Arbeitsblätter zum Praktikum Programmieren

HS Ravensburg-Weingarten

25. September 2015

1 Allgemeines zum Praktikum

Für die Übungen bilden Sie bitte 2-er Gruppen, und zwar möglichst so, dass in jeder Gruppe mindestens ein(e) Teilnehmer(in) bereits eine gewisse Erfahrung im Umgang mit Computern hat. Machen Sie sich bitte zunächst mit Linux vertraut (s. Abschnitt 4). Sie können die Übungsaufgaben zu Hause oder während der Praktikumstermine an der Hochschule bearbeiten. Sie sollten die Aufgaben vor den jeweiligen Praktikumsterminen vorbereiten: Lösungsweg erarbeiten, ggf. Ablaufdiagramm(e) zeichnen, Testfälle beschreiben (Eingabe und erwartetes Ergebnis).

Bei Fragen, Unklarheiten etc. wenden Sie sich bitte an einen Betreuer (Tutor oder Dozent). Sie müssen die Aufgaben nicht "irgendwie hinbekommen", sondern gründlich verstehen, wenn Sie die Prüfung bestehen wollen.

1.1 Geforderte Ergebnisse

Zu jeder Übungsaufgabe geben Sie bitte pro Gruppe eine geheftete Dokumentation ab, ohne Einband, keine Mappe. Die Dokumentation hat folgenden Aufbau:

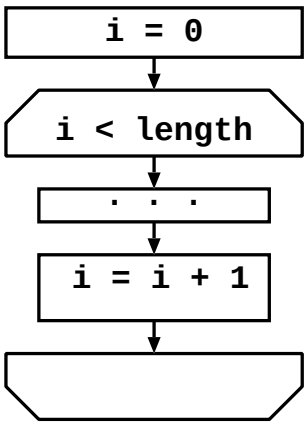
- Deckblatt gemäß Beispiel (s. S. 4) mit Übungsnummer, Namen, Matrikelnummern, Studiengang, Datum und Korrektur-Abschnitt
- Beschreibung des Lösungsweges in folgenden Formen:
  - Prosa, Telegramm-Stil
  - ggf. Ablaufdiagramm(e)
- Beschreibung der Testfälle, mit denen das Programm getestet wurde, einschließlich verwendeter Eingabedaten und Programm-Ausgaben. Bitte immer konkrete Tests angeben, keine allg. Aussagen.
- Programmlisting (Halten Sie bitte die in der Vorlesung besprochenen Stil-Regeln ein.)
- ggf. Aufrufhierachie aller von Ihnen geschriebenen Funktionen.
- ggf. Modulhierachie aller von Ihnen geschriebenen Module.

Die mit ggf. gekennzeichneten Anteile sind nicht für alle Aufgaben verbindlich. Bitte richten Sie sich nach den Hinweisen bei den einzelnen Aufgaben.

Sie finden in Büchern oder im Internet sowohl für Ablaufdiagramme als auch für die Formatierung des Quellcodes unterschiedliche Vorgaben. Für das Praktikum sind folgende Vorgaben verbindlich:

Ablaufdiagramm: Fügen Sie bitte immer einen Block ein, in dem die verwendeten Variablen definiert werden. Verwenden Sie für Schleifen nur das Schleifensymbol. Schreiben Sie in den Kopf der Schleife nur den logischen Ausdruck, der die Schleife steuert. Das rechts stehende Ablaufdiagramm zeigt folgende Schleife: **for**(i = 0; i < length; i = i + 1){ . . . } Geben Sie auch immer ein Start und ein Ende-Symbol an. Ein Beispiel für ein vollständigs Ablaufdiagramm finden Sie auf Seite 5.

Einrückung im Quellcode Verwenden Sie für die Einrückung Leerzeichen, keine Tabulatoren. Eine Stufe der Einrückung ist immer 4 Zeichen breit. Verwenden Sie am besten einen Editor, der die Zeilen automatisch passend einrückt z. B. emacs. mit folgenden Einstellungen:
~/.gnu−emacs: (setq c−basic−offset 4) und (setq indent−tabs−mode nil).



1.2 Hilfsmittel für die Dokumentation

Für die Textverarbeitung empfehle ich LaTeX für Abbildungen tgif bzw. xfig. Ansonsten können Sie auch OpenOffice, Koffice, StarOffice, Abiword, ... verwenden.

Meine Empfehlung: Schauen Sie sich LaTeX an. Dieses Satzprogramm ist m. E. für größere Dokumente wie z. B. Praktikumsberichte oder Bachelor-Arbeiten wesentlich besser geeignet als alle WYSIWYG-Programme. Es ist unter vielen Betriebssystemen verfügbar, u. a. für Linux, Windows und MacOS. Im Rahmen dieses Praktikums können Sie sich in LaTeX einarbeiten. Hilfestellung finden Sie auf dem Austauschverzeichnis

```
.../PUB/DOZ/Stud/Zeller/Schreiben
```

Ich empfehle Ihnen auch, die Hinweise in der Datei `schreibmuster.pdf` zu beherzigen.

1.3 Korrekturen

Sie finden den mehr oder weniger aktuellen Stand der Korrekturen im E-Learning-System der Hochschule: `https://www.elearning.hs-weingarten.de`

Wenn Sie Ihre Dokumentation überarbeiten müssen, dann heften Sie bitte die neuen Blätter an die bisherige Dokumentation an. Geben Sie also bitte die ursprüngliche Fassung und die verbesserten Seiten ab. Die Seiten, die Sie nicht verändert haben, müssen/sollen Sie nicht neu audrucken.

1.4 Vergabe von Scheinen

Sie erhalten einen Schein, wenn Sie die Dokumentation (s. o.) rechtzeitig abgeben und Ihre Programme am Rechner vorführen und erklären können. Der Dozent, der das Praktikum betreut, setzt die Abgabetermine fest.

Vorführung am Rechner: Der Dozent bzw. Betreuer oder Tutor lässt sich einige Programme seiner Wahl vorführen *und erklären*. Dabei kommen nur Programme aus den Übungsaufgaben 4 - 11 in Frage. Die Programme müssen bei "vernünftiger" Eingabe fehlerfrei laufen. Es müssen nicht alle möglichen Fehleingaben korrekt verarbeitet werden, allerdings muss das Verhalten des Programms bei Fehleingaben dokumentiert sein (in Form von Testfällen).

Praktikum Programmieren, WS 2010

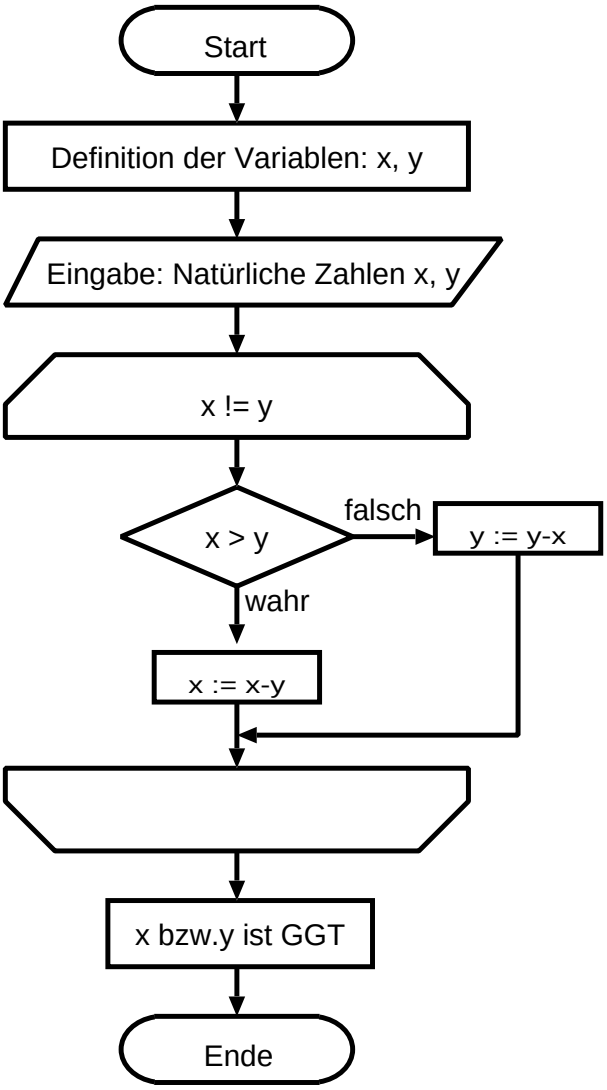
Übung-Nr. *42*
Aufgabe: *Berechnung des größten gemeinsamen Teilers*

Bearbeiter: *F. Carly* Matr. Nr. *123456* Studiengang: AI
L. Gerstner Matr. Nr. *654321* Studiengang: EI

1 Beschreibung der Lösung

Das Programm verwendet den Algorithmus von Euklid. Es liest zunächst zwei Zahlen ein. In einer Schleife vergleicht das Programm die Zahlen. Sind beide Zahlen gleich, so ist dieser Wert der GGT. Ansonsten zieht das Programm die kleinere Zahl von der grüßeren ab und speichert das Ergebnis in der Variable der größeren Zahl. Hier schließt sich die Schleife. Die Schleife wird solange durchlaufen, bis beide Zahlen den gleichen Wert besitzen.

2 Ablaufdiagramm



3 Quellcode

```
/*************************************************
 * Modul ggtmod.c :
 *
 * Aufgabe: Das Programm berechnet den groessten
 *          gemeinsamen Teiler von zwei Zahlen.
 *
 *************************************************/

#include <stdio.h>

int main(void){
    unsigned int num1;
    unsigned int num2;
    unsigned int num1Mod;
    unsigned int num2Mod;

    printf("Berechnung GGT, bitte erste Zahl eingeben: ");
    scanf("%u", &num1);
    printf("Berechnung GGT, bitte zweite Zahl eingeben: ");
    scanf("%u", &num2);

    num1Mod = num1;
    num2Mod = num2;

    while (num1Mod != num2Mod){
        if (num1Mod > num2Mod){
            num1Mod = num1Mod - num2Mod;
        }else{
            num2Mod = num2Mod - num1Mod;
        }
    }

    printf("\n Der GGT der Zahlen %u und %u", num1, num2);
    printf(" lautet: %u \n\n", num1Mod);

    return 0;
}
```

4 Testfälle

Eingabe		Ausgabe/Wirkung				
12,	8	Zahl1:	12,	Zahl2:	8,	GGT: 4
15,	0	Endlosschleife				
70000,	120098727	Zahl1:	70000,	Zahl2:	120098727	GGT: 7
6,	-2	Zahl1:	6,	Zahl2:	4294967294,	GGT: 2

4 Die Entwicklungsumgebung – Editor und Compiler unter Linux

4.1 Vorbereitung

Empfohlene Distribution: Lubuntu/Kubuntu/*buntu; Sie können aber auch jede andere Distribution verwenden. Für die Paketverwaltung d. h. Installation und De-Installation von Programmen unter *buntu empfehle ich `synaptic`.

Installation: Paket "Entwicklung" bzw. "build essentials" installieren (mindestens gcc und vim bzw. emacs).

4.1.1 Konsole bzw. xterm öffnen

Start-Menü →System →(Kommandozeilen →) konsole, terminal, xterm oder bash

Die folgenden Abschnitte beschreiben einige nützliche Befehle. Ein Befehl ist ein Wort, das Sie in ein Terminal-Fenster eingeben. Der Befehl wird ausgeführt, wenn Sie die `<return>`-Taste drücken.

4.1.2 Verzeichnisse anlegen

Legen Sie für jedes Programm ein Verzeichnis an:

```
mkdir ProgPraktikum
cd ProgPraktikum
mkdir MultiplikationsTabelle
mkdir AsciiTabelle
mkdir DezimalDual
mkdir Taschenrechner
     :
     :
mkdir Binomialkoeffizient
```

4.1.3 Programm erstellen

```
cd MultiplikationsTabelle
```

Mit emacs oder vim die Quell-Code Datei erstellen. Die Bedienung von `emacs` und `vim` entnehmen Sie am besten einer sogenannten Quick-Reference-Card. Fragen Sie bei der Suchmaschine Ihrer Wahl nach "emacs quick reference" oder "vim quick reference".

Für die ganz ungeduldigen: Mit der Maus in das Emacs-Fenster klicken. Tastenkombination "Strg-h" drücken, anschließend t drücken. Das Emacs-Tutorial erscheint. Achtung: das Tutorial spricht von Control-Key Ctrl, Ctl bzw. C . . . auf den gängigen PC-Tastaturen ist das die Taste "Strg". Die "Meta"-Taste – oft nur mit "M" bezeichnet – ist auf den gängigen PC-Tastaturen die Taste "ESC".

Die Emacs-Doku enthält eine Reference-Card mit den wichtigsten Befehlen und ein Tutorial (TUTORIAL.de). Unter Debian Linux: /usr/share/emacs/23.2/etc/ . . .

4.1.4 Übersetzen und ausführen

Ein Quellprogramm wird mit folgendem Befehl in ein ausführbares Programm übersetzt[1]:

`gcc -g -std=c99 -Wall -o` *ausgabeDatei eingabeDatei*

Die Eingabedatei muss/sollte die Endung .c besitzen. Die Ausgabedatei sollte den gleichen Namen ohne Endung erhalten.

Also z. B. `gcc -g -std=c99 -Wall -o helloWorld helloWorld.c`

Falls die Bibliothek math.h für mathematische Funktionen eingebunden wird, muss der Befehl zum übersetzen `-lm` enthalten (library math):

gcc -lm -g -std=c99 -Wall -o *ausgabeDatei eingabeDatei*

Wenn sich das Programm übersetzen lässt, so erstellt gcc eine Datei *ausgabeDatei*, die das ausführbare Programm enthält.

Aufruf des Programms: *ausgabeDatei* bzw. *./ausgabeDatei*

4.1.5 Debugging

Ein Debugger mit graphischer Oberfläche: `kdbg &` oder `nemiver &`

Unter File→Executable die Datei *ausgabeDatei* laden (ausführbares Programm laden, nicht den Quell-Code!). Anschließend per Maus den/die Breakpoints setzen und das Programm starten.

Die ausführbare Datei (Executable) kann auch beim Aufruf als Argument angegeben werden, z. B. `kdbg helloWorld &` bzw. `nemiver helloWorld &`

4.2 Integrierte Entwicklungsumgebungen (IDE)

Für die Aufgaben des Programmierpraktikums ist eine IDE etwas überdimensioniert und lenkt vom eigentlichen Thema eher ab. Falls Sie sich in eine IDE einarbeiten möchten, können Sie aus einer ganzen Reihe kostenloser und kommerzieller Produkte wählen, z. B. NetBeans, CodeBlocks, Eclipse, Kdevelop (nur Linux), xCode (Mac OS, iOS) VisualStudio (Windows), Borland C++ Builder

[1] Optionen: `-g` für Debugging-Code, `-std=c99` für ANSI-C99 Sprachstandard `-Wall` für "alle Warnungen" und `-o` für "Name der Ausgabedatei".

Übungsaufgaben

Bei Ihren Ausarbeitungen orientieren Sie sich bitte an der Vorlage auf Seite 4 ff.

Alle Ausarbeitungen müssen grundsätzlich folgende Elemente enthalten: Beschreibung der Lösung, Quell-Code und Testfälle. Wann die Elemente Ablaufdiagramm und Aufrufhierarchie gefordert sind, entnehmen Sie bitte den einzelnen Aufgabenstellungen.

Übung 1: Multiplikationstabelle

Schreiben Sie ein Programm, das eine Multiplikationstabelle ausgibt:

1	2	3	4	...
2	4	6	8	...
3	6	9	12	...
:	:	:	:	
:	:	:	:	

Die Anzahl der Zeilen und die Anzahl der Spalten soll vom Nutzer eingegeben werden können. D.h. das Programm fragt zunächst nach der Anzahl der Spalten, dann nach der Anzahl der Zeilen. Anschließend zeigt das Programm die Tabelle am Bildschirm an.

Achtung: Schreiben Sie zwei Versionen diese Programms: Eine Version mit For-Schleifen und eine Version mit While-Schleifen.

Hinweis: Schreiben Sie erst ein Programm, das genau eine Zeile ausgibt. überlegen Sie sich, wie sie dieses Programm erweitern müssen, so dass es mehrere Zeilen ausgibt.

Ergebnis: Ausarbeitung mit Beschreibung der Lösung, Quell-Code und Testfälle.

Übung 2: ASCII-Tabelle

Schreiben Sie ein Programm, das eine ASCII-Tabelle der Zahlen von 32 - 127 in 4 Doppel- Spalten ausgibt. Zu jeder Zahl zwischen 32 und 127 soll der entsprechende ASCII-Character angezeigt werden:

```
32      56 8    80 P   104 h
33 !    57 9    81 Q   105 i
34 "    58 :    82 R   106 j
35 #    59 ;    83 S   107 k
36 $    60 <    84 T   108 l
. . .
. . .
55 7    79 O   103 g   127
```

Hinweis: Sie können das gleiche Grundmuster, wie in Aufgabe **1** verwenden. Überlegen Sie sich, welchen Wert Sie in der i-ten Zeile und der j-ten Spalte darstellen müssen. D. h. Sie müssen sich überlegen, wie Sie aus dem Spalten- und dem Zeilen-Index den Wert, berechnen den Sie darstellen wollen. Diesen Wert geben Sie einmal als Zahl und einmal als Buchstaben aus. Um einen Zahlenwert als Buchstaben auszugeben verwenden Sie die Formatanweisung `%c` in der Funktion `printf()`.

Sie können auch ein Programm mit nur einer Schleife schreiben – am besten erstellen Sie beide Varianten. Die Ausarbeitung ist natürlich nur für eine Version erforderlich.

Ergebnis: Ausarbeitung mit Ablaufdiagramm.

Übung 3: Umrechnung Dezimalzahl nach Dualzahl

Entwickeln Sie ein Programm, das eine positive ganze Zahl zur Basis 10 in eine Dualzahl (Zahl zur Basis 2) mit 16 Bit umrechnet. Das Programm soll eine Zahl einlesen und anschließend die entsprechende Dualzahl ausgeben.

Verwenden Sie zur Speicherung der Dualzahl ein Array mit 16 Einträgen. Jeder Eintrag ist eine Zahl mit Wert 0 oder mit Wert 1. Das erste Element des Arrays enthält das höchstwertigste Bit der Dualzahl, das letzte Element des Arrays enthält das niederwertigste Bit der Dualzahl.

Bitte beachten Sie, dass das Programm nur Zahlen umrechnen soll, die in 16 Bit darstellbar sind. Zahlenwerte, die nicht in 16 Bit dargestellt werden können, sollen nicht umgerechnet werden. In diesem Fall gibt das Programm eine Fehlermeldung aus.

Hinweis: Der Wert einer ganzen Zahl X zur Basis 10 berechnet sich wie folgt: Sei X dargestellt durch die Ziffernfolge $z_n z_{(n-1)} \dots z_1 z_0$ mit $z \in \{0, 1, \dots 9\}$, dann ist der Wert von X gegeben durch

$$X = \sum_{i=0}^{i=n} z_i * 10^i$$

Bsp.: $5243 = 5 * 10^3 + 2 * 10^2 + 4 * 10^1 + 3 * 10^0$

Der Wert einer ganzen Zahl Y zur Basis 2 berechnet sich wie folgt: Sei Y dargestellt durch die Ziffernfolge $d_n d_{(n-1)} \dots d_1 d_0$ mit $d \in \{0, 1\}$, dann ist der Wert von Y gegeben durch

$$Y = \sum_{i=0}^{i=n} d_i * 2^i$$

Bsp.: $1011 = 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0$
im Dezimalsystem also: $8 + 0 + 2 + 1 = 11$

Es gibt im Wesentlichen zwei Möglichkeiten zur Umrechnung: Von der niederwertigsten Stelle (Einerstelle) zur höchstwertigsten Stelle oder umgekehrt.

Ergebnis: Ausarbeitung mit Ablaufdiagramm.

Ein Hinweis für die folgenden Aufgaben: Wenn Sie die Funktion scanf() für die Eingabe verwenden, sollten Sie vor der Formatanweisung ein Leerzeichen setzen z. B. scanf(" %c", &operator); das Leerzeichen bewirkt, dass alle nicht druckbaren Zeichen, die noch im Eingabepuffer stehen, aus diesem gelöscht werden. Wichtig ist dies, wenn Sie einzelne Buchstaben einlesen.

Übung 4: Ein einfacher Taschenrechner

Entwerfen Sie ein Programm, das als einfacher Taschenrechner dient.

Das Programm soll drei Eingaben akzeptieren und daraus ein Ergebnis berechnen. Die erste Eingabe ist der Operator, ein Buchstabe aus `{+ - * / q}`. Die zweite Eingabe ist der erste Operand, eine Gleitkomma-Zahl. Die dritte Eingabe ist der zweite Operand, ebenfalls eine Gleitkomma-Zahl. Im Anschluß an die Berechnung wird das Ergebnis ausgegeben. Dieser Vorgang wiederholt sich solange, bis für den Operator das Zeichen `q` eingegeben wird.

Teilen Sie Ihr Programm in zwei Funktionen auf: Eine main-Funktion und eine eine Funktion, die aus den Eingaben ein Ergebnis berechnet. Die Signatur der Berechnungs-Funktion lautet dann:
`float` *calculate*`(char` *optr*`, float` *opr1*`, float` *opr2*`);`

Verwenden Sie innerhalb der Berechnungs-Funktion eine switch-Anweisung zur Auswahl der passenden Berechnungs-Anweisung. Das Ergebnis soll in der main-Funktion ausgegeben werden. Die Berechnungs-Funktion soll eine Fehlermeldung ausgeben, wenn sie kein (korrektes) Ergebnis berechnen kann. Die ist z. B. der Fall, wenn bei einer Division der Divisor den Wert 0 hat.

Bsp. für einen Programmlauf:

```
Eingabe: Operator <return> Operand1 <return> Operand2 <return>
Bitte Eingabe Operator (+, -, *, /, q): +
Bitte Eingabe Operand1: 4
Bitte Eingabe Operand2: 7
Ergebnis:    4.00 +    7.00 =   11.00


Eingabe: Operator <return> Operand1 <return> Operand2 <return>
Bitte Eingabe Operator (+, -, *, /, q): /
Bitte Eingabe Operand1: 23.21
Bitte Eingabe Operand2: 14.321
Ergebnis:   23.21 /   14.32 =    1.62


Eingabe: Operator <return> Operand1 <return> Operand2 <return>
Bitte Eingabe Operator (+, -, *, /, q): q
```

Ergebnis: Ein Ablaufdiagramm ist nicht für das gesamte Programm, sondern nur für die Berechnungs-Funktion gefordert. Stellen Sie auch die Aufrufhierarchie der Funktionen dar.

Übung 5: Ein Palindrom-Test

Ein Palindrom ist ein Wort, das vorwärts wie rückwärts gelesen die gleiche Buchstabenfolge besitzt; z.B. rotor, reliefpfeiler, otto, amanaplanacanalpanama (a man a plan a canal panama).

Schreiben Sie ein Programm, das ein Wort einliest (max. 63 Buchstaben) und prüft, ob es sich bei dem Wort um ein Palindrom handelt.

Bsp. für einen Programmlauf:

```
Palindrom-Test
Geben Sie bitte das zu pruefende Wort ein: reliefpfeiler
Das Wort "reliefpfeiler" ist ein Palindrom.

Palindrom-Test
Geben Sie bitte das zu pruefende Wort ein: Palindrom
Das Wort "Palindrom" ist kein Palindrom
```

Gliedern Sie das Programm im drei Funktionen verteilt auf zweiModule / drei Dateien:

- 1. Datei: Main-Funktion: Eingabe und Ausgabe
- 2. Datei:
  - Funktion isPalindrom(...): Prüft, ob ein String ein Palindrom ist. Wenn ja gibt die Funktion den Wert 1 zurück ansonsten 0.
  - Funktion lengthOfString(...): Liefert die Nutz-Länge eines Strings (d. h. die Anzahl der Bytes vom Beginn des Strings, bis zum ersten Byte mit Wert 0). Diese Funktion benötigen Sie vermutlich innerhalb der Funktion isPalindrom(...). Verwenden Sie nicht die Funktion `strlen()`.
- 3. Datei: Header-Datei für die 2. Datei: Enthält die Prototypen der Funktion isPalindrom(...)

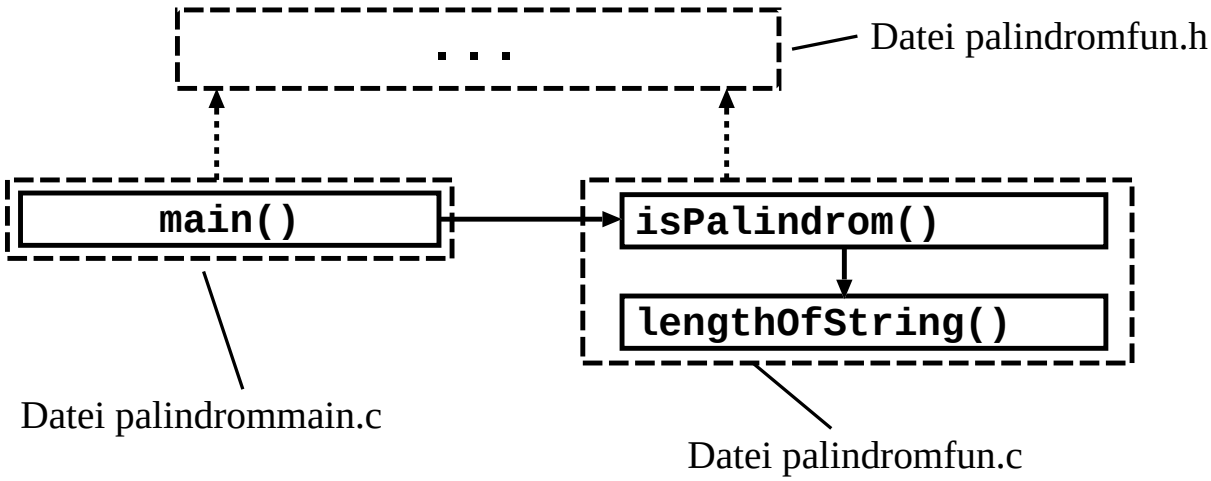
Verwenden Sie für die Funktion `isPalindrom()` folgendes Gerüst:

```
int isPalindrom(char line[]){
   int i;
   int length;
     :
     :

}
```

Definieren Sie keine weiteren Parameter oder lokale Variablen.

Ergebnis: Ein Ablaufdiagramm ist nicht für das gesamte Programm, sondern nur für die Funktion isPalindrom() gefordert. Stellen Sie auch die Aufrufhierarchie der Funktionen und der Dateien dar. Z.B. wie folgt:



A ⟶ **B** Funktion A ruft Funktion B auf

A ⇢ **B** Datei A verwendet Datei B

Freiwillige Zusatzaufgabe: Erweitern Sie das Programm so, dass es (a) Groß- und Kleinschreibung nicht unterscheidet ('Otto' soll als Palindrom erkannt werden) und (b) Leerzeichen ignoriert ('a man a plan a canal panama' soll als Palindrom erkannt werden). Verwenden Sie für (b) die Eingabe-Funktion `fgets( .... , .. , stdin)` statt `scanf()`.

Übung 6: Eine String-Funktion mit Ausgabeparameter

Entwerfen Sie ein Programm, das zu einem Eingabe-String folgende Werte berechnet:

1. Die Länge des Strings
2. Den Buchstaben mit dem größten ASCII-Wert
3. Den Buchstaben mit dem kleinsten ASCII-Wert

Diese Werte sollen von einer einzigen Funktion berechnet werden. Die Länge des Eingabe-Strings soll als Rückgabe-Wert zurückgegeben werden. Die beiden Buchstaben sollen als Ausgabe-Parameter zurückgegeben werden (call by reference). Verwenden Sie also keine globalen Variablen!

Teilen Sie Ihr Programm in zwei Module auf: Eine Datei mit der main-Funktion ("string_main.c") und ein Modul mit der (einen!) Funktion, die aus dem Eingabe-String die Ergebnisse berechnet ("string_hrw.h" und "string_hrw.c").

Das Programm soll so lange einen neuen Eingabe-String anfordern, bis der Nutzer "q" als Eingabe-String eingibt.

Bsp. für einen Programmlauf:

```
Bitte Test-String eingeben: abcdefg
length = 7, minChar = a, maxChar = g

Bitte Test-String eingeben: abc123*-
length = 8, minChar = *, maxChar = c

Bitte Test-String eingeben:
```

Es genügt, wenn das Programm nur Nicht-Umlaute richtig verarbeitet.

Ergebnis: Ein Ablaufdiagramm ist nicht für das gesamte Programm, sondern nur für die Funktion gefordert. Stellen Sie auch die Aufrufhierarchie der Funktionen dar.

Freiwillige Zusatzaufgabe: Erweitern Sie das Programm so, dass es auch Umlaute richtig verarbeitet. Hinweis: Die Operatoren $>$ und $<$ berücksichtigen (natürlich) das Vorzeichen der zu vergleichenden Werte. Der Datentyp `char` ist bei den meisten Compiler vorzeichenbehaftet. Um den Wertebereich von 0 bis 255 darstellen zu können, müssen Sie den Datentyp `unsigned char` verwenden.

Übung 7: Länge einer Strecke im Raum/Fläche eines Dreiecks

Eine Strecke im Raum wird durch zwei Punkte Bestimmt. Jeder Punkt wiederum wird durch drei Koordinaten bestimmt.

Schreiben Sie ein Programm, das zwei Punkte einliest, und die Entfernung dieser Punkte berechnet. Teilen Sie Ihr Programm in zwei Module auf: Eine Datei mit der main-Funktion und ein Modul (Header-Datei und Implementierungs-Datei), das die benötigte Datenstruktur und die verschiedenen Funktionen enthält.

Definieren Sie zunächst eine Datenstruktur `struct point`, die drei Koordinaten aufnehmen kann (alle Zahlenwerte als `float`). Diese Datenstruktur müssen Sie außerhalb der Main-Funktion definieren, damit alle Funktionen Ihres Programms diesen Datentyp kennen. Verwenden Sie aber keine globalen Variablen.

Schreiben Sie eine Funktion, die drei Koordinaten (Datentyp `float`) einliest und als Datenstruktur `struct point` zurück gibt:

struct point readPoint();

Schreiben Sie eine Funktion, die als Parameter zwei Punkte erhält, und den Abstand dieser Punkte zurück gibt:

float distance(**struct** point point1, **struct** point point2);

Die Formel für den Abstand d zweier Punkte (x_1, y_1, z_1) und (x_2, y_2, z_2) lautet:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Berechnen Sie zuerst die Werte $x_1 - x_2$, $y_1 - y_2$, $z_1 - z_2$ und anschließend daraus den Abstand d. Verwenden Sie dabei *nicht* die Funktion `pow()`. Um die Wurzel einer Zahl zu berechnen, können Sie die Funktion `double sqrt(double)` der Bibliothek `math.h` verwenden: `#include <math.h>` . (Unter Linux/Unix müssen Sie evtl. beim Übersetzen die Bibliothek angeben: `gcc -lm -g -std=c99` .... Dabei steht `-lm` für "library math".)

Bsp. für einen Programmlauf:

```
Entfernung berechnen
Bitte x-Koordinate eingeben: 2
Bitte y-Koordinate eingeben: 3
Bitte z-Koordinate eingeben: 4

Bitte x-Koordinate eingeben: 5
Bitte y-Koordinate eingeben: 4
Bitte z-Koordinate eingeben: 0

Die Entfernung von Punkt 1 nach Punkt 2 ist: 5.099020
```

Erweitern Sie das oben beschriebene Programm. Es soll drei Punkte einlesen und die Fläche des durch diese Punkte bestimmten Dreiecks berechnen. Schreiben Sie dazu eine Funktion, die die Fläche berechnet und dazu die Funktion für die Längenberechnung `distance()` nutzt:

float areaTriangle(**struct** point point1, **struct** point point2, **struct** point point3);

Die Formel für die Flächenberechnung:

Die Kanten des Dreiecks besitzen die Längen a, b, c. Der halbe Umfang s beträgt:

$$s = (a + b + c)/2$$

Die Fläche f beträgt:

$$f = \sqrt{s * (s - a) * (s - b) * (s - c)}$$

Sie sollten in Ihrem Quell-Code keine großen Formeln verwenden; berechnen Sie statt dessen Zwischenergebnisse, die dann zum Endergebnis verrechnet werden. Z. B.: Zunächst nur die Differenz der x-, y- und z-Koordinaten berechnen (3 Zeilen), dann die Länge (4. Zeile) anschließend können daraus ggf. weitere Größen berechnet werden.

Ergebnis: Für diese Aufgabe ist kein Ablaufdiagramm gefordert. Stellen Sie die Aufrufhierarchie der Funktionen dar.

Übung 8: Speichern einer Menge von Punkten

Schreiben Sie ein Programm, das beliebig viele Punkte einlesen und ein einer verkettete Liste speichern kann. Die Punkte sollen dann in Umgekehrter Reihenfolge der Eingabe auf dem Bildschirm ausgegeben werden.

Der Benutzer kann nach jeder Eingabe entscheiden, ob er noch einen Punkt eingeben möchte (Eingabe 'p'), oder ob die gespeicherten Punkte ausgegeben werden sollen (Eingabe 'q'. Während der Ausgabe soll der Stack Punkt für Punkt geleert werden, so dass er zuletzt wieder leer ist.

Die Liste soll ausschließlich durch die Stack-Funktionen `push()` und `pop()` verwaltet werden.

Die Punkte sollen in einer Schleife eingelesen und auf dem Stack abgelegt werden. Verwenden Sie für die Elemente auf dem Stack die Datenstruktur aus Übung 7 und eine zusätzliche Struktur als "Umverpackung", wie in der Vorlesung besprochen.

```
struct stackPoint{          /* Umverpackung fuer POINT */
      :
};
typedef struct stackPoint STACK_POINT;
typedef STACK_POINT *STACK_POINT_PTR;
```

Realisieren Sie folgende Funktionen:

```
void push(POINT pushPoint);
POINT pop();
int isEmpty(); /* gibt 'wahr' (1) zurueck, wenn der Stack leer ist */
void printStackElement(POINT aPoint); /* gibt einen Punkt aus */
```

Verteilen Sie Ihr Programm auf drei Dateien:

1. `pointStack.h` Enthält die Deklaration von `POINT`, `push()`, `pop()`, `isEmpty()` und `printStackElement(`
2. `pointStack.c` Importiert `pointStack.h`. Enthält die Implementierung von `push()`, `pop()`, `isEmpty()` und `printStackElement()`.
3. `stackmain.c` Importiert `pointStack.h`. Implementiert die Funktionen `main()` und `readPoint()`

Bsp. für einen Programmlauf:

```
'p' fuer Punkt eingeben, 'q' fuer Ausgabe: p
Bitte X-Wert eingeben: 1
Bitte Y-Wert eingeben: 1
Bitte Z-Wert eingeben: 1

'p' fuer Punkt eingeben, 'q' fuer Ausgabe: p
Bitte X-Wert eingeben: 2
Bitte Y-Wert eingeben: 2
Bitte Z-Wert eingeben: 2

'p' fuer Punkt eingeben, 'q' fuer Ausgabe: p
Bitte X-Wert eingeben: 3
```

```
Bitte Y-Wert eingeben: 3
Bitte Z-Wert eingeben: 3

'p' fuer Punkt eingeben, 'q' fuer Ausgabe: p
Bitte X-Wert eingeben: 4
Bitte Y-Wert eingeben: 4
Bitte Z-Wert eingeben: 3

'p' fuer Punkt eingeben, 'q' fuer Ausgabe: p
Bitte X-Wert eingeben: 2
Bitte Y-Wert eingeben: 3
Bitte Z-Wert eingeben: 4

'p' fuer Punkt eingeben, 'q' fuer Ausgabe: p
Bitte X-Wert eingeben: 5
Bitte Y-Wert eingeben: 5
Bitte Z-Wert eingeben: 5

'p' fuer Punkt eingeben, 'q' fuer Ausgabe: q

Ausgabe in main()
x=  5.00  y=  5.00  z=  5.00
x=  2.00  y=  3.00  z=  4.00
x=  4.00  y=  4.00  z=  3.00
x=  3.00  y=  3.00  z=  3.00
x=  2.00  y=  2.00  z=  2.00
x=  1.00  y=  1.00  z=  1.00
```

Ergebnis: Stellen Sie die Aufrufhierarchie der Funktionen dar.

Übung 9: Zeilen einer Datei umkehren

Schreiben Sie ein Programm, das zwei Dateinamen einliest und die erste Datei zeilenweise in die zweite Datei kopiert. Dabei soll die Reihenfolge der Buchstaben jeder Zeile umgekehrt werden. Verwenden Sie die Funktion fgets(), um zeilenweise zu lesen. Hinweis: `fgets()` liefert das/die Zeilenende-Zeichen mit zurück, d. h. der gelesene String endet mit der Zeilenende-Markierung vor einem 0-Byte. Sie können eine feste Obergrenze für die Zeilenlänge annehmen (z. B. 128 Buchstaben). Überlegen Sie sich, ob sie die Dateien im Text-Mode oder besser im Binär-Mode öffnen.

Schreiben Sie eine Funktion, die die Reihenfolge der Buchstaben innerhalb eines Strings umkehrt (Zeilenende passend berücksichtigen). Wenn Sie beim Öffnen der Datei den passenden Modus wählen, so besteht die Zeilenende-Markierung (EOF) aus dem Zeichen ASCII-Code 10. Im Programm können Sie EOF durch `'\n'` darstellen, z. B. `if (... == '\n'){ ...`

Verwenden Sie für die Funktion folgendes Gerüst und definieren Sie keine weiteren Parameter oder lokale Variablen.

```
void reverse(char line[]){
   int i;
   int length;
   char tmp;
     :
     :
   return;
}
```

Eine zweite Funktion öffnet die Dateien, kopiert die (invertierten) Zeilen und schließt die Dateien wieder. Das Hauptprogramm soll nur die Dateinamen einlesen und die Kopier-Funktion aufrufen (s. Bsp. aus den Arbeitsblättern).

Bsp.: Erste Datei (Eingabe):

```
ABC
die
Katze
sitzt im
Schnee
```

Zweite Datei (Ausgabe):

```
CBA
eid
eztaK
mi tztis
eenhcs
```

Wenn die erste Datei nicht existiert bzw. nicht geöffnet werden kann, soll das Programm eine entsprechende Fehlermeldung ausgeben. Wenn die zweite Datei bereits existiert, soll sie überschrieben werden. Wenn die zweite Datei nicht geöffnet werden kann, soll das Programm ebenfalls eine entsprechende Fehlermeldung ausgeben.

Ergebnis: Stellen Sie die Aufrufhierarchie der Funktionen dar. Ein Ablaufdiagramm ist nur für die Funktion, die das Öffnen und Schließen der Dateien übernimmt (zweite Funktion im text oben), gefordert.

Übung 10: Punkte in Datei speichern

Schreiben Sie ein Programm, das zunächst nach einem Dateinamen fragt und anschließend in einer Schleife drei Kommandos (n, l, q) akzeptiert. Verwenden Sie eine while-Schleife mit Abbruchbedingung und eine switch-Verzweigung um die verschiedenen Kommandos zu bearbeiten:

n Das Programm liest die Koordinaten eines Punktes ein (x,y,z) und schreibt den Punkt an das Ende der Datei.

l Das Programm gibt alle Punkte aus, die in der Datei gespeichert sind. Im Hauptprogramm soll hier nur ein Funktionsaufruf stehen (s. u.).

q Das Programm beendet sich.

Die Datei soll während das Programmlaufes nur einmal geöffnet werden. Verwenden Sie die Funktion fwrite() zum Schreiben auf die Datei und entsprechend fread() zum Lesen von der Datei. Bei jedem Lesen sollen bis zu vier Punkte auf einmal gelesen werden. Wenn die Datei beim Programmstart bereits Punkte enthält, sollen diese nicht gelöscht werden. Vielmehr sollen die neuen Punkte an die bereits vorhandenen angefügt werden.

Datenstruktur für einen Punkt:

```
struct point{
   float x;
   float y;
   float z;
}
```

Beispiel für einen Programmlauf:

```
Dateiname eingeben: points.dat
Eingabe n (neuer Punkt) l (Datei lesen) q (quit): l
Block aus Datei gelesen: 2 Datensaetze
Punkt (x, y, z):   1.00,     0.00,     1.00,
Punkt (x, y, z):   2.00,    42.00,    34.00,

Eingabe n (neuer Punkt) l (Datei lesen) q (quit): n
Bitte x-Koordinate eingeben: -7
Bitte y-Koordinate eingeben: 10
Bitte z-Koordinate eingeben: 333

Eingabe n (neuer Punkt) l (Datei lesen) q (quit): l
Block aus Datei gelesen: 3 Datensaetze
Punkt (x, y, z):   1.00,     0.00,     1.00,
Punkt (x, y, z):   2.00,    42.00,    34.00,
Punkt (x, y, z):  -7.00,    10.00,   333.00,

Eingabe n (neuer Punkt) l (Datei lesen) q (quit): q
```

Um das Hauptprogramm modular zu halten schreiben Sie bitte Funktionen für folgende Teilaufgaben: Einlesen eines Punktes, Auslesen und Anzeigen aller Punkte der Datei.

Testen Sie das Programm mit mindestens 5 Punkten in der Datei.

Ergebnis: Für diese Aufgabe ist kein Ablaufdiagramm gefordert. Stellen Sie die Aufrufhierarchie der Funktionen dar.

Übung 11: Werte von Binomialkoeffizienten berechnen (Pascalsches Dreieck)

Schreiben Sie *zwei* Programme, die jeweils einen Wert des Binomialkoeffizienten berechnen. Der Wert lässt sich u. a. durch folgende Formeln berechnen.

Nichtrekursiv:

$$\binom{n}{k} = \begin{cases} 0 & \text{für} \quad 0 \leq n < k \\ \frac{n!}{k!(n-k)!} & \text{für} \quad 0 \leq k \leq n \end{cases}$$

Alternativ lässt sich der Binomialkoeffizient aus folgender rekursiven Formel berechnen:

$$\binom{n}{k} = \begin{cases} 1 & \text{für} \quad k = 0 \\ 1 & \text{für} \quad n = k \ \text{(oder 0 für } \ n < k) \\ \binom{n-1}{k} + \binom{n-1}{k-1} & \text{für} \quad 0 < k < n \end{cases}$$

Zusätzlich zur mathematischen Definition (s. o.) sollen die Programme den Wert -1 zurückliefern, falls einer der Parameter einen negativen Wert besitzt. Schreiben Sie für beide Varianten ein Programm. Die Berechnungsfunktion arbeitet also im ersten Programm nichtrekursiv und im anderen Programm rekursiv. Testen Sie die Programme mit Werten aus dem Zahlenbereich: $0 \leq k \leq n \leq 30$ und überlegen Sie sich, worin das teilweise unterschiedliche Verhalten begründet ist.

Verwenden Sie jeweils folgendes Code-Gerüst als Basis:

```
int binomialCoefficient(int n, int k){
  :
  :
}

int main(void){
    int n;
    int k;
    int binKoeff;

    printf("\n Berechnung des Binomialkoeffizienten n ueber k \n" );
    printf("\n Bitte Wert fuer n eingeben: ");
    scanf("%d", &n);
    printf("\n Bitte Wert fuer k eingeben: ");
    scanf("%d", &k);
    binKoeff = binomialCoefficient(n, k);
    printf("\n Der Binomialkoeffizienten n ueber k von %d und %d", n, k);
    printf(" ist: %d \n", binKoeff);
    return 0;
}
```

Für die Berechnung der Fakultät $n!$ in der nichtrekursiven Variante schreiben Sie bitte eine eigene Funktion:

```
int faktorial(int n){
  :
  :
}
```

Ergebnis: Für diese Aufgabe ist kein Ablaufdiagramm gefordert.

PS: Auf meiner HomePage finden Sie ein Programm, das den Binomialkoeffizient effizient berechnet und keine Zwischenergebnisse verwendet, die größer sind als das Endergebnis. Bitte melden Sie sich, wenn Sie dazu Fragen haben.