C-Kurs E4: Vertiefung II

Zielsetzung:
- Bitoperatoren, Arbeiten mit Masken
- Zeigerarithmetik
- Funktionszeiger und Funktionszeigerlisten
- Rekursion

Aufgabenblock 1: Bitoperationen
Zu den Bitoperatoren zählen: **`&`**, **`|`**, **`^`**, **`~`**, **`>>`** und **`<<`**. Die Operatoren **`&`** und **`|`** sind nicht zu verwechseln mit den logischen Operatoren **`&&`** und **`||`**. Die Bitoperatoren verknüpfen die Zahlen bitweise.

A 39: Bitweise Ausgabe
Schreiben Sie ein Funktion, die eine Zahl in Binärform, d.h. Bit für Bit auf der Konsole ausgibt. So soll z.B. der Aufruf von **`printBin(4711)`** die folgende Ausgabe erzeugen:
`00000000000000000001001001100111`

Der **`printf()`**-Befehl ist nicht in der Lage, Zahlen in Binärform auszugeben. Ein Lösungsansatz besteht darin, mit einer Maske zu arbeiten. Bei der Maske wird nur ein einzelnes Bit gesetzt, das in einer Schleife immer um eine Stelle nach rechts verschoben wird. Untenstehend ist der erste Schleifendurchgang (bei 8Bit) gezeigt:

Bit gesetzt

Zahl	1	0	1	0	1	1	1	1
				&				
Maske	1	0	0	0	0	0	0	0
				=				
Resultat	1	0	0	0	0	0	0	0

Bit nicht gesetzt

Zahl	0	0	1	0	0	1	1	1
				&				
Maske	1	0	0	0	0	0	0	0
				=				
Resultat	0	0	0	0	0	0	0	0

Wenn das Resultat **`!=0`** ist, ist das Bit in der Zahl gesetzt, bei Resultat **`0`** ist das Bit nicht gesetzt. Diese Information kann verwendet werden, um entweder ein **`'0'`** oder ein **`'1'`** mit **`printf()`** auf der Konsole auszugeben. Um die Maske für den ersten Schleifendurchgang vorzubereiten, kann der untenstehende Ausdruck eingesetzt werden:
```
unsigned int maske = 1 << (sizeof(zahl)*8)-1);
```

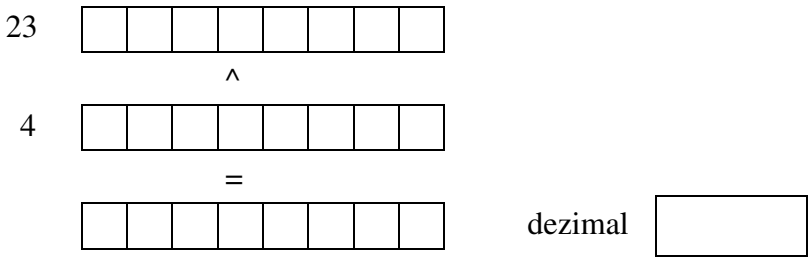
Die Maske muss als **`unsigned int`** deklariert werden, da es sonst vom Compiler abhängig ist, ob ein logischer oder arithmetischer Shift (vorzeichenbehaftet) durchgeführt wird.

A 40: Bitweises Invertieren

Der **~** Operator invertiert die Bits einer Zahl, bei zweimaligem Anwenden erhalten Sie wieder die ursprüngliche Zahl:
x = ~(~x)
Schreiben Sie eine Funktion, die dies zeigt.

A 41: Der EXOR-Operator

Bestimmen Sie manuell, wie gross x ist, und schreiben Sie anschliessend eine Funktion, um Ihr Resultat zu überprüfen: x = 23 ^ 4.



Aufgabenblock 2: Zeigerarithmetik

Wenn bei einem Zeiger eine Zahl hinzugezählt wird, richtet sich die neue Adresse nach der Grösse des Datentyps, auf den der Zeiger zeigt. Wenn p ein Zeiger und n eine ganze Zahl ist, kann man also sagen: p + n = p + n * (sizeof(*p)).

A 42: Summe und Differenz eines Zeigers mit einer ganzen Zahl

Welche Adresse erhalten Sie für den Zeiger, wenn diesem bei ... die Adresse **`100`** zugewiesen werde? Für ein **`char`** werde 1 Byte, für ein **`int`** 4 Byte eingesetzt.

Befehle	Adresse des Zeigers
`char* pc; ...` `pc = pc + 1;`	
`int * pi; ...` `pi = pi + 2;`	
`int* pi; ...` `pi = ++pi;`	

A 43: Stringumwandlung

Schreiben Sie ein Programm, welches die Kleinbuchstaben eines Strings in Grossbuchstaben umwandelt. Wenn Sie die Header-Datei **`ctype.h`** importieren, stehen u.a. folgende Funktionen zur Verfügung:
`int islower(int c)` `//` liefert !=0, wenn c Kleinbuchstabe
`int toupper(int c)` `//` liefert das ASCII-Zeichen für den Grossbuchstaben von c

Setzen Sie einen Zeiger ein, um über den String zu iterieren und die entsprechenden Zeichen auszuwechseln. Damit der Inhalt des Strings über den Zeiger änderbar ist, müssen Sie Ihn als Vektor und nicht als Zeichenkonstante (**`char *`**) definieren:

```
char text[] = "Ein kleiner Besipieltext";
```

Aufgabenblock 3: Funktionszeiger

Eine Besonderheit der Sprache C ist die Möglichkeit, eine Funktion einer Variablen zuzuweisen.

A 44: Deklaration und Einsatz von Funktionszeigern

Ein Programm stelle die untenstehenden Funktionen zur Verfügung:

```
double mul(double x, double y) { return x*y; }
void printText(char * text) { printf("%s\n", text); }
```

Schreiben Sie ein Programm, das die beiden Funktionen innerhalb von **`main()`** mittels Funktionszeigern aufruft.

A 45: Funktionszeigerlisten

Ein Programm soll die Möglichkeit geben, Funktionszeiger in einem Vektor anzulegen, hierzu soll die Funktion

```
void addFunction(void (*ptr) (void));
```

zur Verfügung gestellt werden. Der Parameter der Funktion zeigt, dass Funktionen mit der Signatur **`void <funktionsname>(void)`** übergeben werden dürfen. Eine zweite Funktion **`execute()`** soll alle Funktionen ausführen, die im Vektor gespeichert sind. Beispiele für einzutragende Funktionen sind:

```
void printRed() { printf("Ampel ist rot.\n"); }
void printOrange() { printf("Ampel ist orange.\n"); }
void printGreen() { printf("Ampel ist grün.\n"); }
```

Aufgabenblock 4: Rekursion

A 46: Fakultät

Schreiben Sie ein Programm, das die Fakultät einer Zahl n! mit einer Rekursion ermittelt. Die Regeln für die Berechnung einer Fakultät lauten:

n! = n * (n-1)!

1! = 1

Die zweite Regel ist die Abbruchbedingung der Rekursion.