit höchster Priorität geben, soll das in der Reihenfolge erste Element ausgegeben werden.

8.11. Geben Sie die Daten aller Aufgaben von einem einzigen Mitarbeiter aus.

8.12. Schreiben Sie eine Funktion, welche die Aufgabe am Anfang einer verketteten Aufgabenliste entfernt. Die Liste kann auch leer sein! Die Funktion soll den neuen Anfang der Liste zurück geben. Rufen Sie diese Funktion in Ihrer `main`-Funktion auf.

8.13. Schreiben Sie eine Funktion, welche die Aufgabe am Ende einer verketteten Aufgabenliste entfernt. Die Liste kann auch leer sein! Die Funktion soll den ggf. neuen Anfang der Liste zurück geben. Rufen Sie diese Funktion in Ihrer `main`-Funktion auf.

8.14. Schreiben Sie eine Funktion, die eine Aufgabe an n-ter Position aus einer verketteten Aufgabenliste entfernt. Die Liste kann auch leer sein! Wenn n größer als die Anzahl der Listenelemente ist, soll die Aufgabe am Ende der Liste entfernt werden. Die Funktion soll den neuen Anfang der Liste zurück geben. Rufen Sie diese Funktion in Ihrer `main`-Funktion auf.

8.15. Schreiben Sie eine Funktion, die den Benutzer nach dem Namen einer Datei fragt. Anschließend nutzen Sie die Funktion `datenausgabe` (Teilaufgabe 3), um in diese Datei hinein zu schreiben. Rufen Sie diese Funktion in Ihrer `main`-Funktion auf.

8.16. Schreiben Sie eine Funktion, die den Benutzer nach dem Namen einer Datei fragt. Anschließend importieren Sie die Aufgaben und Mitarbeiter. Das Format ist das von der Funktion `datenausgabe` (Teilaufgabe 3) produzierte. Rufen Sie diese Funktion in Ihrer `main`-Funktion auf.

8.17. Per Kommandozeilenaufruf sollen eine verkette Aufgabenliste aus der angegebenen Datei importiert werden. Beispielsweise importiert der Aufruf

```
programm.exe daten.txt
```

die Daten aus der Datei `daten.txt`.