Übung 8

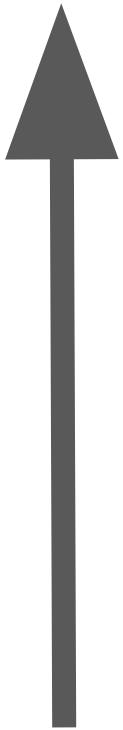
Binärdarstellung einer Dezimalzahl

Binär	0	1	1	0	1	1	0	0
Dezimal	128	64	32	16	8	4	2	1

0*128 + 1*64 + 1*32 + 0*16 + 1*8 + 1*4 + 0*2 + 0*1 = 108

Berechnung

Rechnung	Ergebnis	Rest
108:2	49	0
49:2	24	1
24:2	12	0
12:2	6	0
6:2	0	0
3:2	1	1
1:2	0	1
0:2	0	0



Ergebnis: 01100010

Bitweises UND

Bit A	Bit B	Bit A & Bit B
0	0	0
1	0	0
0	1	0
1	1	1

Bit wird auf 1 gesetzt, wenn beide Bits 1 entsprechen

Bitweises ODER

Bit A	Bit B	Bit A | Bit B
0	0	0
1	0	1
0	1	1
1	1	1

Bit wird auf 1 gesetzt, wenn ein Bit oder beide Bits auf 1 gesetzt sind.

Bitweise Verschiebung

- Linksverschiebung: <<
  - Verschiebung aller Bits einer Zahl um n Stellen nach links
  - rechte Leerstellen werden mit 0 aufgefüllt
- Rechtsverschiebung: >>
  - Verschiebung aller Bits einer Zahl um n Stellen nach rechts
  - linke Leerstellen werden mit 0 aufgefüllt

Bitweise Linksverschiebung

- int x = 4;

Binär	0	0	0	0	0	1	0	0
Dezimal	128	64	32	16	8	4	2	1

- x <<= 1;
  - Wie sieht x nach dieser Verschiebung aus?

Bitweise Linksverschiebung

- int x = 4;

Binär	0	0	0	0	0	1	0	0
Dezimal	128	64	32	16	8	4	2	1

- x <<= 1;
  - Wie sieht x nach dieser Verschiebung aus?

Binär	0	0	0	0	1	0	0	0
Dezimal	128	64	32	16	8	4	2	1

Bitweise Rechtsverschiebung

- int x = 4;

Binär	0	0	0	0	0	1	0	0
Dezimal	128	64	32	16	8	4	2	1

- x >>= 1;
  - Wie sieht x nach dieser Verschiebung aus?

Bitweise Rechtsverschiebung

- int x = 4;

Binär	0	0	0	0	0	1	0	0
Dezimal	128	64	32	16	8	4	2	1

- x >>= 1;
  - Wie sieht x nach dieser Verschiebung aus?

Binär	0	0	0	0	0	0	1	0
Dezimal	128	64	32	16	8	4	2	1

Bitweises Komplement

- NOT-Operator (~)
  - Invertierung jedes einzelnen Bits

Bit A	~Bit A
0	1
1	0

Exklusives Oder

- XOR-Operator (^)

Bit A	Bit B	Bit A ^ Bit B
0	0	0
1	0	1
0	1	1
1	1	0

Beispiel: Bitweises Oder

```
int x = 1;
```

Binär	0	0	0	1
Dezimal	8	4	2	1

```
int y = 2;
```

Binär	0	0	1	0
Dezimal	8	4	2	1

```
int or = x | y; // 3
```

Binär	0	0	1	1
Dezimal	8	4	2	1

Hinweis: Integer sind 32 Bit lang (aus Übersichtlichkeitsgründen nutze ich nur 4 Bits)

Beispiel: Bitweises Und

```
int x = 1;
```

Binär	0	0	0	1
Dezimal	8	4	2	1

```
int y = 2;
```

Binär	0	0	1	0
Dezimal	8	4	2	1

```
int or = x & y; // 0
```

Binär	0	0	0	0
Dezimal	8	4	2	1

Hinweis: Integer sind 32 Bit lang (aus Übersichtlichkeitsgründen nutze ich nur 4 Bits)

Beispiel: Bitweises Und

```
int x = 1;
```

Binär	0	0	0	1
Dezimal	8	4	2	1

```
int y = 3;
```

Binär	0	0	1	1
Dezimal	8	4	2	1

```
int or = x & y; // 1
```

Binär	0	0	0	1
Dezimal	8	4	2	1

Hinweis: Integer sind 32 Bit lang (aus Übersichtlichkeitsgründen nutze ich nur 4 Bits)

1a) - Bit zurückgeben

- Maske erstellen (um ein bestimmtes Bit zu isolieren)
  - int maske = 1 << n
    - an der n-ten Stelle (beginnend bei 0) steht eine 1
  - alle anderen Bits sind 0
  - Beispiel: int maske = 1 << 3 (Dezimal 8 | Binär 0…1000)
- Bitweise AND-Operation zwischen number und der Maske
  - number & maske
- Überprüfen, ob das Ergebnis der bitweisen AND-Operation ungleich null ist
  - Falls ja: das n-te Bit ist in number gesetzt

1b) - Bestimmtes Bit auf 1 setzen

- Überprüfen, ob die Stelle größer ist als die Zahl
  - n > sizeof(number) * 8
    - Fehler: return -1
- Maske erstellen
  - 1 << n
- Bitweises ODER einsetzen
  - number | maske

Bit A	Bit B	Bit A | Bit B
0	0	0
1	0	1
0	1	1
1	1	1

1c) - Bestimmtes Bit auf 0 setzen

- Überprüfen, ob die Stelle größer ist als die Zahl
  - n > sizeof(number) * 8
    - Fehler: return -1
- Maske erstellen
  - ~(1 << n)
- Bitweises UND einsetzen
  - number & maske

2a) - Bitweises Leuchten

- Schleifenstart von der Größe der Zahl -1
  - Zahl um Iteratorstellen nach rechts verschieben
  - Bitweises & mit 1 nutzen
  - Ausgabe der Zahl

2b) - Ausschalten/ Anschalten

- Setze eine Variable des Typen unsigned int auf 0
- Nutze den NOT Operator, um die Zahl auf den maximalen Wert des unsigned ints zu setzen
- Rufe dann die Funktion zur bitweisen Ausgabe auf (2a)