Universität Augsburg
Institut für Informatik

Übung zur Vorlesung Informatik 1 WiSe 2022/2023
Fakultät für Angewandte Informatik
Institut für Informatik
Dr. M. Frieb, Dr. J. Mische, V. Schmid, L. Petrak, P. Schalk 21.10.2022

Übungsblatt 1

Abgabe: bis 31.10.2022 9:00 Uhr

- Dieses Übungsblatt muss im Team abgegeben werden (Einzelabgaben sind nicht erlaubt!).
- Die **Zeitangaben** geben zur Orientierung an, wie viel Zeit für eine Aufgabe später in der Klausur vorgesehen wäre; gehen Sie davon aus, dass Sie zum jetzigen Zeitpunkt wesentlich länger brauchen und die angegebene Zeit erst nach ausreichender Übung erreichen.

* leichte Aufgabe / ** mittelschwere Aufgabe / *** schwere Aufgabe

Allgemeine Hinweise zum Übungsbetrieb

1. Die Übungsblätter sind so gedacht, dass Sie bereits **vor** dem Übungstermin versuchen, einige Teilaufgaben zu lösen (am besten aus allen Aufgaben). Am Übungstermin (Tutorium) tauschen Sie sich mit Ihrem Team aus, wo Sie nicht weiter kommen und fragen Ihren Tutor, wenn Sie im Team keine Lösung finden. Diskutieren Sie auch außerhalb des Übungstermins mit Ihrem Team über die verschiedenen Lösungsmöglichkeiten und verständigen sich auf **eine** gemeinsame Lösung, die sie abgeben. Dies soll aber nicht heißen, dass jede Person im Team nur eine Aufgabe löst! Da Sie in der Prüfung (und im echten Leben) auf sich gestellt sind, bearbeiten Sie **alle** Aufgaben grundsätzlich selbst, damit Sie sie auch zukünftig selbst lösen können.

2. Neben den Übungsgruppen bieten wir jede Woche den *Offenen Zoom-Arbeitsraum* an: Dieser findet jeden Freitag von 14:00 bis 17:00 Uhr statt. Es ist eine zusätzliche Möglichkeit, um sich mit Ihrem Team zu treffen, um weiter am Übungsblatt zu arbeiten. Für Fragen stehen Ihnen Tutoren zur Verfügung und Sie können zudem Fragen stellen, die sich nicht auf das aktuelle Übungsblatt beziehen. Der erste Offene Zoom-Arbeitsraum findet am **28.10.2022** statt.
   Zoom-Link:
   https://uni-augsburg.zoom.us/j/97733319417?pwd=RnZjOVhpZktINlVKTlVPcm1uTEEyUT09

3. Zusätzlich zu den Übungsblättern können Sie Ihre Programmierfähigkeiten im *Betreuten Programmieren* ausbauen. Hierzu erscheint jede Woche ein eigenes Übungsblatt mit einer größeren Programmieraufgabe. Jeden Mittwoch in der Zeit 14:00 bis 17:15 Uhr erhalten Sie hierzu Betreuung in den CIP-Pools 1001N, 1002N und 1005N (bei geringer Nachfrage wird die Betreuung evtl. nur noch in einem oder zwei CIP-Pools stattfinden). Die Aufgaben können Sie alleine oder im Team dort bearbeiten. Das erste Betreute Programmieren findet am **26.10.2022** statt.

Allgemeine Hinweise zur Abgabe der Übungsblätter

Wir geben Ihnen die Möglichkeit, Ihre Bearbeitung der wöchentlich herausgegebenen Übungsblätter abzugeben und von einem Tutor korrigieren zu lassen. So erhalten Sie zeitnah Feedback, wo Sie stehen und erhalten im Idealfall noch einen Notenbonus von 0,3 auf eine bestandene Klausur. Folgende Punkte sind dabei zu beachten:

1. **Abgabeort:** Die Abgabe erfolgt digital als Upload in der Digicampus-Veranstaltung der Vorlesung. Gehen Sie dafür in der Digicampus-Veranstaltung der Vorlesung auf den Reiter *Dateien* und wählen Sie den Ordner *Abgabe von Übungsblättern*. Innerhalb dieses Ordners wählen Sie den Ordner zu dem Übungsblatt aus, das Sie gerne abgeben möchten (z.B. *Übungsblatt 01*). Dort laden Sie Ihre Abgabe in der Form hoch, wie es im nächsten Punkt beschrieben wird.

2. **Abgabeformat:** Für Programmieraufgaben sind alle Quellcode-Dateien als `*.c/*.h`-Dateien abzugeben. Alle weiteren Aufgaben geben sie als `*.pdf`-Datei ab (z.B. als gut lesbarer Scan Ihrer handschriftlichen Lösung). Neben den schriftlichen Aufgaben muss auch der von Ihnen verfasste Programmcode in die `*.pdf`-Datei eingebunden werden muss, damit die Korrektoren einen Ort haben, an den sie Anmerkungen aufschreiben können. Sämtliche Dateien, die zu Ihrer Lösung gehören, packen Sie in ein Archiv mit dem Namen

   `TutoriumXX_<Name1>_<Name2>_<Name3>.zip`

   Dabei ersetzen Sie `XX` durch die Nummer des Ihnen zugeteilten Tutoriums (zum Beispiel `03`) und `<Name1>`, `<Name2>` und `<Name3>` durch die Namen Ihrer Teammitglieder. Ein korrekter Dateiname wäre zum Beispiel `Tutorium05_Schalk_Petrak_Lorenz.zip`.
   Achtung: Alle Abgaben, die nicht exakt dieser Form entsprechen, werden aufgrund des erhöhten Organisationsaufwandes nicht korrigiert!

3. **Korrektur:** Wenn Sie sich bei Ihrer Abgabe an die obenstehenden Punkte gehalten haben, erhalten Sie nach ein paar Tagen die Korrektur Ihrer Abgabe per Mail von Ihrem Tutor. Darin finden Sie neben der Bewertung der Lösung auch hilfreiche Hinweise und Tipps, was Sie in Zukunft beachten sollten. Lesen Sie sich Ihre Korrektur sorgfältig durch und beachten Sie die darin enthaltenen Hinweise für die Zukunft.

Allgemeine Hinweise zu den Programmieraufgaben

- Achten Sie bei allen Programmieraufgaben auf *Kompilierbarkeit* und *Einhaltung der Coding Conventions*; auch dann, wenn es nicht explizit im Aufgabentext gefordert ist.

- Gehen Sie davon aus, dass alle Programme in der Programmiersprache C (Spezifikation C89) zu erstellen sind.

- Kompilieren Sie alle Ihre Programme mit den folgenden *Compiler-Schaltern*:
  `-Wall -Wextra -ansi -pedantic`
  Achten Sie darauf, dass trotz Verwendung dieser Schalter keine Fehler-/Warnmeldungen erzeugt werden.

C-Standard

Für die Programmiersprache C existieren verschiedene **Standards** (Spezifikationen), die im Laufe der Zeit zur Normierung der Sprache von speziellen Komitees entwickelt wurden bzw. immer noch werden.
In der Veranstaltung *Informatik 1* werden wir uns an den Standard **C89** (auch: ISO C90, ANSI C) halten, der Grundlage vieler Weiterentwicklungen und die am weitesten verbreitete Spezifikation von C ist.

- *C89*
  http://port70.net/~nsz/c/c89/c89-draft.html
- *C89 Rationale* (Begleitdokument mit Erklärungen)
  http://port70.net/~nsz/c/c89/rationale/
- *Übersicht über die C-Standard-Bibliothek* (Deutsch)
  https://www2.hs-fulda.de/~klingebiel/c-stdlib/

Achtung: Die Lektüre des Standards ist nicht zum Lernen gedacht. Verwenden Sie diesen, um konkrete Sachverhalte (wie die Verwendung einer bestimmten Funktion der Standard-Bibliothek) nachzuschlagen.

Eine Anleitung für die Einrichtung des C-Compilers **gcc** unter Windows/macOS wird im Digicampus zur Verfügung gestellt (GNU/Linux: üblicherweise vorinstalliert).

Entwicklungsumgebung (IDE)

Von der Verwendung einer Entwicklungsumgebung wird Programmieranfängern *abgeraten*. Benutzen Sie stattdessen einen einfachen Texteditor und die Kommandozeile.

Empfehlungen (Auswahl):

- Einfache Editoren, die nur Syntax Highlighting unterstützen, z.B. *gedit* [Linux]
- *Notepad++* [Windows]
  https://notepad-plus-plus.org/
- *Visual Studio Code* [Windows, macOS, GNU/Linux]
  https://code.visualstudio.com/
  Hierbei handelt es sich bereits um eine vollwertige Entwicklungsumgebung. Benutzen Sie für den Einstieg nur die einfachen Funktionen und stellen die automatische Codevervollständigung ab!

Hinweis: Sowohl Notepad++ als auch Visual Studio Code verfügen neben dem nützlichen Syntax-Highlighting auch über Funktionen zur Autovervollständigung. Diese sollten sie *deaktivieren*, da sie beim programmieren lernen hinderlich sind und in der Klausur zur Informatik 1 auch nicht zur Verfügung stehen:

- *Notepad++*: In der Menüleiste Einstellungen → Einstellungen klicken, im sich öffnenden Fenster links Autovervollständigung auswählen und die Optionen „Autovervollständigung aktivieren“ und „Funktionsparameter anzeigen“ beide deaktivieren.
- *Visual Studio Code*: In der Menüleiste File → Preferences → Settings auswählen und im Suchfeld nach Word Based Suggestions suchen und deaktivieren. Installieren Sie auch keine Erweiterungen – das bereits vorhandene Syntax-Highlighting ist vollkommen ausreichend.

Coding Conventions

Ein *guter Programmierstil* zeichnet sich dadurch aus, dass sich der Programmierer an vorgegebene Konventionen hält, die sich im Laufe der Zeit als vorteilhaft erwiesen haben. Wesentliche Gründe für die Einhaltung eines guten Stils sind:

- Bessere Lesbarkeit
- Bessere Wartbarkeit
- Bessere Team-/Projektarbeit
- Geringere Fehleranfälligkeit

Es gibt allerdings nicht *den richtigen Stil*, sondern lediglich eine Menge an verschiedenen Vorschlägen, die viele Gemeinsamkeiten aufweisen. Wichtig ist es deshalb, sich an firmen-/projektspezifische Vorgaben halten zu können, indem man die Grundlagen lernt.

Als Stil für dieses Semester orientieren Sie sich an diesem Dokument:

- Linux Kernel Style (Kapitel 1-4, 6, 8, 16)
  https://www.kernel.org/doc/html/v4.18/process/coding-style.html

Lesen Sie nicht zu Beginn des Semesters das Dokument komplett, sondern wöchentlich jeweils die Kapitel des Style Guides, die zu den vorgestellten Vorlesungsinhalten passen.

Achten Sie bei allen zukünftigen Übungsaufgaben auf **Kompilierbarkeit** und die **Einhaltung der Coding Conventions**. Kommentieren Sie gegebenenfalls nicht funktionierende Programmteile aus, um ein minimales kompilierbares Programm abgeben zu können. Schreiben Sie in den auskommentierten Teil auch, was dieser bezwecken sollte.

Coding Style Guide – Ein kleiner Auszug

Die in diesem Auszug gegebenen Hinweise zum Stil sind ausschließlich Hinweise für Code-Elemente, die Sie im Vorkurs Informatik kennengelernt haben. Erweitern Sie Ihr Wissen durch Nachlesen in den oben genannten Dokumenten **selbstständig**, sobald neue Elemente in der Vorlesung vorgestellt werden.

1. Sprechende Namen für Variablen/Funktionen mit Underscore-Trennung

```
RICHTIG:
int alter_hund;

FALSCH:
int avh;
int Altervonhund;
```

2. Ein Tab einrücken pro Verschachtelungstiefe (== 1x Tab-Taste pro {}-Block)
   (Tiefe: 8 Leerzeichen pro Tab)
   (Tipp: Leicht einstellbar in z.B. Notepad++ oder Visual Studio Code)

```
RICHTIG:
if (a == b) {
        for (...) {
                x = 4;
                y = 3;
        }
        z = 6;
} else {
        a = 8;
}

FALSCH:
if (a == b) {
for (...) {
x = 4;
y = 3;
}
z = 6;
}
else { a = 8; }
```

3. Geschweifte Klammer bei Funktionen unterhalb der Signatur-Zeile

```
RICHTIG:
int meine_funktion()
{
  ...
}

FALSCH:
int meine_funktion() {
  ...
}
```

4. Das „else“ passend setzen

```
RICHTIG:
if (a == b) {
  ...
} else {
  ...
}
```

5. Leerzeichen nach Kommata

```
RICHTIG:
(a, b, c)

FALSCH:
(a,b,c)
```

6. Kein inneres Leerzeichen vor/nach runden Klammern

```
RICHTIG:
(a == b)

FALSCH:
( a == b )
```

7. Leerzeichen vor/nach zweistelligen Operatoren

```
RICHTIG:
(a + b)

FALSCH:
(a+b)
```

8. Leerzeichen vor/nach den Bedingungen von for/while/if

```
RICHTIG:
while (x != 0) {

FALSCH:
while(x != 0){
```

9. Sinnvoll kommentieren:

- So wenig wie möglich, aber so viel wie nötig!
- Man sollte den Quellcode dank sprechender Namen bereits flüssig lesen können.
- Komplexe Zusammenhänge lassen sich mit Kommentaren darstellen.

```
FALSCH:
int x = 5; /* weist x den Wert 5 zu */
int y = 7; /* weist y den Wert 7 zu */
int z = x + y; /* berechnet Summe */
```

10. Variablen so lokal wie möglich anlegen

```
RICHTIG:
int main(void)
{
        int x = 4 + 5;
        printf("%i", x);
        return 0;
}

FALSCH:
int x;
int main(void) {
        x = 4 + 5;
        printf("%i", x);
        return 0;
}
```

Aufgabe 0 * *(Coding Conventions)*

Diese Aufgabe ist nicht abzugeben. Sie wird interaktiv in der ersten Übungsstunde erarbeitet. Sie können Sie aber natürlich vorher schon selbst versuchen.

Verbessern Sie die Formatierung der nachfolgenden C-Funktion, so dass sie konform zu den Stil-Richtlinien aus dem Auszug auf diesem Übungsblatt ist.

```
int a;
int AddOrMultiply(int number1,int number2){
if(number1<0){number1=number1*(-2);/*multiply number1 by -2*/
a=number1+number2 ;/*add number1 to number2*/
} /*endif*/
else {
a=number1 *number2; /* multiply number1 with number2*/
}/*endelse*/
a=a+1;
return a;}
```

Aufgabe 1 * *(Wiederholung Vorkurs: Einfache Syntaxfehler)*

Die folgenden C-Programme sollten direkt nach dem Kompilieren auf der Kommandozeile ausführbar sein. Leider haben sich einige Fehler eingeschlichen! Untersuchen Sie die zugehörige(n) Fehlermeldung(en) des Compilers und schreiben Sie jeweils eine fehlerbereinigte Version des Programms.

a)

```
int Main(void)
{
        return 0
}
```

b)

```
int main(void)
{
        printf("Hallo");
        return 0;
```

c)

```
#include <stdio.h>

int main(void)
{
        x = 3;
        printf("%i"; x);
        return 0;
}
```

d)

```
#include "studio.h"
#include "ctype.h"

int main(void)
{
        int n;
        5 = n;
        printf("%i\n", n);
        return 0;
}
```

Aufgabe 2 * *(Wiederholung Vorkurs: Einfache Programme mit `int`/`char`/`double`)*

In dieser Aufgabe soll in jeder Teilaufgabe ein C-Programm erstellt und dazu die in Worten beschriebenen Anweisungen in C-Anweisungen umformuliert werden.
Beispielsweise soll die in Worten beschriebene Anweisung

- Deklarieren Sie eine Variable `x` vom Typ `int`.

in die C-Anweisung

- `int x;`

überführt werden.

Erstellen Sie für jede Teilaufgabe jeweils eine C-Datei mit einer eigenen `main`-Funktion. Kompilieren Sie Ihre Programme mit den Compilerschaltern `-ansi -pedantic -Wall -Wextra` und führen Sie sie aus (jeweils über die Kommandozeile).

a)

- Deklarieren Sie drei `int`-Variablen `k`, `m` und `n`.
- Weisen Sie `k` den Wert `-1` zu.
- Erzeugen Sie mit `rand` eine ganze (Pseudo-)Zufallszahl und weisen Sie `m` den Wert der Zufallszahl zu.
- Weisen Sie `n` den Wert der Multiplikation von `k` und `m` zu.
- Geben Sie den Wert von `n` aus.

b)

- Deklarieren Sie eine `int`-Variable `n`.
- Weisen Sie ihr den Wert `INT_MIN` zu.
- Verringern Sie dann den Wert von `n` um `1`.
- Geben Sie den Wert von `n` aus.

c)

- Deklarieren Sie eine `char`-Variable `c`.
- Weisen Sie ihr den Wert `'?'` zu.
- Geben Sie den Wert von `c` als Zeichen aus.
- Geben Sie den Wert von `c` als ganze Zahl in einer neuen Zeile aus.

d)

- Deklarieren Sie eine `double`-Variable `x`.
- Weisen Sie ihr den Wert `4.5 * 10e2` zu.
- Geben Sie den Wert von `x` in Festkommaschreibweise aus.
- Geben Sie den Wert von `x` in Fließkommaschreibweise in einer neuen Zeile aus.

e) Alle Ausgaben in dieser Teilaufgabe sollen in Festkommaschreibweise mit 10 Nachkommastellen erfolgen.

- Deklarieren Sie eine `double`-Variable `x`.
- Deklarieren Sie eine `int`-Variable `k`.
- Weisen Sie beiden Variablen den Wert `3.9` zu.
- Geben Sie den Wert beider Variablen aus.
- Dividieren Sie beide Variablen jeweils durch `2`.
- Geben Sie in einer neuen Zeile den Wert beider Variablen aus.

f) Alle Ausgaben in dieser Teilaufgabe sollen in Festkommaschreibweise mit 10 Nachkommastellen erfolgen.

- Geben Sie den Wert von `7` dividiert durch `4` aus.
- Geben Sie den Wert von `7` dividiert durch `4.0` aus.

Erklären Sie die auftretende Warnung.

g) Geben Sie folgende Werte jeweils in einer eigenen Zeile in Festkommaschreibweise aus, indem Sie jeweils geeignete Funktionen aus `math.h` benutzen:

- den Cosinus von `2.0`
- den natürlichen Logarithmus von `250.0`

Hinweis: Auf manchen Systemen kann es notwendig sein, den Quellcode mit dem Compiler-Schalter `-lm` zu kompilieren, um die in `math.h` deklarierten Funktionen verwenden zu können.

h)

- Deklarieren Sie drei `int`-Variablen `a`, `b` und `c`.
- Erzeugen Sie mit `rand` drei Zufallszahlen und weisen Sie `a`, `b` und `c` jeweils eine der Zufallszahlen als Wert zu.
- Multiplizieren Sie die Summe von `a` und `b` mit `c` und geben Sie das Ergebnis aus.
- Addieren Sie `c` zum Quotienten von `a` und `b` und geben Sie das Ergebnis aus.

i)

- Deklarieren Sie eine `int`-Variable `n`.
- Deklarieren Sie eine `double`-Variable `x`.
- Erzeugen Sie mit `rand` eine ganze Zufallszahl und weisen Sie `n` den Wert der Zufallszahl zu.
- Negieren Sie den Wert von `n`.
- Casten Sie den Wert von `n` auf `double`, dividieren Sie ihn durch `RAND_MAX` und weisen Sie `x` das Ergebnis dieser Rechnung als Wert zu.
- Geben Sie mit einer einzelnen `printf`-Anweisung die Werte von `n` und `x` jeweils in einer eigenen Zeile aus.

j)

- Deklarieren Sie eine `int`-Variable `n`.
- Deklarieren Sie eine `char`-Variable `c`.
- Erzeugen Sie mit `rand` eine ganze Zufallszahl und weisen Sie `n` den Wert der Zufallszahl zu.
- Berechnen Sie den Rest der ganzzahligen Division von `n` durch `128` und weisen Sie `c` das Ergebnis dieser Rechnung als Wert zu.
- Bestimmen Sie **ohne** Benutzung einer Bibliotheksfunktion, ob `c` eine Ziffer ist, und geben Sie im positiven Fall `1` aus.
- Bestimmen Sie **ohne** Benutzung einer Bibliotheksfunktion, ob `c` ein lateinischer Großbuchstabe ist, und geben Sie im positiven Fall `2` aus.
- Bestimmen Sie **mit** Benutzung einer Bibliotheksfunktion, ob `c` ein lateinischer Kleinbuchstabe ist, und geben Sie im positiven Fall `3` aus.
- Bestimmen Sie **mit** Benutzung einer Bibliotheksfunktion, ob `c` eine hexadezimale Ziffer ist, und geben Sie im positiven Fall `4` aus.
- Geben Sie `c` in einer neuen Zeile als Zeichen aus.
- Geben Sie den ASCII-Code von `c` in einer neuen Zeile aus.

Aufgabe 3 ** *(Wiederholung Vorkurs: Bedingungen und Schleifen)*

Erstellen Sie für jede Teilaufgabe jeweils eine C-Datei mit einer eigenen `main`-Funktion.

a) (*, 4 Minuten)
Schreiben Sie ein C-Programm, das in einer übersichtlichen Tabelle die ersten 100 natürlichen Zahlen und ihren quadrierten Wert ausgibt.

b) (**, 8 Minuten)
Schreiben Sie ein C-Programm, das eine ganze Zufallszahl `n` zwischen 0 und 127 (jeweils einschließlich) berechnet und dann für jede Zahl zwischen 0 und `n` (jeweils einschließlich) in einer eigenen Zeile die Zahl und `"Ja"` bzw. `"Nein"` ausgibt, je nachdem ob das zugehörige ASCII-Zeichen ein lateinischer Klein- oder Großbuchstabe ist oder nicht.

c) (***, 6 Minuten)
Betrachten Sie folgendes C-Programm und beschreiben Sie in zwei bis drei Sätzen das Verhalten des Programms. Orientieren Sie sich dabei an der Aufgabenstellung der Teilaufgabe a). Geben Sie außerdem eine mögliche Ausgabe des Programms an.

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

int main(void)
{
        int i;
        int k;
        int r;
        r = rand() % (6 - 2 + 1) + 2;
        for (i = 0; i < r; i++) {
                for (k = i; k >= 0; k--) {
                        printf("00");
                }
                printf("\n");
        }
        return 0;
}
```

Diskutieren Sie mit Ihrem Team über Ihre Lösung. Erst danach probieren Sie aus, ob Sie richtig liegen!

Aufgabe 4 *** *(Wiederholung Vorkurs: Eigene Funktionen und Felder)*

a) (**, 5 Minuten)
Erstellen Sie eine Funktion `int sum(int w[], int size)`, die die ersten `size` Komponenten von `w` aufsummiert und das Ergebnis zurückgibt.

Testen Sie Ihre Funktion in einem kurzen Hauptprogramm.

b) (**, 7 Minuten)
Implementieren Sie eine Funktion `int array_to_upper(char v[], char w[], int size)`, die zunächst überprüft, ob `v` nur lateinische Buchstaben enthält. Im positiven Fall werden alle Zeichen in `v` (wenn nötig) in Großbuchstaben konvertiert und in `w` am gleichen Index einfügt und 0 zurückgegeben. Enthält `v` Zeichen, die nicht lateinische Buchstaben sind, wird 1 zurückgegeben.

Testen Sie Ihre Funktion in einem Hauptprogramm mit folgenden Beispielfeldern und einem leeren Feld passender Länge:

- `'H', 'a', 'l', 'l', 'o'`
- `87, 101, 108, 116`
- `'I', 'n', 'f', 'o', 'r', 'm', 'a', 't', 'i', 'k', '1'`

c) (**, 7 Minuten)
Implementieren Sie eine Funktion `void manipulate_and_print(int v[], int size)`, die die ersten `size` Komponenten des übergebenen Feldes `v` durchläuft und für jede Komponente mit nicht-negativem Wert folgende Berechnung durchführt:

- Falls die `i`-te Komponente von `v` durch 3 teilbar ist, halbieren Sie die Zahl und geben diese in einer neuen Zeile auf der Kommandozeile aus.
- Falls die Division der `i`-ten Komponente durch 3 den Rest 1 ergibt, inkrementieren Sie diese und geben sie in einer neuen Zeile aus.
- Falls die Division der `i`-ten Komponente durch 3 den Rest 2 ergibt, negieren Sie diese, verringern ihren Wert dann um 2 und geben diesen in einer neuen Zeile aus.

Ist der Wert der Komponente negativ, soll eine leere Zeile ausgegeben werden.

Testen Sie Ihre Funktion in einem kurzen Hauptprogramm mit geeigneten Beispielen.

d) (***, 10 Minuten)
Implementieren Sie ein C-Programm, das den Inhalt eines `char`-Feldes in ein zweites (leeres) Feld kopiert und dabei alle vorkommenden Ziffern in den lateinischen Großbuchstaben an dem entsprechend-letzten Index im Alphabet umwandelt ('0' wird zu 'Z', '1' zu 'Y' usw.).
Geben Sie danach die Inhalte beider Felder jeweils in einer eigenen Zeile auf Kommandozeile aus.

Hinweis: Schreiben Sie eine Methode

```
void copy_and_transform_digits(char v[], char w[], int size)
```

welche die oben beschriebene Funktionalität erfüllt und testen Sie diese Funktion in einem kleinen, selbst geschriebenen Hauptprogramm.