KONTAKT STUDIUM IMP

Übungsaufgaben Informatik 1

Anwendungsaufgaben

In den folgenden Aufgaben werden Sie Zahlen in verschiedene Zahlensysteme umrechnen. Damit immer klar ist, in welchem Zahlensystem Sie sich befinden, schreiben Sie bitte an alle Zahlen einen entsprechenden Subscript; beispielsweise 110_2 für Binär-, 110_8 für Oktal-, 110_{10} für Dezimal-, 110_{16} für Hexadezimalzahlen und 110_{BCD} für Binary Coded Decimal an.

Aufgabe 1: *Umrechnungen zwischen Binär- und Dezimalsystem (6 Punkte)*

a) (1 Punkt) Wandeln Sie die folgende Binärzahl in eine Dezimalzahl um. Dokumentieren Sie den Verlauf der Umrechnung durch Darstellung der Summe, welche die Lösung ergibt.

$$101001,101_2$$

b) (2 Punkte) Wandeln Sie die folgende Dezimalzahl in eine Binärzahl um. Dokumentieren Sie den Verlauf der Umrechnung analog zum Beispiel aus der Vorlesung.

$$19,75_{10}$$

c) (1 Punkte) Betriebssysteme speichern normalerweise nicht ein oder zwei einzelne Bits ab, sondern immer einen Block aus genau 8 Bits - unabhängig davon ob alle 8 Bits zum Codieren einer Zahl benötigt werden. Die Zahl 3_{10} wird binär also beispielsweise als 00000011_2 abgespeichert. Man spricht von einem Byte, welches genau 8 Bits repräsentiert. Zwei Bytes entsprechen dann 16 Bits. Ein Betriebssystem kann in der Regel nur Byte-weise von der Festplatte lesen bzw. darauf schreiben.
Ihre Aufgabe: Berechnen Sie die Anzahl der verschiedenen Werte oder Zustände, die man mit einem Byte darstellen kann. Geben Sie den Rechenweg mit ab.

d) (2 Punkte) Um es sich im Kontaktstudium, aber auch im Unterricht etwas leichter zu machen, bietet es sich an, einige wichtige Binärzahlen auswendig zu lernen. Füllen Sie bitte die folgende Tabelle mit den fehlenden Dezimalzahlen aus und lernen Sie diese am besten auswendig - z.B. Ausschneiden und an den Kühlschrank hängen :).

Binärzahl	Dezimalzahl	Binärzahl	Dezimalzahl
$0000\ 000\mathbf{1}_2$		$0000\ 00\mathbf{10}_2$	
$0000\ 00\mathbf{11}_2$		$0000\ 0\mathbf{100}_2$	
$0000\ 0\mathbf{111}_2$		$0000\ \mathbf{1000}_2$	
$0000\ \mathbf{1111}_2$		$000\mathbf{1\ 0000}_2$	
$0001\ \mathbf{1111}_2$		$00\mathbf{10\ 0000}_2$	
$00\mathbf{11\ 1111}_2$		$0\mathbf{100\ 0000}_2$	
$0\mathbf{111\ 1111}_2$		$\mathbf{1000\ 0000}_2$	
$\mathbf{1111\ 1111}_2$		$\mathbf{1\ 0000\ 0000}_2$	

Hinweis: die Abstände in den Binärzahlen werden zur besseren Lesbarkeit verwendet. Normalerweise würden Sie alle 8 Bit ohne Abstand aneinander schreiben.

Aufgabe 2: *Umrechnungen zwischen Oktal-, BCD- und Dezimalsystem (3 Punkte)*

a) (1 Punkt) Wandeln Sie die folgende Binärzahl in eine Oktalzahl um:

$$111\ 000\ 011\ 100\ 110\ 001\ 101_2$$

b) (1 Punkt) Wandeln Sie die folgende Oktalzahl in eine Binärzahl um:

$$7\ 6\ 0\ 2\ 3\ 1_8$$

c) (1 Punkt) Wandeln Sie die folgende Dezimalzahl in einen BCD (=binary coded decimal) um.

$$129,42_{10}$$

Transferaufgaben

Neben dem Binärsystem ist das Hexadezimalsystem das wichtigste Zahlensystem in der Informatik. Die Dezimalzahlen $0-15_{10}$ werden dabei jeweils mit einer Ziffer bzw. einem Buchstaben dargestellt:

Dezimal	Hexadezimal	Dezimal	Hexadezimal	Dezimal	Hexadezimal
0_{10}	0_{16}	6_{10}	6_{16}	$\mathbf{12}_{10}$	$\mathbf{C}_{16}$
1_{10}	1_{16}	7_{10}	7_{16}	$\mathbf{13}_{10}$	$\mathbf{D}_{16}$
2_{10}	2_{16}	8_{10}	8_{16}	$\mathbf{14}_{10}$	$\mathbf{E}_{16}$
3_{10}	3_{16}	9_{10}	9_{16}	$\mathbf{15}_{10}$	$\mathbf{F}_{16}$
4_{10}	4_{16}	$\mathbf{10}_{10}$	$\mathbf{A}_{16}$	–	–
5_{10}	5_{16}	$\mathbf{11}_{10}$	$\mathbf{B}_{16}$	–	–

Aufgabe 3: *Umrechnungen zwischen Binär- und Hexadezimalsystem (3 Punkte)*

Hinweis: Auch diese Tabelle lohnt sich auswendig zu lernen. Wir werden in den kommenden Lerneinheiten ganz oft zwischen Dezimal-, Hexadezimal- und Binärsystem hin- und herrechnen.

a) (0.5 Punkte) Wie viele Bits werden benötigt um die Zahl F_{16} im Binärsystem darzustellen?

b) (0.5 Punkte) Zeigen Sie, dass die Hexadezimalzahl FF_{16} der Dezimalzahl 255_{10} entspricht.

c) (1 Punkt) Rechnen Sie die Binärzahl $11\ 0101\ 1110\ 1010_2$ in eine Hexadezimalzahl um. Sie können den gleichen Trick wie bei der Umrechnung zwischen Oktal- und Binärzahl anwenden.

d) (1 Punkt) Rechnen Sie die Hexadezimalzahl ABC_{16} in eine Binärzahl um. Auch hier können Sie den gleichen Trick anwenden.

Aufgabe 4: *Hexadezimalsystem und die Codierung von Farben (3 Punkte)*

Um Farben, zum Beispiel in JPG Bild-Dateien, zu speichern, gibt es verschiedene Modelle. Bei dem sehr verbreiteten RGB-Modell wird für jede der drei Grundfarben Rot, Grün und Blau ein Wert zwischen 0 und 255 angegeben. Die codierte Farbe ergibt sich dann additiv aus diesen Farb-Anteilen (= additiver Farbraum). Häufig werden die Farbanteile in Hex codiert, d.h. jeder Farbanteil befindet sich im Bereich $00_{16} - FF_{16}$ (vgl. Aufgabe 3 b). Statt dem Subscript sehen wir in vielen Programmen auch eine Raute '#' vor der hexadezimalen Darstellung:

Einige Beispiele:

rgb(0,0,0)	#000000	
rgb(255,0,0)	#FF0000	
rgb(0,255,0)	#00FF00	
rgb(255,255,0)	#FFFF00	
rgb(255,255,255)	#FFFFFF	Farbe weiß

Hinweis: RGB Farben können Sie unter Windows zum Beispiel schnell und einfach mit der Software Paint wählen (in der Farb-Palette unten rechts), unter MacOS bietet sich für einen einfachen Zugang Pages an (hier können auch mit dem Farbregler RGB-Farben gewählt werden). Langfristig empfehlen wir, sich für schulische Zwecke mit open-source Programm GIMP (`https://www.gimp.org`) zu beschäftigen. Das Programm kommt auch in der Lerneinheit 2 zum Einsatz.

a) (1 Punkt) Gegeben sei die RGB Farbe `#FF33A2`. Welche Farbe wird codiert? (Es reicht, den groben Farbton anzugeben.)

b) (1 Punkt) Ändern Sie die Farbe aus Teilaufgabe (a) ab; gesucht ist ein Lila-Ton. Geben Sie die Codierung (in Hexadezimalziffern) eines Lila-Tons an, ohne die Farbwerte für Grün und Blau zu verändern. Verwenden Sie ein Programm um Ihr Ergebnis zu überprüfen.

c) (1 Punkt) Wie viele verschiedene Farben können mit dem RGB Farbsystem dargestellt werden? Geben Sie Ihre Antwort im Dezimalsystem an; gerne als Produkt oder Potenz.

Programmieren mit Scratch

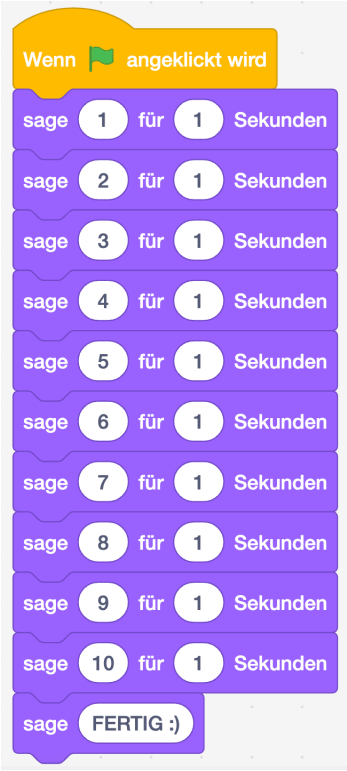
Jede*r Informatiker*in muss die Grundlagen der Programmierung beherschen. Deshalb wird jedes Übungsblatt eine oder mehrere Programmieraufgaben mit insgesamt 5 Punkten beinhalten. Auch die Programmieraufgaben sind prüfungsrelevant und eine Mindestpunktzahl in den Übungen ist nötig, um die Klausurzulassung zu erhalten.

In der Schule startet man fast immer mit einer ikonischen bzw. visuellen Programmiersprache. Auch wir werden mit Scratch starten und damit die grundlegenden Programmierkonzepte in den Übungen lernen. Mit Scratch kann man sehr einfach Spiele entwickeln (z.B. `https://de.scratch-wiki.info/wiki/Kultspiele_in_Scratch`) und erreicht damit das Interesse von vielen Schüler*innen. Wir konzentrieren uns aber eher auf die Entwicklung von Algorithmen. Warum trotzdem Scratch? Damit müssen wir uns im ersten Schritt nicht auf einen Syntax konzentrieren, sondern können mit den grundlegenden Konzepten starten. Nach dem Präsenztag im November steigen wir dann auf Java um und können damit komplexere Programme effizient schreiben.

Aufgabe 5: *Von der Katze, die das Zählen lernt … (5 Punkte)*

Ziel unseres ersten Programmes ist es, dass die Katze das Zählen von 1, 2, 3, … , bis n lernt. Wir werden dabei Schritt-für-Schritt vorgehen und uns als Programmierkonzepte die Veränderung von Variablen, Verarbeitung von Benutzereingaben und verschiedene Schleifenarten anschauen. Die Schritte in den Teilaufgaben sollen dabei auch das typische Vorgehen beim Programmieren wiederspiegeln: wir starten mit einem Programm für eine konkrete Aufgabe (in unserem Fall gibt die Katze die Zahlen 1 – 10 aus) und wandeln das Programm dann Stück für Stück in ein generisches Programm um, sodass die Katze von 1 – n zählt; wobei n vom Benutzer eingegeben wird. Lassen Sie sich also auch auf die einzelnen Schritte ein.

Als Hilfsmittel geben wir Ihnen folgendes Programm vor. Darin zählt die Katze von 1 bis 10 und gibt die einzelnen Ziffern in der Sprechblase aus. Schreiben Sie das o.g. Programm ab. Dies dient als Grundlage für die folgenden Teilaufgaben.



Bitte geben Sie von jeder Teilaufgabe (a) - (c) jeweils einen Screenshot ab (eingebunden in Ihre PDF Datei). Dann können wir Ihre Abgabe direkt kommentieren. Zusätzlich geben Sie bitte das Programm nach Teilaufgabe (c) als Scatch Datei (=*.sb3-Datei) mit ab. Klicken Sie dazu bitte in Scratch auf *Datei* → *Auf deinem Computer speichern* und geben die heruntergeladene Datei ebenfalls in Moodle ab. Für die Teilaufgaben (a) - (b) müssen Sie nicht extra eine Datei abgeben, aber die Screenshots! Trotzdem lohnt es sich einzelne Zwischenschritte abzuspeichern, damit Sie ggfs. auf einen früheren Stand Ihres Programms zurückspringen können.

a) (1 Punkt) Schauen wir uns im ersten Schritt das vorgegebene Programm an und probieren dieses aus. Wir sehen direkt: das Programm funktioniert für die konkreten Zahlen 1 – 10. Wollen wir das Programm auf 1–20 erweitern, so müssten wir aber für jede Zahl einen neuen Befehl schreiben. Das ist natürlich sehr unpraktisch. Gehen wir nun den ersten Schritt, um das Programm dynamisch für eine Eingabe anzupassen. Gehen Sie dazu wie folgt vor:
   - **1. Schritt: vereinfachen.** Geben Sie nur noch die Zahlen 1, 2 und 3 aus. Der Rest scheint gleich zu sein und kann erstmal weg. Das erleichtert uns die Umsetzung und das Testen.
   - **2. Schritt: Ausgaben durch eine Variable ersetzen.** Statt der konkreten Ausgaben *Sage '1' für '1' Sekunde*, *Sage '2' für '1' Sekunde*, ... möchte wir eine Variable verwenden:
     - Legen Sie eine neue Variable `zahl` an.
     - Initialisieren Sie die Variable mit dem ersten Wert, also `zahl := 1`.
     - Geben Sie den Wert der Variablen aus.
     - Erhöhen Sie den Wert der Variable um 1.
     - Geben Sie den Wert der Variablen aus.
     - usw.
   - Ihr Programm sollte nun immer noch gleich funktionieren und die Zahlen 1, 2, 3 ausgeben. Wir haben aber die konkreten Ausgaben mit einer Variable umgesetzt.
   - **Erstellen Sie einen Screenshot und fügen diesen in Ihre Abgabedatei ein.**

b) (2 Punkte) Schauen Sie sich Ihr Programm einmal genau an: die Ausgabe und die Änderung der Variablen `zahl` wird genau **dreimal gleich ausgeführt**, also dreimal die gleiche Befehlsfolge. Das möchten wir jetzt ändern. Wenn ein oder mehrere Befehle mehrfach ausgeführt werden, kommt eine Schleife zum Einsetz. Nutzen Sie in dieser Teilaufgabe die Schleife *Wiederhole 10 mal*, ändern die 10 durch eine 3 und binden Sie diese in Ihr Programm ein. Sie brauchen dann den Befehl zur Ausgabe und zur Erhöhung der Variablen nur noch genau einmal innerhalb der Schleife. Achten Sie darauf, dass die Initialisierung von `zahl := 1` vor der Schleife gemacht wird. Ihr Programm müsste nun weiterhin die Zahlen 1, 2, 3 ausgeben - aber wesentlich eleganter aussehen und keine doppelten Befehle mehr enthalten.
Erstellen Sie einen Screenshot und fügen diesen in Ihre Abgabedatei ein.

c) (2 Punkte) Nun möchten wir unser Programm so ändern, dass der Benutzer eingeben kann, wie weit die Katze zählen soll. Testen Sie dies nach der Programmierung erstmal mit einer kleinen Zahl und dann ggfs. auch mit der Informatiker-typischen 42 - dazu in den kommenden Übungen noch mehr :). Gehen Sie dann wie folgt vor:
   - Legen Sie eine neue Variable `n` an.
   - Bitten Sie den Benutzer in Ihrem Programm eine Zahl einzugeben (vgl. Beispielprogramm im Scratch-Vorkurs).
   - Speichern Sie die Antwort des Benutzers in die Variable `n`.
   - Schauen Sie sich nun Ihr Programm genau an. Welchen Teil müssen Sie verändern, damit die Schleife nicht 3-mal sondern `n`-mal durchläuft? Ändern Sie den Teil entsprechend und testen Sie Ihr Programm.

- Bei Schleifen ist es wichtig, dass wir die Randfälle testen. Schauen Sie also was passiert, wenn die Katze nur bis `1` zählen soll. Oder bis `0`. Oder was passiert, wenn Sie einen negativen Wert eingeben. Sie müssen hier nichts programmieren, sondern nur beobachten. Und es soll Ihnen in **Fleisch und Blut übergehen**, Schleifen immer ausführlich zu testen und besonders die Randfälle anzuschauen.
- **Erstellen Sie einen Screenshot und fügen diesen in Ihre Abgabedatei ein.**