Frankfurt University of Applied Sciences
Fachbereich 2: Informatik und Ingenieurwissenschaften

Einführung in die Programmierung mit der Programmiersprache C (**EifP**, WS2025/26)

Prof. Dr. Thomas Gabel
Roman Ahmad, Emre Özöner, Gina Romanazzi

Aufgabenblatt 7

Aufgabe 29: Vom Flussdiagramm zum Programmtext

Inhalte und Sprachkonzepte der vorliegenden Aufgabe

- Umwandeln eines Flussdiagramms in C-Code

Aufgabenstellung

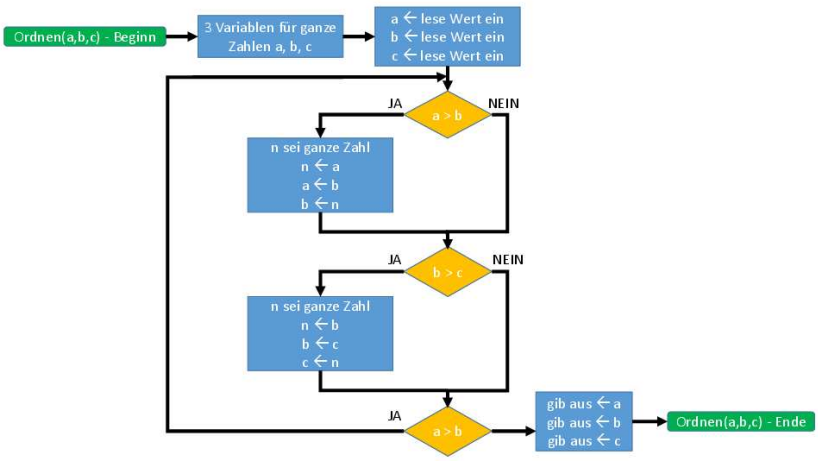
Schauen Sie sich das in der folgenden Abbildung dargestellte Flussdiagramm an und wandeln Sie dies in C-Code um. Führen Sie anschließend das fertige Programm mit den folgenden Testwerten aus und beantworten Sie umgangssprachlich die Frage: "Was tut dieses Programm?"

a) $a = 1 \quad b = 3 \quad c = 2$

b) $a = 0 \quad b = -1 \quad c = -100$

c) $a = 23 \quad b = 23 \quad c = 22$

d) $a = -1 \quad b = -2 \quad c = -3$



Aufgabe 30: Schleifenumformungen

Inhalte und Sprachkonzepte der vorliegenden Aufgabe

- Verwandtschaft der `do`-, `while`-Schleife

Aufgabenstellung

Gegeben sei die folgende `while`-Schleife:

```
int n, sum;
...
scanf("%d", &n);
sum = 0;
while ( n > -3 )  // (4)
{                 // (5)
  sum += n;       // (6)
  n--;            // (7)
}                 // (8)
```

Wandeln Sie dieses Code-Fragment (Zeilen 4 bis 8) in eine `do`-Schleife mit gleicher Bedeutung um.

Aufgabe 31: Schleifendurchläufe

Inhalte und Sprachkonzepte der vorliegenden Aufgabe

- Analyse der `for`-Anweisung
- ineinander geschachtelte `for`-Anweisungen

Aufgabenstellung

Bestimmen Sie für die folgenden Algorithmen eine Funktion $f(n)$, die in Abhängigkeit von $n \in \mathbb{N}$ angibt, wie häufig die <`AnweisungA`> bei der Ausführung des Programms abgearbeitet wird.
Hinweise:
$\sum_{k=1}^{n} k = \frac{1}{2}n(n+1)$, $\sum_{k=1}^{n} k^2 = \frac{1}{6}n(n+1)(2n+1)$, $\sum_{k=1}^{n} k^3 = \frac{1}{4}n^2(n+1)^2$

a)
```
int i;
for (i=1; i<=n+2; i++)
{
  <AnweisungA>;
}
```

b)
```
int i, j;
for (i=1; i<=n+1; i++)
{
  for (j=i; j<=2*n; j++)
  {
    <AnweisungA>;
  }
}
```

c)
```
int i, j, k;
for (i=1; i<=n; i++)
{
  for (j=i; j<=n; j++)
  {
    for (k=1; k<=j; k++)
    {
      <AnweisungA>;
    }
  }
}
```

Aufgabe 32: Ziffern als Wörter

Inhalte und Sprachkonzepte der vorliegenden Aufgabe

- Algorithmusentwurf und -umsetzung
- Auswahlanweisung

Aufgabenstellung

a) Schreiben Sie ein iteratives C-Programm, das eine über die Tastatur eingelesene Ganzzahl (`int`) im Wortlaut ausgibt. Es genügt, wenn das Programm beispielsweise die Zahl 547 in der Form "fünf vier sieben" ausgibt. Sorgen Sie dafür, dass das Programm alle Sonderfälle korrekt behandelt. *Hinweis:* Programmieren Sie eine Funktion, die eine Ziffer z als Parameter entgegennimmt und diese unter Verwendung der Auswahlanweisung als Wort auf dem Bildschirm ausgibt. Rufen Sie diese Funktion an allen Stellen auf, an denen eine einzelne Ziffer als Wort ausgegeben werden muss.

b) *Zusatzaufgabe:* Wandeln Sie Ihre Lösung zum Aufgabenteil a) in einen rekursiven Algorithmus um, also einen, der von einer rekursiven Funktion Gebrauch macht. Welche Lösung (a) oder b)) ist "eleganter"?

Aufgabe 33: Zufälliges Grundschulrechnen

Inhalte und Sprachkonzepte der vorliegenden Aufgabe[1]

- Zufallsgenerator
- Funktionen
- Schleifen und Auswahlanweisung

Aufgabenstellung

a) Schreiben Sie ein Programm, das eine zufällige Rechenaufgabe mit den vier Grundrechenarten ausführt. Die Rechenart (Addition, Subtraktion, Multiplikation oder Division) soll zufällig bestimmt werden. Verwenden Sie hierbei die Auswahlanweisung. Verwenden Sie durchgängig als Datentyp `int`. Vergessen Sie die passende Initialisierung des Zufallsgenerator nicht.
Die beiden Operanden sollen zufällig aus dem Intervall $[0, 9]$ ermittelt werden. Das Ergebnis der Rechenaufgabe soll sodann ermittelt und auf den Bildschirm in folgendem Format ausgegeben werden:

```
<operand1> <operator> <operand2> = <ergebnis>
```

Beispiel: `5 * 8 = 40`

b) Ändern Sie Ihr Programm nun so ab, dass die Berechnung des Ergebnisses in vier Funktionen erfolgt, die alle zwei ganzzahlige Parameter entgegennehmen und als Ergebnis ebenfalls eine ganze Zahl zurückliefern. Rufen Sie diese Funktionen an den passenden Stellen aus Ihrem Hauptprogramm heraus auf.

c) Ergänzen Sie Ihr Hauptprogramm um eine Zählschleife, die bewirkt, dass die in a) und b) realisierten zufälligen Berechnungen zehnmal ausgeführt werden.
Mögliche Ausgabe des Programmes:

```
6 + 3 = 9
0 - 3 = -3
6 * 5 = 30
1 + 6 = 7
3 + 1 = 4
4 / 9 = 0
4 / 6 = 0
6 * 2 = 12
6 - 7 = -1
0 / 6 = 0
```

[1] Beachten Sie bitte, dass die Formulierung dieser Aufgabenstellung sich stark an typischen Klausuraufgaben orientiert.

d) *Zusatzaufgabe:* Fangen Sie den Sonderfall der Division durch Null in passender Weise ab. Erweitern Sie Ihr Programm außerdem so, dass die Division mit Fließkommazahlen unterstützt wird.

Aufgabe 34: Tutorial: Mehrere Kompilationseinheiten

Hinweis 1: Bei der folgenden Aufgabe handelt es sich nicht um eine "normale" Aufgabenstellung, sondern um ein kurzes "Tutorial". D.h. für Sie geht es nicht darum, eine eigene Lösung zu entwickeln bzw. ein eigenes Programm zu schreiben, sondern darum, alle Teilschritte a) bis i)) des Tutorials exakt nachzuspielen und dabei zu erkennen und zu verinnerlichen, wie man größere Programme aus mehreren Teilen (Modulen) zusammensetzen kann. *Hinweis 2:* Auch wenn es sich um keine herkömmliche Aufgabe handelt, ist es sehr wichtig, dass Sie alle Schritte dieses Tutorials vollständig verstehen und durcharbeiten, da Ihr Vorankommen und Erfolg in den folgenden Wochen sowie in der Klausur darauf aufbauen werden.

Inhalte und Sprachkonzepte der vorliegenden Aufgabe

- Verteilung des Quellkodes auf mehrere Dateien
- Kompilieren und Linken mehrerer Kompilationseinheiten

Aufgabenstellung

Wenn Sie größere Programme schreiben, so werden Sie diese (ab einer gewissen Größe) auf mehrere Dateien verteilen. Die Liste aller Funktionsprototypen einer c-Datei wird dabei in eine zugehörige h-Datei (Header-Datei) ausgelagert. Es gilt: Zu jeder c-Datei (normalerweise mit Ausnahme derjenigen, in der sich das Hauptprogramm befindet) sollte eine zugehörige Header-Datei existieren. Wir üben dies nun Schritt für Schritt:

a) Schreiben Sie eine c-Datei, in der Sie *ausschließlich* die folgenden beiden mathematischen Funktionen implementieren. Geben Sie dieser Datei den Namen `my_math_functions.c`.

```
int absWertInt( int v )
{
  if ( v < 0 )
    return -v;
  else
    return v;
}

double absWertDouble( double v )
{
```

```
  if ( v < 0.0 )
    return -v;
  else
    return v;
}
```

b) Schreiben Sie nun eine zugehörige h-Datei `my_math_functions.h`, in der Sie die Funktionsdeklarationen aus der c-Datei wie folgt auflisten:

```
int absWertInt( int v );
double absWertDouble( double v );
```

c) Fügen Sie Ihrer c-Datei `my_math_functions.c` aus Teilaufgabe a) als erste Zeile die folgende Präprozessor-Anweisung hinzu, um die Funktionsdeklarationen zu inkludieren:

```
#include "my_math_functions.h"
...
```

d) Ihre Mathematikfunktionen stellen nun eine gekapselte Einheit – bestehend aus der c-Datei und der h-Datei – dar, die Sie für sich stehend übersetzen können. Beachten Sie, dass Sie nun kein ausführbares Programm (es ist ja keine main()-Funktion vorhanden ...) erzeugen, sondern ein sogenanntes *Modul* (eine Object-Datei):
Befehl: `gcc -c my_math_functions.c`
Vergessen Sie den Parameter "-c" nicht; durch diesen wird nur diese Programmeinheit (nur dieses Modul) übersetzt und die Datei `my_math_functions.o` erzeugt.
Überprüfen Sie mittels `ls -la`, dass die o-Datei erzeugt worden ist.

e) Schreiben Sie nun eine weitere c-Datei, die Ihr Hauptprogramm enthalten und die den Namen `my_program.c` tragen soll. Dieses Programm soll die von Ihnen in der Datei (im Modul) `my_math_functions.c` implementierten Funktionen benutzen, daher ist die zweite Zeile sehr wichtig, in der Sie die Deklarationen Ihrer mathematischen Funktionen inkludieren. Beachten Sie, dass bei selbstdefinierten Bibliotheken (wie `my_math_functions.h`) beim Inkludieren keine spitzen Klammern, sondern Anführungsstriche zu benutzen sind.

```
#include <stdio.h>
#include "my_math_functions.h"

int main(void)
{
```

```
  int a = -3;
  printf("Der Absolutwert von %d ist: %d\n", a, absWertInt(a) );

  double b = -2.7;
  printf("Der Absolutwert von %lf ist: %lf\n", b, absWertDouble(b) );

  return 0;
}
```

f) Übersetzen Sie nun diese Kompilationseinheit mit dem Befehl `gcc -c my_program.c` und erzeugen Sie so die entsprechende Object-Datei `my_program.o`.
Überprüfen Sie mittels `ls -la`, dass die o-Datei erzeugt worden ist.

g) Beachten Sie, dass Sie bis hierher nur zwei Module in Form von Object-Dateien erzeugt haben, die jeweils für sich *nicht* ausführbar sind. Denn: Nur im Zusammenspiel können Sie funktionieren, da ein Modul Funktionalität aus dem zweiten benötigt.
Erzeugen Sie nun ein ausführbares Programm (Binary), indem Sie die beiden Module miteinander wie folgt verlinken (verschmelzen, d.h. Benutzung der Linker-Funktionalität des gcc):
`gcc my_program.o my_math_functions.o -o my_program`

h) Der nach dem Parameter-Flag "-o" folgende Name des ausführbaren Programmes ist natürlich wie gewohnt frei wählbar (muss also nicht `my_program` lauten): Führen Sie Ihr erzeugtes lauffähiges Programm durch Eingabe von `./my_program` aus.

i) *Übungsstundenaufgabe:* Erweitern Sie Ihre Sammlung von Mathematikfunktionen in den Dateien `my_math_functions.c` und `my_math_functions.h` um folgende Funktion:
`void grundrechenarten(double a, double b);`
Diese Funktion soll die Ergebnisse der vier Grundrechenoperationen Addition, Subtraktion, Multiplikation und Division ermitteln und auf den Bildschirm ausgeben. Selbstverständlich dürfen Sie hierbei Teile Ihres für Aufgabe 18 (Aufgabenblatt 5) entwickelten Quellkodes wiederverwenden. Lassen Sie im Hauptprogramm vom Benutzer die zwei Operanden a und b eingeben und rufen Sie für diese anschließend Ihre `grundrechenarten`-Funktion auf!