INF11a

Zürcher Hochschule für Angewandte Wissenschaften

Arbeitsblatt: INF1

Name: _____ **Kurznamen:** _____

Bit-Operationen

Aufgabe 1: Bit-Operationen

In den Unterlagen finden Sie eine Funktion, die eine Zahl Binär auf die Konsole ausgibt. Entwickeln Sie eine Funktion *byte2binary* mit folgendem Prototyp (der *Prototyp* wird oft auch als *Signatur* bezeichnet):

```
void byte2binary (int num, char* result);
```

Die Funktion bekommt eine Zahl, deren *niederwertigste acht Bit* sie in Binärdarstellung umwandelt. Die Binärdarstellung wird in einen Zeichenpuffer (*result) geschrieben, der als zweites Argument übergeben wird (ähnlich wie bei *scanf* erfolgt die Rückgabe hier also nicht über den *return*-Wert der Funktion, sondern über einen zuvor bereitgestellten Speicherbereich).

Hinweise

Die Zeile

```
*result = (mask & num) ? '1' : '0';
```

ist auf den ersten Blick etwas unübersichtlich, da sie den ternären Operator "?:" verwendet. Die Zeile könnte man durch eine einfache aber längere Auswahlanweisung ersetzen:

```
if (mask & num) {
    *result = '1';
} else {
    *result = '0';
}
```

Die Anweisung *mask >>= 1* könnte man schreiben als *mask = mask >> 1*.

Aufgaben

- Machen Sie sich klar, wie *byte2binary* funktioniert. Testen Sie das Programm mit ein paar Eingabewerten.
- Schreiben Sie das Hauptprogramm so, dass zwei *int*-Zahlen m und n eingelesen werden. Diese sollen dann zuerst wieder im Binärformat ausgegeben werden und ausserdem die Ergebnisse der Operationen a & b, a | b, a >> 1, b << 2 und ~a.

Abgabe

Praktikum: INF11.1

Filename: bitoperations.c

Aufgabe 2: Funktionen zur Bitmanipulation

Implementieren Sie die Arduino Funktionen, mit denen einzelne Bits eines Datenwertes vom Typ *unsigned int* gesetzt, gelöscht, invertiert und abgefragt werden können. Parameter sind die Nummer des Bit und der Datenwert:
https://www.arduino.cc/reference/en/

```
// Computes the value of the specified bit.
// returns: bit 0 is 1, bit 1 is 2, bit 2 is 4, etc.
unsigned int bit(unsigned int n)

// Sets (writes a 1 to) a bit of a numeric variable.
// x: the numeric variable whose bit to set.
// n: which bit to set, starting at 0 for therightmost bit.
void bitSet(unsigned int* x, unsigned int n)

// Clears (writes a 0 to) a bit of a numeric variable.
// Sets (writes a 1 to) a bit of a numeric variable.
// x: the numeric variable whose bit to set.
// n: which bit to set, starting at 0 for therightmost bit.
void bitClear(unsigned int* x, unsigned int n)

// Read a bit of a number.
// x: the number from which to read
// n: which bit to read; 0 = rightmost bit
// returns: The value of the bit (0 or 1).
int bitRead(unsigned int x, unsigned int n)

// Write a bit of a number.
// x: the numeric variable to which to write.
// n: which bit of the number to write, 0 = rightmost bit
// b: the value to write to the bit (0 or 1).
void bitWrite(unsigned int* x, unsigned int n, unsigned int b)
```

Abgabe

Praktikum: INF11.2

Filename: arduinobit.c

Aufgabe 3: Parity Generator und Checker

Wenn auf einer Kommunikationsleitung Bytes mit 7-Bit-ASCII-Code übertragen werden, ist es üblich, die einzelnen Bytes mit einem sogenannten achten Paritäts- Bit zu versehen; damit soll die Übertragungssicherheit verbessert bzw. Fehler erkannt werden.

Das Parity-Bit "sagt", ob die Anzahl der Bits (in einem Byte), die auf Eins gesetzt sind, *gerade* oder *ungerade* ist.

Der Sender zählt somit die "Einer"-Bits im zu übertragenden Byte und ergänzt das Byte mit dem berechneten Paritäts- Bit so, dass die *Gesamtanzahl der Einser* gerade oder ungerade ist.

Der Empfänger berechnet auf die gleiche Weise die Parität des empfangen Bytes und vergleicht diesen mit dem übertragenen Paritäts-Bit.

Aufgabe

- Bei der seriellen Übertragung werden manchmal nur 7 Bits für Nutzdaten (Bits 0..6) und ein Parity-Bit (Bit 7) verwendet (vorderstes Bit = Parity)
- Berechnung der Parität: gerade Anzahl Einser in Bits 0..6 (even) -> Parität-Bit =0, ungerade Anzahl (odd) -> Parität-Bit =1.
- Schreiben Sie dafür eine Funktion `void addParity(unsigned char* num)`. Diese Parität wird in der obigen Funktion als vorderstes Bit des übergebenen Bytes gesetzt (**even Parity**).

Schreiben Sie eine weitere Funktion `bool checkParity(unsigned char num)` die die Parity überprüft

Summe der Einsen eines Informationsworts	Wert des Paritätbits	
	bei Even-Parity	bei Odd-Parity
gerade	0	1
ungerade	1	0

Beispiel:

Informationswort	Summe der Einsen	Paritätsbit / Codewort	
		bei Even-Parity	bei Odd-Parity
0011.1010	gerade	0 / 0011.1010 **0**	1 / 0011.1010 **1**
1010.0100	ungerade	1 / 1010.0100 **1**	0 / 1010.0100 **0**

Mit diesen beiden Funktionen könnten 1-Bit Fehler erkannt aber nicht korrigiert werden. Noch besser wäre ein CRC-Verfahren, das aber wesentlich komplexer ist.
https://de.wikipedia.org/wiki/Zyklische_Redundanzpr%C3%BCfung

Abgabe

Filename: parity.c

Praktikum: INF11.3