

Übungspaket 21

Funktionen mit Zeigern und Arrays als Parameter

Übungsziele:

1. Funktionen mit Zeigern als Parameter
2. Emulation von Variablenparametern
3. Funktionen mit Arrays als Parameter
4. Zeigervariablen im Stack-Frame

Skript:

Kapitel: 44 bis 47 sowie die Übungspakete 15, 19 und 20

Semester:

Wintersemester 2025/26

Betreuer:

Benjamin, Thomas und Ralf

Synopsis:

In den beiden vorherigen Übungspakten haben wir Funktionen und Zeiger getrennt voneinander geübt. Nun wollen wir beide Aspekte zusammenbringen. Entsprechend werden wir Zeigerarithmetik betreiben und Funktionen sowohl mit Zeigern als auch Arrays aufrufen. Neben einigen einfachen, grundlegenden Algorithmen greifen wir am Ende wieder das alte Thema „Suchen und Sortieren“ auf aus Übungspaket 15.

Dieses Übungspaket ist eher lang, schwierig und recht aufwändig. Aber wer hier einigermaßen durchkommt und die Dinge versteht, die er hier bearbeitet, der hat so ziemlich das Schlimmste hinter sich gebracht. Mit dem hier erworbenen Wissen lässt sich der Rest frohen Mutes angehen. Wer aber noch Schwierigkeiten mit Zeigern, Arrays oder allgemein mit Funktionsaufrufen hat, der sollte sich die entsprechenden Übungspakete nochmals anschauen. Denn je besser man hier vorbereitet ist, um so leichter geht's.

Teil I: Stoffwiederholung

Zunächst wiederholen wir zum x-ten Male einige wesentliche Aspekte aus den vorherigen Übungspaketen. Auch wenn es einigen bereits aus den Ohren heraushängt, so kann man dennoch einige wesentliche Punkte anfangs nicht oft genug erwähnen.

Aufgabe 1: Was sind Zeiger und Arrays

Beantworte die drei folgenden Fragen wieder kurz in eigenen Worten. Was ist ein Array?

Was repräsentiert der Array-Name?

Was ist ein Zeiger?

Aufgabe 2: Grundlagen der Zeiger und Arrays

Für die Beantwortung der weiteren Fragen gehen wir davon aus, dass wir uns immer in demjenigen Block befinden, in dem das Array definiert wurde. Unter dem Begriff „Zielelement“ verstehen wir immer dasjenige Element, auf das der Zeiger zeigt.

Was repräsentiert der Array-Name	<input type="text"/>
Ist der <i>Array-Name</i> eine Konstante oder Variable?	<input type="text"/>
Kann man dem Array-Namen etwas zuweisen?	<input type="text"/>
Was ergibt <code>sizeof(Array-Name)</code> ?	<input type="text"/>
Was ergibt <code>sizeof(Array-Name)</code> nicht?	<input type="text"/>
Was ergibt <code>sizeof(* Array-Name)</code> ?	<input type="text"/>
Ist ein Zeiger eine Konstante oder Variable?	<input type="text"/>
Kann man einem Zeiger etwas zuweisen?	<input type="text"/>
Was ergibt <code>sizeof(Zeiger)</code> ?	<input type="text"/>
Was ergibt <code>sizeof(Zeiger)</code> <i>nicht</i> ?	<input type="text"/>
Was ergibt <code>sizeof(* Zeiger)</code> ?	<input type="text"/>

Aufgabe 3: Fragen zu Funktionen

- Wie nennt man die Platzhalter im Funktionskopf?
- Wie nennt man die Werte beim Funktionsaufruf?
- Wo steht die Rücksprungadresse?
- Wo steht der Funktionswert?
- Wo stehen die lokalen Variablen?
- Wann wird eigentlich der Stack-Frame angelegt?
- Was lässt sich dadurch realisieren?
- Bietet C die Parameterübergabe: Call-by-Value?
- Was passiert dabei?
- Bietet C den Call-by-Reference Mechanismus?

Aufgabe 4: Zeiger und Arrays als formale Parameter

Gegeben seien die beiden Funktionen `int f(int * p)` und `int g(int arr[])`. Bei den folgenden Fragen befinden wir uns immer *innerhalb* der Funktionsrumpfe von `f()` und `g()`.

- Welchen Typ hat der Formalparameter `p`?
- Welchen Typ hat der Formalparameter `arr`?
- Unterscheiden sich beide Parameter voneinander?
- Kennt `f()` die Größe des übergebenen Arrays?
- Kennt `g()` die Größe des übergebenen Arrays?
- Was müssen wir machen?
- Und wie machen wir das?
- Wie sähe das bei `f()` aus?

Aufgabe 5: Kurzformen von Zeigerausdrücken

Was bedeutet der Ausdruck `c = *p++`, wenn `c` und `p` ein `char` bzw. Zeiger darauf sind?

Teil II: Quiz

Aufgabe 1: Zeiger und Arrays als Parameter

Für die folgenden Quizfragen betrachten wir folgende Definitionen und Speicherbild:

```

1 int i = 4711;      3 int a[] = { 1, 2, 4, 8, 16 };
2 int *p = &i;      4 int size_a = sizeof(a)/sizeof(a[0]);

```

Adresse	Variable	Wert	Adresse	Variable	Wert
0xFE7C	int i :		0xFE6C	int a[3]:	
0xFE78	int * p :		0xFE68	int a[2]:	
0xFE74	int size_a:		0xFE64	int a[1]:	
0xFE70	int a[4]:		0xFE60	int a[0]:	

Vervollständige obiges Speicherbild. f() ist nun eine Funktion, deren Parameter jedes Mal richtig spezifiziert sind. Notiere nun jeweils Typ und Wert der beiden aktuellen Parameter.

Aufruf	Parameter 1		Parameter 2		Was „sieht“ f()
	Typ	Wert	Typ	Wert	
f(a, size_a)
f(a + 1, 3)
f(&(a[1]),3)
f(&i, 1)
f(p, 1)
f(&p, 1)
f(a,a+size_a)

Gegeben seien die beiden folgenden Programmzeilen. Vervollständige das Speicherbild für den Fall, dass beide abgearbeitet wurden.

	Adresse	Variable	Wert	Konstanten Segment:
1 char *p = "Ah!";	0xFE84	char c:		0xFE02
2 char c = *p++;	0xFE80	char *p:		0xFE00
				'A' 'h'

Teil III: Fehlersuche

Aufgabe 1: Zeiger und Funktionen

Unser Grufti DR. DO-IT-RIGHT hat bei den Übungen zugeschaut und selbst ein wenig probiert. Bei seinen Versuchen handelt es sich nur um ein paar Testzeilen und nicht um ein sinnvolles Programm. Die Intention wird jeweils durch den Funktionsnamen bzw. die entsprechenden Begleitkommentare klar. Viel Spass beim Finden und Korrigieren der Fehler.

```
1 int who_is_larger( int arr_1[], int arr_2[] )
2     {
3         if ( sizeof(arr_1) > sizeof(arr_2) )
4             return 1;                // arr_1 is larger
5         else if ( sizeof(arr_1) < sizeof(arr_2) )
6             return 2;                // arr_2 is larger
7         else return 0;                // none
8     }
9 int set_to_zero( int a )
10    {
11        int *p;                        // one pointer
12        *p = & a;                       // get address of the actual parameter
13        *p = 0;                          // accessing actual parameter and set to zero
14    }
15 int main( int argc, char **argv )
16    {
17        int *p, a[ 10 ], b[ 5 ], i;
18        for( p = a + 10; a < p; a++ )    // the loop
19            *a = -1;                       // init with -1
20        a -= 10;                           // reset a to its beginning
21        set_to_zero( i );                   // set i to zero
22        i = who_is_larger( a, b );         // comparing sizes
23    }
```

Teil IV: Anwendungen

Vorbemerkung

Den meisten von euch sollte klar geworden sein, dass es hier irgendwie um Funktionen und Zeiger geht. Sollte dies im Einzelfalle doch nicht der Fall sein, dann unbedingt mindestens drei Übungspakete zurück gehen, das Skript nochmals lesen und sofort damit anfangen.

Ferner sollte jedem, der hier weiter arbeitet, folgende Dinge klar sein, und zwar glasklar:

1. In der Programmiersprache C gibt es *nur einen* Parameterübergabemechanismus, den man Call-by-Value nennt. Dabei werden die aktuellen Parameter ausgewertet und den formalen Parametern als Werte zugewiesen.
2. Es gibt in C *kein* Call-by-Reference, auch wenn dies an vielen Stellen immer wiederholt wird.
3. Wenn man Zeiger, beispielsweise die Adressen beliebiger Variablen, übergibt, dann wird dies immer noch mittels Call-by-Value abgearbeitet. Aber dadurch, dass die Funktion jetzt nicht die Variablenwerte sondern deren Adressen bekommt, kann sie quasi „aus der Funktion heraus schauen“ und im Block der aufrufenden Stelle frisch, fromm, fröhlich, frei ändern wie es ihr beliebt.

Aufgabe 1: Tausch zweier char-Parameter

1. Aufgabenstellung

Ziel dieser Aufgabe ist es, eine Funktion `my_ch_swap()` zu entwickeln, die die Werte zweier `char`-Variablen nachhaltig vertauscht.

Beispiel: Im folgenden Programm soll erst `ko` und dann `ok` ausgegeben werden.

```
1 #include <stdio.h>
2
3 int main( int argc, char **argv )
4     {
5         char c1 = 'k', c2 = 'o';
6
7         printf( "vorher : %c%c\n", c1, c2 );
8         my_ch_swap( ... );
9         printf( "nachher: %c%c\n", c1, c2 );
10    }
```

2. Vorüberlegungen

Da die zu entwickelnde Funktion Variablenwerte vertauschen soll, ist es wenig sinnvoll, diese Werte einfach an die formalen Parameter der Funktion zu übergeben. Vielmehr benötigen wir Zeiger auf die konkreten Variablen, damit wir deren Inhalte nachhaltig tauschen können. Andernfalls würden zwar die Variablenwerte *innerhalb* der Funktion getauscht werden, was aber ausserhalb der Funktion keinen Effekt hätte. Also benötigen wir keine „normalen“ Parameter sondern die allseits geliebten Zeiger.

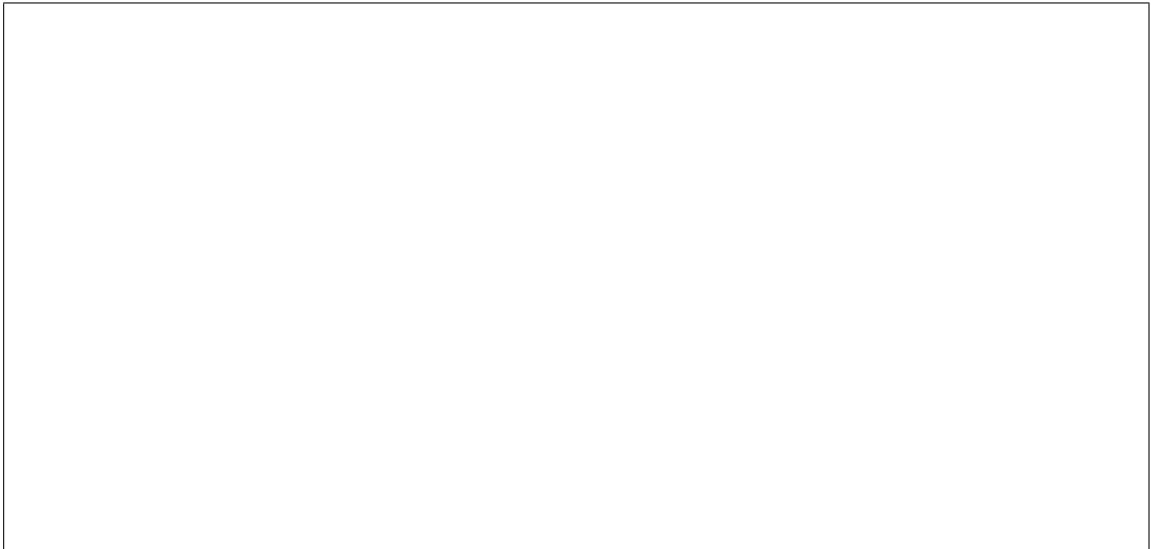
3. Pflichtenheft: Aufgabe, Parameter, Rückgabewert

4. Implementierung

5. Kodierung

6. Stack Frame

Zeichne den Stack-Frame zu Beginn des Funktionsaufrufs:

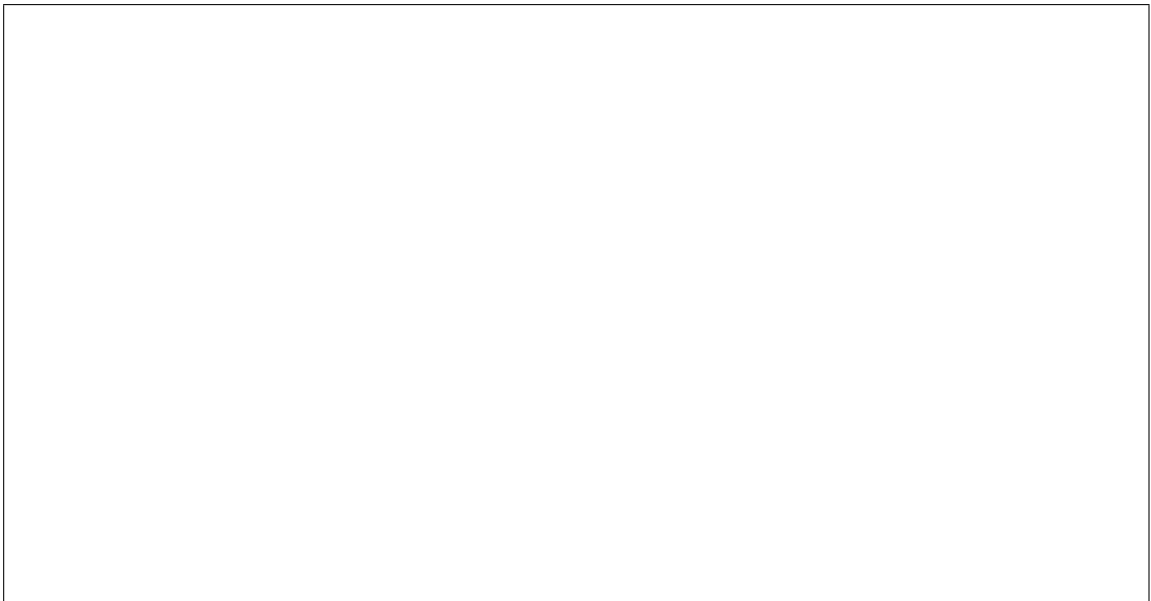


Aufgabe 2: Tausch zweier double-Parameter

1. Aufgabenstellung

Wiederhole die vorherige Aufgabe. Nur sollen diesmal zwei **double**-Werte vertauscht werden. Aufgrund der Vorarbeiten könnt ihr direkt mit der Kodierung beginnen.

2. Kodierung



Aufgabe 3: Ringtausch dreier int-Parameter

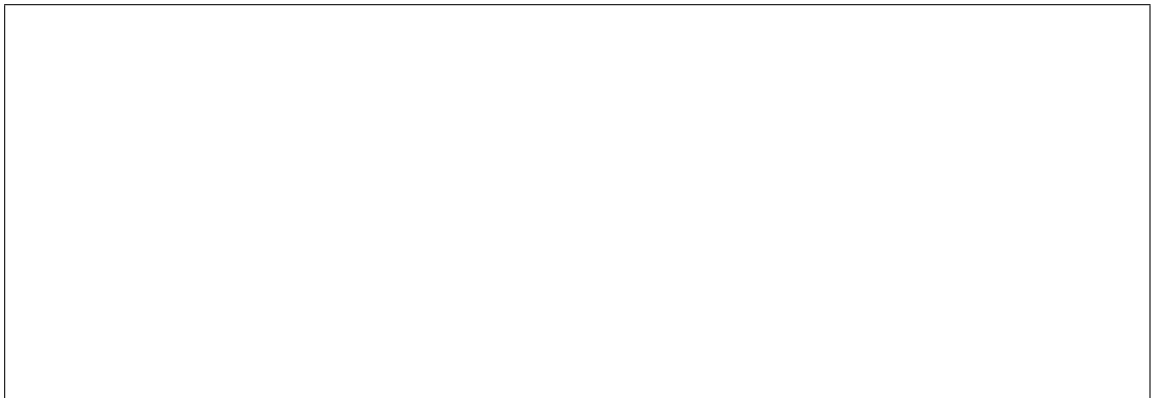
1. Aufgabenstellung

Entwickle eine Funktion, die die Werte dreier `int`-Parameter zyklisch vertauscht.

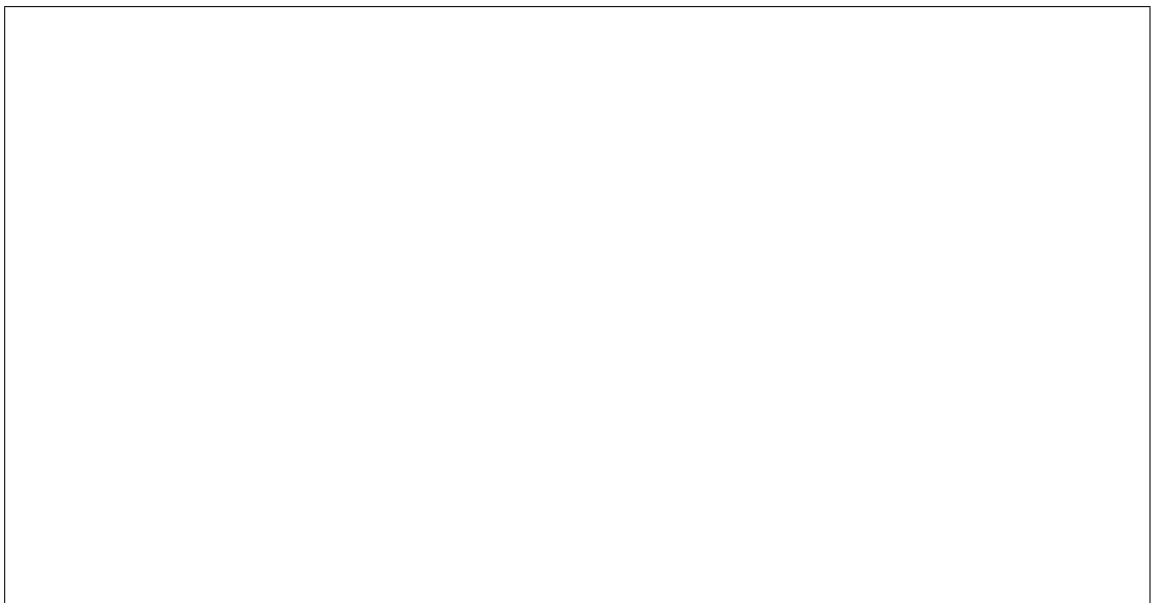
Beispiel: Die Werte der Parameter `a`, `b` und `c` sollen nach dem Funktionsaufruf in den Variablen `b`, `c` und `a` stehen.

Da auch diese Aufgabe den beiden vorherigen sehr ähnelt, reichen hier Implementierung und Kodierung.

2. Implementierung



3. Kodierung



Aufgabe 4: Initialisierung von Arrays

1. Aufgabenstellung

Entwickle zwei Funktionen zur Initialisierung von Zeichen-Arrays. Die erste Funktion, `init_lower()` soll die Elemente des übergebenen Arrays mit den 26 Kleinbuchstaben initialisieren. Die zweite Funktion, `init_digits()` soll die Elemente des übergebenen Arrays mit den zehn Ziffern initialisieren. Ferner benötigen wir (zu Testzwecken) eine Funktion zum Ausgeben eines Zeichen-Arrays.

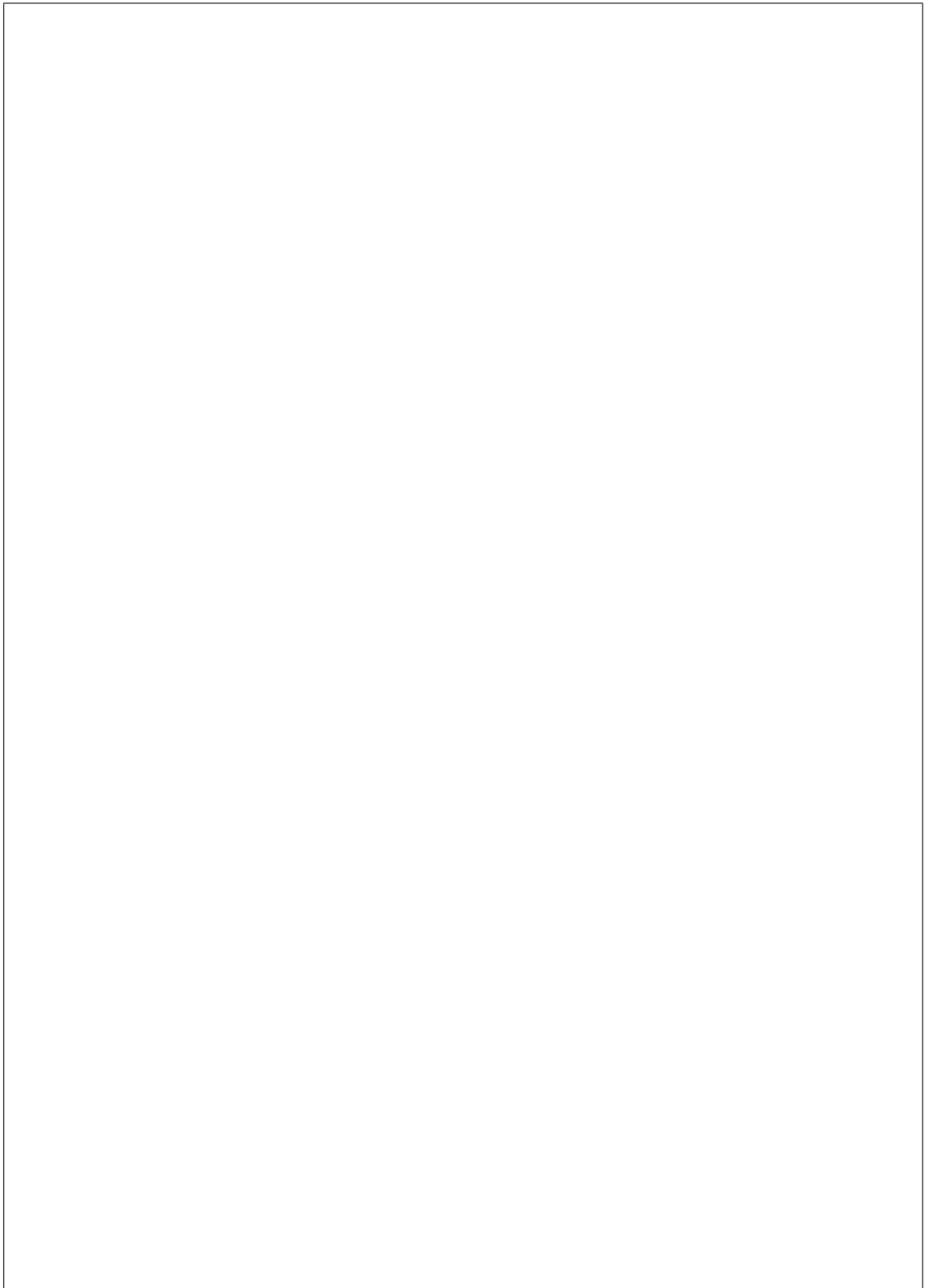
2. Pflichtenheft: Aufgabe, Eingabe, Ausgabe, Sonderfälle

3. Implementierung

Funktion zur Ausgabe eines Arrays

Funktion zur Initialisierung eines Arrays mit Kleinbuchstaben

4. Kodierung

A large, empty rectangular box with a thin black border, occupying most of the page. It is intended for the student to write their code for the '4. Kodierung' task.

Aufgabe 5: Generalisierte Initialisierungsfunktion

1. Aufgabenstellung

In der vorherigen Aufgabe haben wir zwei unterschiedliche Funktionen zum Initialisieren eines Zeichen-Arrays entwickelt. Ein genauer Blick auf die beiden Funktionen zeigt, dass sie sich sehr ähneln.

Was sind die beiden Unterschiede?

Ziel dieser Aufgabe ist es, beide Funktionen in eine generalisierte Initialisierungsfunktion zu überführen. Dazu müssen wir sie lediglich um zwei Parameter erweitern, die den Unterschieden Rechnung trägt. Zeige Anhand der Initialisierung mit Ziffern sowie mit Großbuchstaben, dass die neue Funktion korrekt funktioniert. Führe hierzu am besten eine Handsimulation durch.

2. Pflichtenheft: Aufgabe, Parameter, Ausgabe

3. Testdaten

Testdaten, wie in der Aufgabenstellung vorgegeben.

4. Implementierung

Generalisierte Funktion zur Initialisierung eines Arrays

5. Kodierung



Aufgabe 6: Berechnung von Partialsummen

1. Aufgabenstellung

In vielen Anwendungen liegen unterschiedliche Daten in einem Array vor. Für den Nutzer, beispielsweise einem Masterstudenten der Elektrotechnik, kann es nun interessant sein, die Daten unterschiedlich zu verknüpfen. Eine dieser Möglichkeiten ist das Bilden von Partialsummen, die jeweils einen spezifizierten Bereich des Arrays aufsummieren. In dieser Übungsaufgabe sind wir an Partialsummen interessiert, die sich jeweils symmetrisch um das mittlere Element befinden.

Beispiel: Daten im Array: 1 3 5 7 9, Ausgabe: 25, 15 und 5

Entwickle eine Funktion, die die entsprechende Summe der Elemente eines `int`-Arrays ermittelt. Zum Testen können obige Daten direkt in ein Array geschrieben werden. Das Hauptprogramm soll alle Partialsummen berechnen und ausgeben.

2. Vorüberlegungen

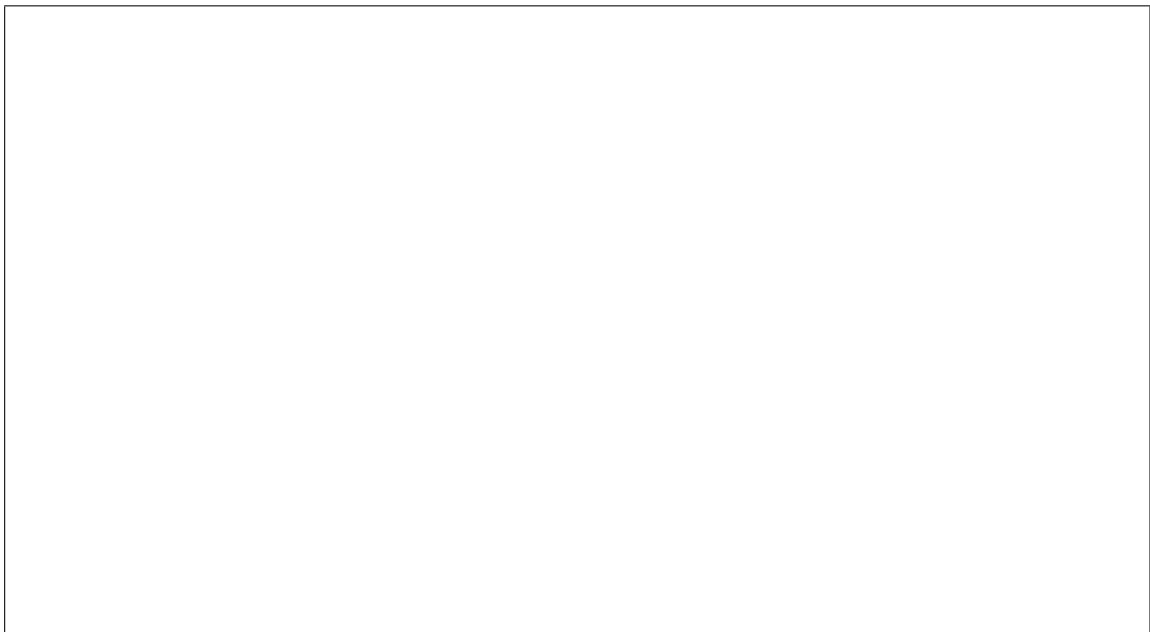
In der Aufgabenstellung sind schon genügend viele Hinweise für eine Implementierung gegeben. Dennoch gibt es grundsätzlich zwei unterschiedliche Möglichkeiten, die zu implementierende Funktion zu parametrisieren.

1. Wir rufen die Funktion mit drei Parametern auf: dem eigentlichen Array sowie dem Anfang und dem Ende der Partialsumme. Für die obigen Testdaten könnten diese Parameter die Werte 1 und 3 sein, sodass 15 als Ergebnis geliefert würde.
2. Wir rufen die Funktion mit nur zwei Parametern auf: dem eigentlichen Array und der Länge der Partialsumme. Jetzt müssten wir aber beim Funktionsaufruf den Anfang des Arrays „verschieben“, was wir aber bereits geübt haben.

3. Pflichtenheft: Aufgabe, Eingabe, Ausgabe, Parameter, Rückgabewert



4. Implementierung



5. Kodierung

Eine Funktion zum Drucken der betrachteten Array-Elemente:

Die Funktion zum Berechnen der Partialsumme:

Das Hauptprogramm:

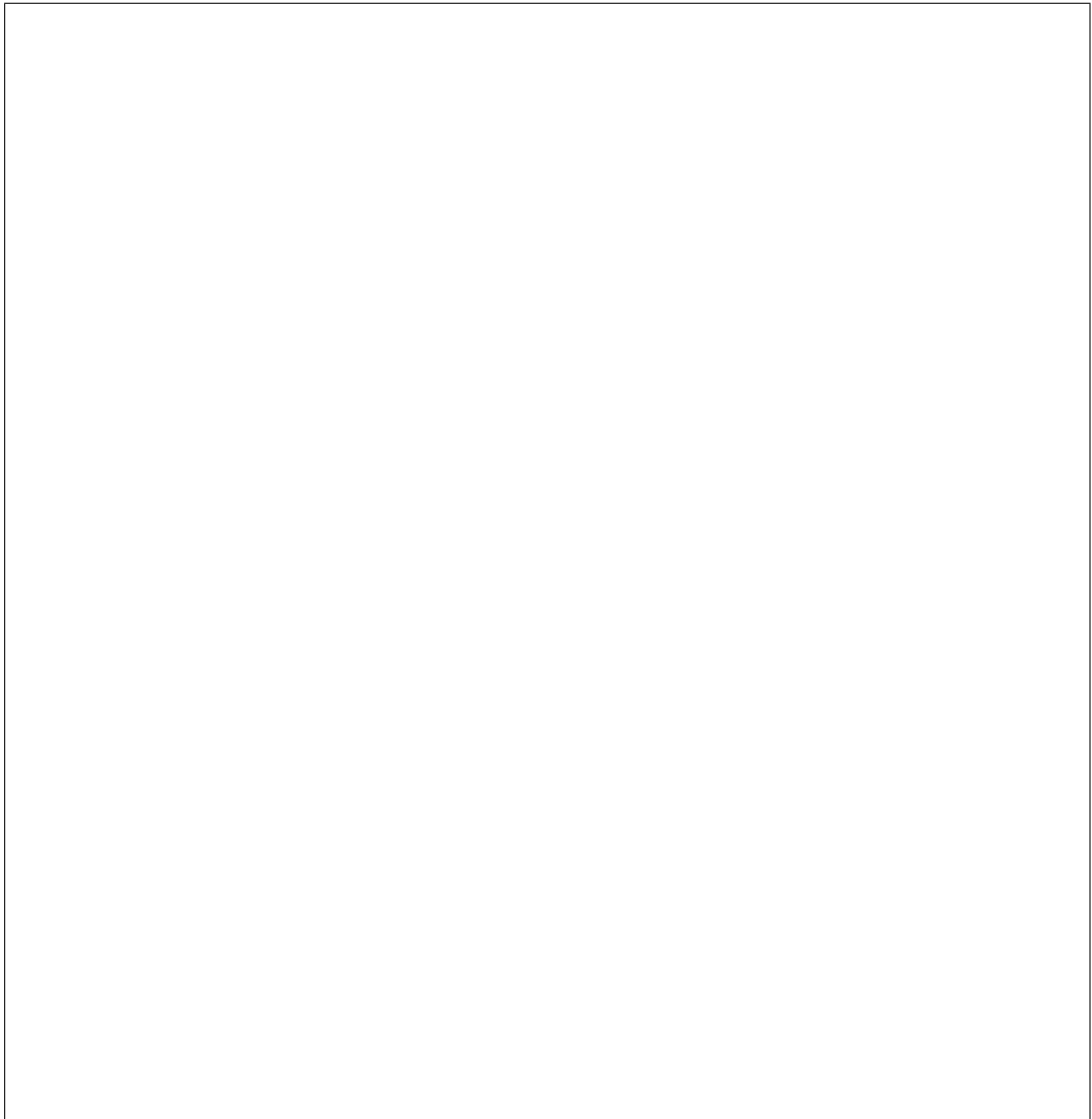
Aufgabe 7: Bubble Sort als Funktion

Bereits in Übungspaket 15 haben wir den Bubble Sort Algorithmus entwickelt. Damals geschah alles im Hauptprogramm `main()`, was wir hier nun ändern wollen ;-)

1. Aufgabenstellung

Nimm den Bubble-Sort Algorithmus aus Übungspaket 15 und „kapsel“ ihn in einer Funktion. Ebenso sind die Anweisungen zum Drucken des Arrays in einer Funktion unterzubringen. Da es sich hier lediglich um das Verschieben einiger Anweisungen handelt, können wir gleich mit der Kodierung fortfahren.

2. Kodierung



Aufgabe 8: Suchen in Tabellen

1. Aufgabenstellung

Nimm den Algorithmus zum Suchen in Tabellen aus Übungspaket 15 und „kapsel“ ihn in einer Funktion `search()`. Diese Funktion benötigt die zu suchende Zahl, das Array und dessen Größe, die wir mit `size` bezeichnen. Ist die gesuchte Zahl im Array vorhanden, soll `search()` den entsprechenden Index zurückgeben. Da wir wieder nur einige Anweisungen verschieben, beginnen wir gleich mit der Kodierung.

Es bleibt jetzt aber noch die Frage zu klären, welcher Index im Fehlerfalle zurückgegeben werden soll. Dazu folgende Fragen:

Welche Indizes sind „gültig“?

Welcher Index bietet sich im Fehlerfalle an?

Warum bietet sich dieser Wert an?

2. Kodierung