



Universität Augsburg
Institut für Informatik

Vorkurs Informatik

Fakultät für Angewandte Informatik

Lehrprofessur für Informatik

PROF. DR. LORENZ

Übungblatt Thema 6

Allgemeine Hinweise:

- Achten Sie bei allen Programmieraufgaben auf *Kompilierbarkeit* und *Einhaltung der Coding Conventions* (werden zu jedem Thema in den Vorlesungsfolien erklärt); auch dann, wenn es nicht explizit im Aufgabentext gefordert ist.

- Gehen Sie davon aus, dass alle Programme in der Programmiersprache C zu erstellen sind.

- Kompilieren Sie alle Ihre Programme mit den folgenden *Compiler-Schaltern*:

`-Wall -Wextra -std=c11 -pedantic`

Achten Sie darauf, dass trotz Verwendung dieser Schalter keine Fehler-/Warnmeldungen erzeugt werden.

In den folgenden Programmieraufgaben werden Sie sich mit der Benutzung von

- Arrays und
- Strings

beschäftigen.

Zur Eigenrecherche können u.a. die folgenden Internetseiten dienen:

- https://de.wikibooks.org/wiki/C-Programmierung:_Arrays
- https://de.wikibooks.org/wiki/C-Programmierung:_Arrays#Strings
- https://de.wikibooks.org/wiki/C-Programmierung:_Standard_Header#string.h

Aufgabe 1 (*Arrays, ca. 15 Minuten*)

Erstellen Sie für jede Teilaufgabe jeweils eine C-Datei mit einer `main`-Funktion (ohne weitere Funktionen).

a)

- Deklarieren Sie `double`-Array `v` mit 5 Komponenten.
- Speichern Sie in allen Komponenten von `v` den Wert 0.0 (Schleife benutzen!).
- Geben Sie den Wert der 4-ten Komponente aus.

b)

- Deklarieren Sie `int`-Array `v` mit 10 Komponenten.
- Speichern Sie in allen Komponenten von `v` eine ganze Zufallszahl (Schleife benutzen!).
- Geben Sie zeilenweise die Werte der Komponenten von `v` aus (Schleife benutzen!).

c)

- Deklarieren Sie `char`-Array `v` mit 10 Komponenten.
- Speichern Sie in den Komponenten von `v` die Ziffern von '0' bis '9' (Schleife benutzen!).
- Geben Sie zeilenweise die Werte der 4-ten und 7-ten Komponente aus.

d)

- Deklarieren Sie `int`-Array `v` mit 10 Komponenten.
- Speichern Sie in der `i`-ten Komponenten von `v` das Quadrat von `i` (Schleife benutzen!).
- Geben Sie zeilenweise die Differenzen zwischen zwei aufeinanderfolgenden Komponenten aus.

Aufgabe 2 (*Funktionen für Arrays, ca. 30 Minuten*)

Erstellen Sie für jede Teilaufgabe jeweils eine C-Datei mit einer `main`-Funktion, einer weiteren Funktion gemäß der Aufgabenstellung und ggf. nützlichen Funktionen aus vorherigen Teilaufgaben. Die Erstellung der neuen Funktion erfordert jeweils die Benutzung von geeigneten Schleifen. In der `main`-Funktion testen Sie jeweils die neue Funktion.

a) Erstellen Sie eine Funktion mit dem Prototyp

```
void array_rand(int v[], int size)
```

nach folgenden Vorgaben:

- Es soll in den ersten `size` Komponenten des Arrays `v` jeweils ein Zufallswert zwischen -100 und +100 gespeichert werden.

In der `main`-Funktion testen Sie die neue Funktion, indem Sie ein Array mit 10 Komponenten anlegen, dann mit Hilfe der neuen Funktion `array_rand` Zufallswerte in dem Array abspeichern, und diese zum Schluss in einer Schleife jeweils getrennt durch Leerzeichen ausgeben.

b) Erstellen Sie eine Funktion mit dem Prototyp

```
void array_print(int v[], int size)
```

nach folgenden Vorgaben:

-
- Es sollen die ersten **size** Komponenten des Arrays **v** zeilenweise auf Kommandozeile ausgegeben werden.

In der **main**-Funktion testen Sie die neue Funktion, indem Sie ein Array mit 10 Komponenten anlegen, in den ersten 8 dieser Komponenten Zufallswerte abspeichern, und diese zum Schluss mit **array_print** ausgeben. Wie erklären Sie die Ausgabe, wenn Sie alle 10 Komponenten ausgeben?

c) Erstellen Sie eine Funktion mit dem Prototyp

```
int array_ispos(int v[], int size)
```

nach folgenden Vorgaben:

- Falls die ersten **size** Komponenten des Arrays **v** jeweils positive Zahlen sind, soll 1 zurückgegeben werden, sonst soll 0 zurückgegeben werden.

In der **main**-Funktion testen Sie die neue Funktion, indem Sie ein Array mit 10 Komponenten anlegen, in diesem Zufallswerte abspeichern, dann mit **array_ispos** testen, ob alle Werte positiv sind, und zum Schluss alle Werte ausgeben.

d) Erstellen Sie eine Funktion mit dem Prototyp

```
int array_contains(int v[], int size, int number)
```

nach folgenden Vorgaben:

- Es soll berechnet und zurückgegeben werden, wie oft **number** unter den ersten **size** Komponenten des Arrays **v** vorkommt.

In der **main**-Funktion testen Sie die neue Funktion, indem Sie ein Array mit 10 Komponenten anlegen, in diesem Zufallswerte abspeichern, dann mit **array_contains** testen, ob der Wert 0 enthalten ist, und zum Schluss alle Werte ausgeben.

Aufgabe 3 (*Funktionen für Arrays - Fortsetzung, ca. 45 Minuten*)

Implementieren Sie nach dem Schema aus Aufgabe 2 folgende weitere Funktionen und testen Sie diese in einer **main**-Funktion. Einen geeigneten Prototyp der Funktion entwerfen Sie dabei jeweils selbst.

a) (Minimum)

Erstellen Sie eine Funktion, die den kleinsten Wert in einem Array von Dezimalzahlen berechnet und zurückgibt.

b) (Summe / Mittelwert)

Erstellen Sie eine Funktion, die die Summe (den Mittelwert) aller Werte in einem Array von Dezimalzahlen berechnet und zurückgibt (*Der Mittelwert ist die Summe der Werte geteilt durch die Anzahl der Werte*).

c) (Addition / Subtraktion)

Erstellen Sie eine Funktion, die zwei Arrays von Dezimalzahlen komponentenweise addiert (subtrahiert).

d) (Vergleich)

Erstellen Sie eine Funktion, die testet, ob zwei Arrays von Dezimalzahlen den gleichen Inhalt

haben.

e) (Kopie)

Erstellen Sie eine Funktion, die den Inhalt eines Arrays von Dezimalzahlen in ein anderes Array kopiert.

f) (Zählen)

Erstellen Sie eine Funktion, die zählt, wie viele Komponenten eines Arrays von ganzen Zahlen ungerade Zahlen (gerade Zahlen / Primzahlen / Quadratzahlen / positive Zahlen) sind.

g) (Initialisieren)

Erstellen Sie eine Funktion, die ein Array von ganzen Zahlen mit n Komponenten mit den ersten n positiven ganzen Zahlen (positiven geraden Zahlen / positiven ungeraden Zahlen / Primzahlen / Quadratzahlen) befüllt.

Aufgabe 4 (*Strings, ca. 75 Minuten*)

Ein wichtiger Spezialfall von Arrays sind **Strings**. Strings kann man in C unter anderem als Arrays vom Typ `char` realisieren. Ein String hat also eine durch die Deklaration festgelegte Anzahl von Komponenten und in jeder Komponente kann ein Zeichen gespeichert werden. Wir können String-Variablen dann wie Arrays deklarieren und uns dabei entscheiden, ob wir die Variable bereits in der Deklaration initialisieren oder die einzelnen Zeichen erst später über Wertzuweisungen an die Arraykomponenten festlegen.

Es gibt eine zentrale Besonderheit bei Strings im Vergleich zu allgemeinen Arrays: Strings werden im Arbeitsspeicher immer durch ein bestimmtes Zeichen abgeschlossen, nämlich dem Zeichen `'\0'`. Dieses Zeichen heißt **binäre Null**. Die binäre Null ist das erste Zeichen der ASCII-Tabelle. Eine String-Variable repräsentiert in C deshalb die Folge aller Zeichen im Speicher ab ihrer Speicheradresse bis zur ersten Speicherzelle mit dem Inhalt `'\0'`. Deshalb müssen wir immer streng darauf achten, dass das Ende eines Strings entsprechend gesetzt ist. Zum Beispiel erhält man mit

```
char w[10];  
w[0] = '\0';
```

einen leeren String der Länge 0 (entspricht "").

Wir sehen an diesem Beispiel, dass wir den reservierten Speicherbereich nicht komplett ausnutzen müssen. Es gilt das gleiche Prinzip wie bei Arrays: Man legt die maximale Länge eines Strings zum Beispiel durch eine symbolische Konstante fest, und wählt diese Konstante so, dass in der Programmanwendung keine längeren Strings vorkommen.

Dabei müssen wir beachten, immer noch eine freie Komponente für die abschließende binäre Null zu haben - auch diese sollte immer im reservierten Bereich liegen. Legt man also einen String mit n Komponenten an, so kann man in diesem maximal $n - 1$ Zeichen speichern.

Bei der Übergabe eines Strings an eine Funktion müssen wir nicht, wie bei allgemeinen Arrays, die Länge des Strings in einem separaten Eingabeparameter mit übergeben. Wenn man den übergebenen String Zeichen für Zeichen durchläuft, kann man dessen Ende durch Erreichen der binären Null feststellen:

```
if (w[i] == '\0') ...
```

So kann man zum Beispiel mit einer Funktion die Anzahl der Zeichen eines Strings berechnen - dies macht die Bibliotheksfunktion `strlen` aus `string.h`.

Wir können Strings entweder zeichenweise mit einer Schleife ausgeben (siehe Übungsaufgaben), oder mit `printf` und der Umwandlungsangabe `%s`:

```
printf("%s", w);
```

Implementieren Sie nun nach dem Schema aus Aufgabe 2 folgende Funktionen für Strings und testen Sie diese in einer `main`-Funktion. Einen geeigneten Prototyp der Funktion entwerfen Sie dabei jeweils selbst.

a) (Strings kopieren (`strcpy`))

Implementieren Sie eine eigene Funktion, die wie die Bibliotheksfunktion `strcpy` einen String kopiert.

b) (Strings aneinanderhängen (`strcat`))

Implementieren Sie eine eigene Funktion, die wie die Bibliotheksfunktion `strcat` zwei Strings aneinanderhängt.

c) (Strings ausgeben)

Implementieren Sie eine Funktion, die ohne Aufruf von `printf` einen String ausgibt.

d) (Zeichen zählen)

Implementieren Sie eine Funktion, die zählt, wie oft ein frei wählbares Zeichen in einem String vorkommt.

e) (Zeichen finden)

Implementieren Sie eine Funktion, die herausfindet, ob ein frei wählbares Zeichen in einem String vorkommt, und im positiven Fall den kleinsten Index dieses Zeichens in dem String zurückgibt.

f) (Zeichen ersetzen)

Implementieren Sie eine Funktion, die jedes Vorkommen eines frei wählbaren Zeichens in einem String durch ein anderes frei wählbares Zeichen ersetzt.

g) (Mehr Zeichen zählen)

Implementieren Sie eine eigene Funktion, die sich wie die Bibliotheksfunktion `strspn` verhält.

Aufgabe 5 (*Syntaxfehler, ca. 15 Minuten*)

In die folgenden Ausschnitte von C-Programmen haben sich einige Fehler eingeschlichen! Finden, benennen und korrigieren Sie die Fehler.

Hinweis:

Der Code ist nicht immer ein vollständiges Programm. Wenn Sie den Code in diesem Fall testen wollen, müssen Sie diesen vorher zu einem vollständigen Programm ergänzen!

a)

```
1  int main(void)
2  {
3      int x;
4      int v[x];
5      return 0;
6  }
```

b)

```
1 int main(void)
2 {
3     int v[5.0];
4     return 0;
5 }
```

c)

```
1 int main(void)
2 {
3     int v[];
4     return 0;
5 }
```

d)

```
1 void test(void)
2 {
3     return 0;
4 }
```

e)

```
1 int test(void)
2 {
3     printf("5");
4 }
```

f)

```
1 int test(void)
2 {
3     return 0;
4     printf("5");
5 }
```