Universität Augsburg
Institut für Informatik

Übung zur Vorlesung Informatik 1 WS 2020/21
Fakultät für Angewandte Informatik
Lehrprofessur für Informatik
Dr. Martin Frieb, Johannes Metzger 02.11.2020

Lösungsvorschlag zu Übungsblatt 1

Abgabe spätestens bis: 09.11.2020 10:00 Uhr

- Dieses Übungsblatt soll in den in der Übungsgruppe festgelegten Teams abgegeben werden (Einzelabgaben sind erlaubt, falls noch keine Teamzuteilung erfolgt ist).
- 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 zur Abgabe der Übungsblätter

1. Abgabeort: Die Abgabe erfolgt digital als Upload in der Digicampus-Veranstaltung der zugeteilten Übungsgruppe.
2. Abgabeformat Quellcode: Alle Quellcode-Dateien sind als `*.c`/`*.h`-Dateien abzugeben.
3. Abgabeformat Sonstige Aufgaben: Alle weiteren Aufgaben sind als `*.pdf` abzugeben. Im einfachsten Fall kann es sich hierbei um einen gut lesbaren Scan Ihrer handschriftlichen Lösung handeln.

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*
  https://port70.net/~nsz/c/c89/c89-draft.html
- *C89 Rationale* (Begleitdokument mit Erklärungen)
  https://port70.net/~nsz/c/c89/rationale/title.html
- *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.

Compiler-Schalter

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

```
-Wall -Wextra -ansi -pedantic
```

Achten Sie darauf, dass Ihr Programm trotz der Verwendung dieser Schalter keine Fehler-/Warnmeldungen beim Kompilieren erzeugt.

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):

- *Notepad++* [Windows]
  https://notepad-plus-plus.org/
- *Atom* [Windows, macOS, GNU/Linux]
  https://atom.io/

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.

Die von Ihnen geschriebenen Programme laden Sie zur Abgabe ausschließlich als Quellcode/Header-Dateien (Dateiendungen `.c` und `.h`) in der Digicampus-Veranstaltung Ihrer Übungsgruppe hoch.

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 Atom)

```
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.

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;}
```

Lösung:

```
/*
   Calculates the sum of -2x and y if x < 0,
   or the product of x and y if x >= 0.

   Returns the result of this operation incremented by 1.
*/
int add_or_multiply(int x, int y)
{
        int result;
        if (x < 0) {
                x = x * (-2);
                result = x + y;
        } else {
                result = x * y;
        }
        result = result + 1;
        return result;
}
```

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
}
```

Lösung:

```
int main(void)
{
        return 0;
}
```

b)

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

Lösung:

```
#include <stdio.h>

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

c)

```
#include <stdio.h>

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

Lösung:

```
#include <stdio.h>

int main(void)
{
        int 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;
}
```

Lösung:

```c
#include <stdio.h>

int main(void)
{
        int n;
        n = 5;
        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 ein Kommandozeilen-Programm).

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 Addition von `k` und `m` zu.
- Geben Sie den Wert von `n` aus.

Lösung:

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

int main(void)
{
        int k;
        int m;
        int n;
        k = 1;
        m = rand();
        n = k + m;
        printf("%i", n);
        return 0;
}
```

b)

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

Lösung:

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

int main(void)
{
        int n;
        n = INT_MAX;
        n = n + 1;
        printf("%i", n);
        return 0;
}
```

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.

Lösung:

```c
#include <stdio.h>

int main(void)
{
        char c;
        c = '!';
        printf("%c", c);
        printf("\n%i", c);
        return 0;
}
```

d)

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

Lösung:

```c
#include <stdio.h>

int main(void)
{
        double x;
        x = 10e2;
        printf("%f", x);
        printf("\n%e", x);
        return 0;
}
```

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

- Deklarieren Sie eine `double`-Variable `x`.
- Deklarieren Sie eine `int`-Variable `n`.
- Weisen Sie beiden Variablen den Wert `7.9` 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.

Lösung:

```c
#include <stdio.h>

int main(void)
{
        double x;
        int n;
        x = 7.9;
        n = (int) 7.9;
        printf("%.10f %.10f\n", x, (double) n);
        x = x / 2;
        n = n / 2;
        printf("%.10f %.10f", x, (double) n);
        return 0;
}
```

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

- Geben Sie den Wert von `9` dividiert durch `4` aus.
- Geben Sie den Wert von `9` dividiert durch `4.0` aus.

Erklären Sie die auftretende Warnung.

Lösung:

```c
#include <stdio.h>

int main(void)
{
        printf("%.10f\n", 9 / 4);
        printf("%.10f", 9 / 4.0);
        return 0;
}
```

Die Warnung weist darauf hin, dass es sich bei der ersten Division um eine ganzzahlige Division handelt, deren Ergebnis eine ganze Zahl ist. Der Platzhalter `%f` erwartet allerdings eine Zahl vom Typ `double`.

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

- den Sinus von `1.0`
- den natürlichen Logarithmus von `150.0`

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.

Lösung:

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

int main(void)
{
        printf("%f\n", sin(1.0));
        printf("%f", log(150.0));
        return 0;
}
```

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.
- Dividieren Sie die Summe von `a` und `b` durch `c` und geben Sie das Ergebnis aus.
- Addieren Sie `a` zum Quotienten von `b` und `c` und geben Sie das Ergebnis aus.

Lösung:

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

int main(void)
{
        int a;
        int b;
        int c;
        a = rand();
        b = rand();
        c = rand();
        printf("%i\n", (a + b) / c);
        printf("%i", a + b / c);
        return 0;
}
```

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.
- 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.
- Negieren Sie den Wert von `x`.
- Geben Sie mit einer einzelnen `printf`-Anweisung die Werte von `n` und `x` jeweils in einer eigenen Zeile aus.

Lösung:

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

int main(void)
{
        int n;
        double x;
        n = rand();
        x = (double) n / RAND_MAX;
        x = -x;
        printf("%i\n%f", n, x);
        return 0;
}
```

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 Kleinbuchstabe ist, und geben Sie im positiven Fall `2` aus.
- Bestimmen Sie mit Benutzung einer Bibliotheksfunktion, ob `c` ein lateinischer Großbuchstabe ist, und geben Sie im positiven Fall `3` aus.
- Bestimmen Sie mit Benutzung einer Bibliotheksfunktion, ob `c` ein Zwischenraumzeichen 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.

Lösung:

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

int main(void)
{
        int n;
        char c;
        n = rand();
        c = n % 128;
        /*
                Ausgeben => printf
                Zurückgeben => return
        */
        if (c >= '0' && c <= '9') {
                printf("1");
        } else if (c >= 'a' && c <= 'z') {
                printf("2");
        } else if (isupper(c)) {
                printf("3");
        } else if (isspace(c)) {
                printf("4");
        }
        printf("\n%c", c);
        printf("\n%i", c);
        return 0;
}
```

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

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

a) (**, 8 Minuten)
Schreiben Sie ein C-Programm, das eine ganze Zufallszahl zwischen 0 und 1000 (jeweils einschließlich) berechnet und dann zeilenweise in aufsteigender Reihenfolge alle Teiler dieser Zahl ausgibt.

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

int main(void)
{
        int i;
        int r;
        r = rand() % 1001;
        for (i = 1; i <= r / 2; ++i) {
                if (r % i == 0) {
                        printf("%i\n", i);
                }
        }
        printf("%i\n", r);
        return 0;
}
```

b) (*, 4 Minuten)
Schreiben Sie ein C-Programm, das in einer übersichtlichen Tabelle die Dezimal-, Oktal- und Hexadezimaldarstellungen der ganzen Zahlen zwischen 0 und 1000 (jeweils einschließlich) ausgibt.

```c
#include <stdio.h>

int main(void)
{
        int i;
        for (i = 0; i <= 1000; ++i) {
                printf("%i\t %o\t %x\n", i, i, i);
        }
        return 0;
}
```

c) (**, 6 Minuten)
Schreiben Sie ein C-Programm mit folgendem Verhalten:
Es wird zunächst eine Zufallszahl `r` im Intervall $[6; 14]$ berechnet. Anschließend geben Sie mit zwei ineinander verschachtelten `for`-Schleifen `r` Zeilen aus, wobei die erste Zeile aus `r` hintereinander geschriebenen Nullen besteht und die Anzahl der Nullen mit jeder weiteren Zeile um eins abnimmt.

Beispielhafte Ausgabe für $r = 6$:

```
000000
00000
0000
000
00
0
```

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

int main(void)
{
        int i;
        int k;
        int r;
        r = rand() % (14 - 6 + 1) + 6;
        for (i = r; i >= 1; i--) {
                for (k = 0; k < i; k++) {
                        printf("0");
                }
                printf("\n");
        }
        return 0;
}
```

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

a) (**, 5 Minuten)
Erstellen Sie eine Funktion `void print_letters(char w[], int size)`, die die ersten `size` Komponenten von `w` mit Komma getrennt auf der Kommandozeile ausgibt. Achten Sie darauf, dass Sie nicht zu viele Kommas ausgeben.

Testen Sie Ihre Funktion in einem kurzen Hauptprogramm.

```c
#include <stdio.h>

void print_letters(char w[], int size);

int main(void)
{
        char feld[] = {'i', 'n', 'f', 'o', '1'};
        print_letters(feld, 5);
        return 0;
}

void print_letters(char w[], int size)
{
        int i;
        for (i = 0; i < size; i++) {
                printf("%c", w[i]);
                if (i < size - 1) {
                        printf(",");
                }
        }
}
```

b) (**, 5 Minuten)
Implementieren Sie eine Funktion `int compare_arrays(int v[], int w[], int size_v, int size_w)`, die vergleicht, ob `v` und `w` identische Komponenten an gleichen Indizes haben und im positiven Fall 0 zurückgibt.

Testen Sie Ihre Funktion in einem Hauptprogramm mit folgenden Beispielfeldern:

- 1, 4, 6, 2, 7, 5, 8
- 1, 4, 2, 6, 7, 5, 8
- 1, 4, 6, 2

```c
#include <stdio.h>

int compare_arrays(int v[], int w[], int size_v, int size_w);

int main(void)
{
        int numbers1[] = {1, 4, 6, 2, 7, 5, 8};
        int numbers2[] = {1, 4, 2, 6, 7, 5, 8};
        int numbers3[] = {1, 4, 6, 2};
        printf("numbers1 and numbers2: %i\n", compare_arrays(numbers1, numbers2, 7, 7));
        printf("numbers1 and numbers3: %i\n", compare_arrays(numbers1, numbers3, 7, 4));
        printf("numbers1 and numbers1: %i\n", compare_arrays(numbers1, numbers1, 7, 7));
        return 0;
}

int compare_arrays(int v[], int w[], int size_v, int size_w)
{
        int i;
        if (size_v != size_w) {
                return -1;
        }
```

```
        for (i = 0; i < size_v; i++) {
                if (v[i] != w[i]) {
                        return -2;
                }
        }
        return 0;
}
```

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 `i` folgende Berechnung durchführt:

- Falls die `i`-te Komponente von `v` negativ ist, verdoppeln Sie die Zahl und geben diese in einer neuen Zeile auf der Kommandozeile aus.
- Falls die `i`-te Komponente nicht-negativ und gerade ist, inkrementieren Sie diese und geben sie in einer neuen Zeile aus.
- Falls die `i`-te Komponente nicht-negativ und ungerade ist, negieren Sie diese und geben sie in einer neuen Zeile aus.

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

```
#include <stdio.h>

void manipulate_and_print(int v[], int size);

int main(void)
{
        int numbers[] = {1, -4, 6, 2, -7, 5, 0};
        manipulate_and_print(numbers, 7);
        return 0;
}

void manipulate_and_print(int v[], int size)
{
        int i;
        for (i = 0; i < size; i++) {
                if (v[i] < 0) {
                        printf("%i\n", 2 * v[i]);
                } else if (v[i] % 2 == 0) {
                        printf("%i\n", ++v[i]);
                } else {
                        printf("%i\n", -v[i]);
                }
        }
}
```

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 Kleinbuchstaben an dem entsprechenden Index im Alphabet umwandelt ('0' wird zu 'a', '1' zu 'b' usw.).
Geben Sie danach die Inhalte beider Felder jeweils in einer eigenen Zeile auf Kommandozeile aus.
Überlegen Sie sich selbst geeignete Funktionen und deren Prototypen.
Testen Sie Ihre Funktionen schließlich in einem kleinen Hauptprogramm.

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

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

int main(void)
{
        char alt[] = {'i', 'n', 'f', 'o', '1', '-', '2', '0', '2', '0', '/', '2', '1'};
        char neu[13];
        copy_and_transform_digits(alt, neu, 13);
        print(alt, 13);
        print(neu, 13);
        return 0;
}

void copy_and_transform_digits(char v[], char w[], int size)
{
        int i;
        for (i = 0; i < size; ++i) {
                if (isdigit(v[i])) {
                        w[i] = v[i] - '0' + 'a';
                } else {
                        w[i] = v[i];
                }
        }
}

void print(char w[], int size)
{
        int i;
        for (i = 0; i < size; ++i) {
                printf("%c", w[i]);
        }
        printf("\n");
}
```