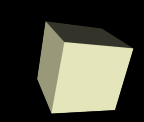
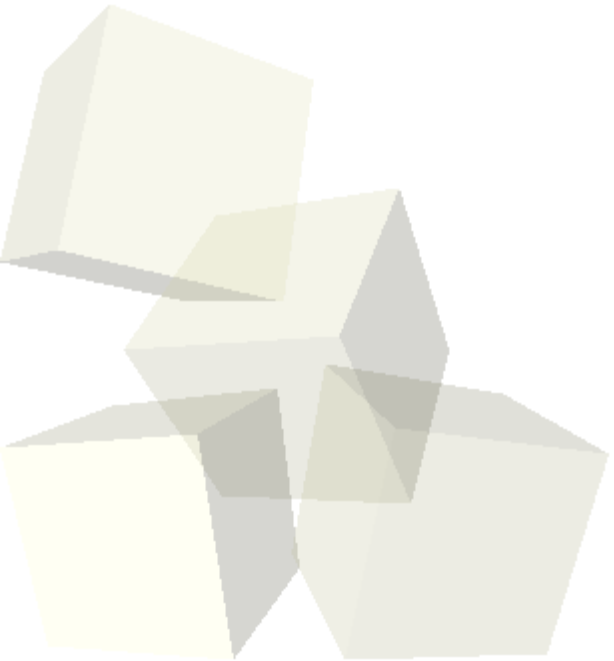


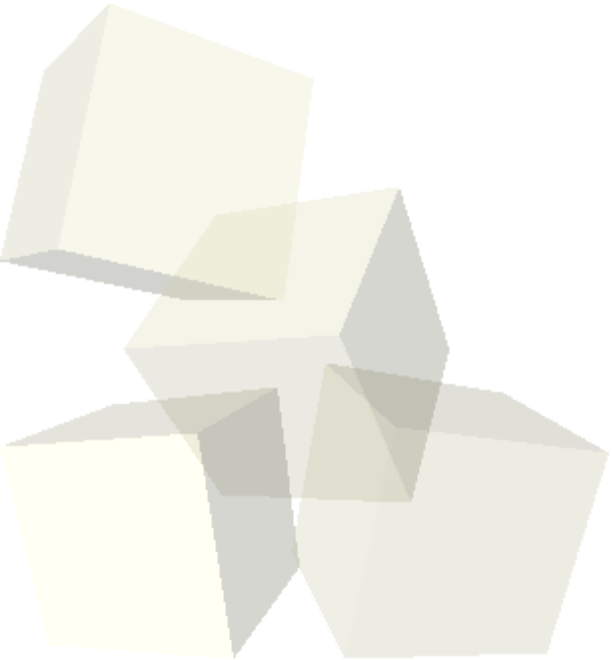
C-Schulung Tag 2

Herzlich willkommen



Agenda

- Repräsentation von Zahlen
- Zweierkomplement
- Bitweise Arithmetik



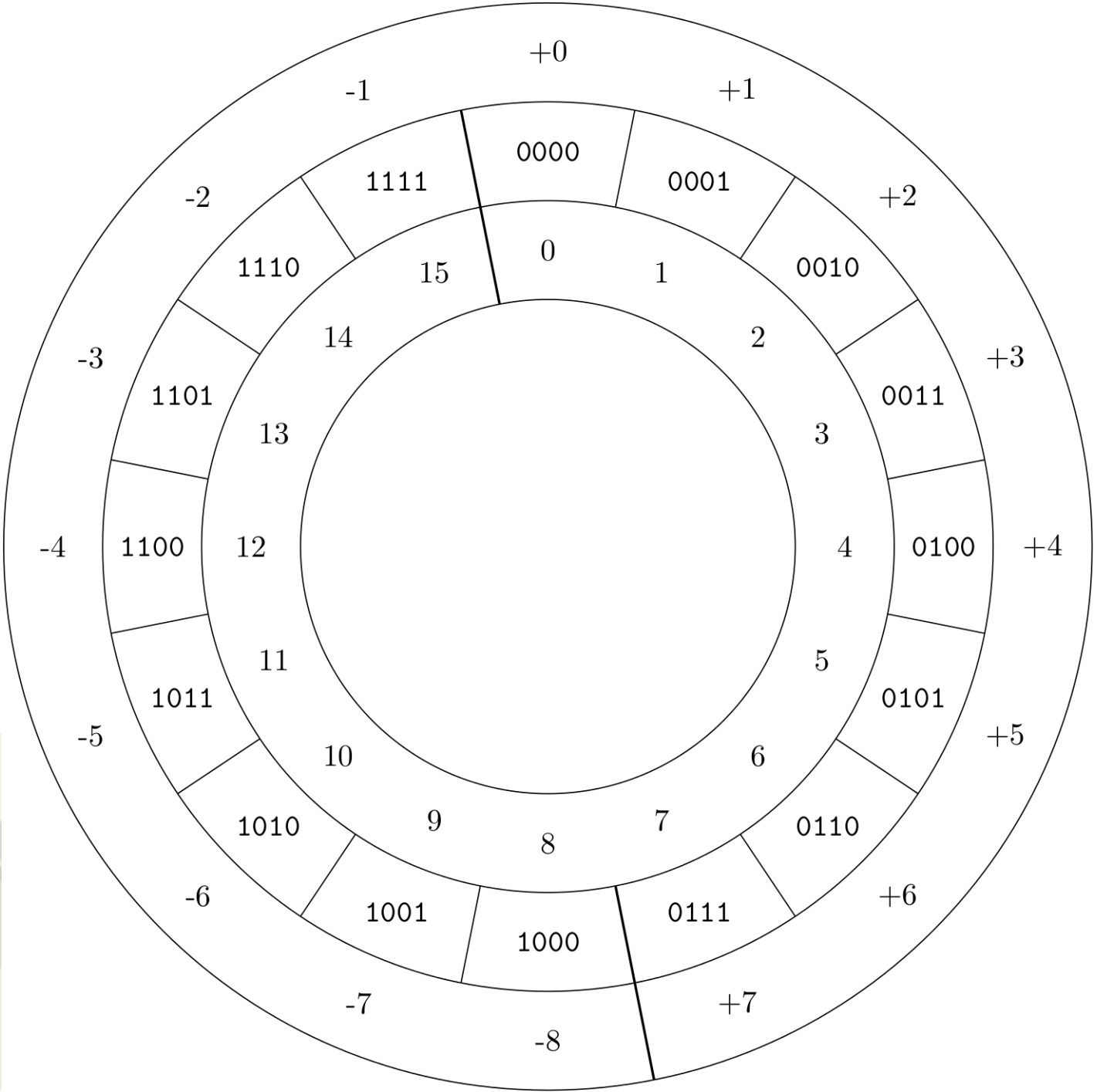
Repräsentation von Ganzzahlen

- Ein int ist eine (binär gespeicherte) Ganzzahl.
- Die Darstellung eines int-Werts auf dem Bildschirm erfordert eine Interpretation der Zahl, meist als Wandlung in eine Folge von Zeichen.
- Menschen bevorzugen dafür das Dezimalsystem.
  - Ein int ist deswegen aber trotzdem keine Dezimalzahl!
- Compiler und Bibliotheksfunktionen wandeln recht transparent Darstellungen und Zahlen für uns:
  - printf("%x\n%d\n", 25, 0777);

Zählen in verschiedenen Systemen

binär	oktal	dezimal	hexadezimal
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F

signed/unsigned am Beispiel 4 Bit



Zahlen negieren

- Um zu einer Zahl x die Zahl -x zu berechnen, kippt der Rechner alle Bits und addiert anschließend 1.

- Für einen Menschen ist es deutlich einfacher, lediglich die Bits links von der letzten 1 zu kippen.

```
 00000110     6                    10110000   -80

 11111001    ~6                    01001111  ~-80
 00000001    +1                    00000001    +1
==============                   ==============
 11111010    -6                    01010000    80
 00000110     6                    10110000   -80
==============                   ==============
100000000     0                   100000000     0
```

Bitweise Arithmetik

x	y	x & y	x | y	x ^ y	~x
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Die Bits der Operanden werden parallel („bitweise“) behandelt:

x	x_{n-1}	x_{n-2}	...	x_1	x_0
y	y_{n-1}	y_{n-2}	...	y_1	y_0
x o y	x_{n-1} o y_{n-1}	x_{n-2} o y_{n-2}	...	x_1 o y_1	x_0 o y_0

Bitmasken

x	1	0	1	0	0	1	1	0
y	0	0	0	0	1	1	1	1
x & y	0	0	0	0	0	1	1	0

x & 1 = x
x & 0 = 0

x	1	0	1	0	0	1	1	0
y	0	0	0	0	1	1	1	1
x | y	1	0	1	0	1	1	1	1

x | 0 = x
x | 1 = 1

x	1	0	1	0	0	1	1	0
y	0	0	0	0	1	1	1	1
x ^ y	1	0	1	0	1	0	0	1

x ^ 0 = x
x ^ 1 = ~x

Bit-Shifts

Links-Shift (entspricht Multiplikation mit Zweierpotenz)

x	1	0	1	0	0	1	1	0
x << 3	0	0	1	1	0	0	0	0

Vorzeichenloser Rechts-Shift (entspricht Division)

x	1	0	1	0	0	1	1	0
x >> 3	0	0	0	1	0	1	0	0

Vorzeichenbehafteter Rechts-Shift (entspricht Division)

x	1	0	1	0	0	1	1	0
x >> 3	1	1	1	1	0	1	0	0

Byte-Parität berechnen

```
/*
Die Paritaet eines Bytes gibt an, ob die Anzahl der gesetzten Bits
gerade ist (Paritaet = 0) oder ungerade (Paritaet = 1).
*/
int parity(unsigned char x)
{
    int bit0 = (x >> 0) & 1;   // 10100110 → 0
    int bit1 = (x >> 1) & 1;   // 10100110 → 1
    int bit2 = (x >> 2) & 1;   // 10100110 → 1
    int bit3 = (x >> 3) & 1;   // 10100110 → 0
    int bit4 = (x >> 4) & 1;   // 10100110 → 0
    int bit5 = (x >> 5) & 1;   // 10100110 → 1
    int bit6 = (x >> 6) & 1;   // 10100110 → 0
    int bit7 = (x >> 7) & 1;   // 10100110 → 1



    return bit0^bit1^bit2^bit3^bit4^bit5^bit6^bit7;   // 0^1^1^0^0^1^0^1 → 0
}
```

Teilmengen berechnen

```
void power_set(char a[ ], char b[ ], char c[ ], char d[ ])
{
    unsigned i;
    for (i = 0; i < 16; ++i)
    {
        if (i & 0x08) printf("%s ", a);
        if (i & 0x04) printf("%s ", b);
        if (i & 0x02) printf("%s ", c);
        if (i & 0x01) printf("%s ", d);
        putchar('\n');
    }
}

int main()
{
    power_set("Salat", "Tomate", "Gurke", "Paprika");
}
```

Datum komprimieren

```
short compressed_date(int year, int month, int day)
{
    // 0-127        0-15      0-31
    // YYYYYYY MMMM DDDDD
    return (year - 1970) << 9 | month << 5 | day;
}

void print_compressed_date(short date)
{
    int year = 1970 + (date >> 9);
    int month = (date >> 5) & 15;
    int day = date & 31;
    printf("%02d.%02d.%4d\n", day, month, year);
}

int main()
{
    short cd = compressed_date(2019, 2, 5);
    print_compressed_date(cd);
    return 0;
}
```

Unix-Datei-Berechtigungen

```
const char WHO[3][6] = {"owner", "group", "other"};
const char WHAT[3][8] = {"read", "write", "execute"};

void explain_permissions(short permissions)
{
    int mask = 1 << 8, who, what;
    printf("explaining %3o:\n", permissions);
    for (who = 0; who < 3; ++who)
    {
        for (what = 0; what < 3; ++what, mask = mask >> 1)
        {
            if (permissions & mask) printf("%s may %s\n", WHO[who], WHAT[what]);
        }
    }
}

int main()
{
    explain_permissions(0644); // pom.xml
    explain_permissions(0755); // sloc
    return 0;
}
```

Nützliche Bitmanipulationen

```
unsigned get_bit_n(unsigned x, unsigned n)
{
    return (x >> n) & 1;
}

unsigned set_bit_n(unsigned x, unsigned n)
{
    return x | (1 << n);
}

unsigned clear_bit_n(unsigned x, unsigned n)
{
    return x & ~(1 << n);
}

unsigned toggle_bit_n(unsigned x, unsigned n)
{
    return x ^ (1 << n);
}
```

```
unsigned lo_byte(unsigned x)
{
    return x & 0xff;
}

unsigned hi_byte(unsigned x)
{
    return x >> 24;
}
```