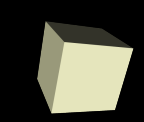
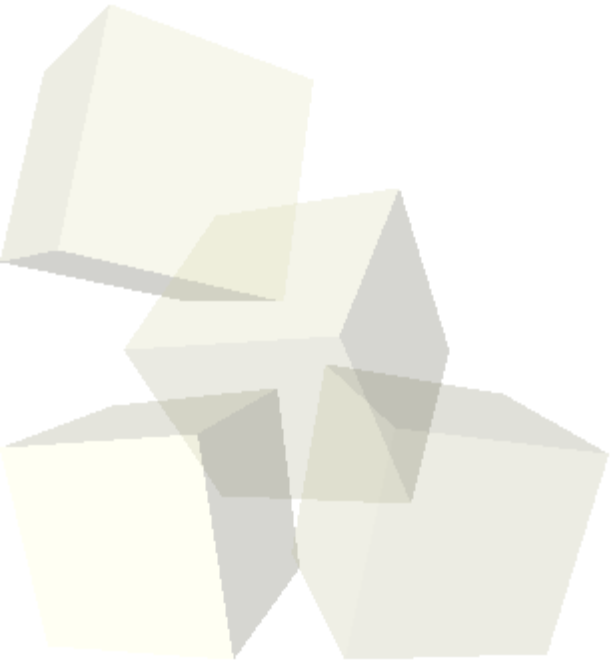
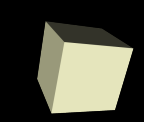


C-Schulung Tag 4

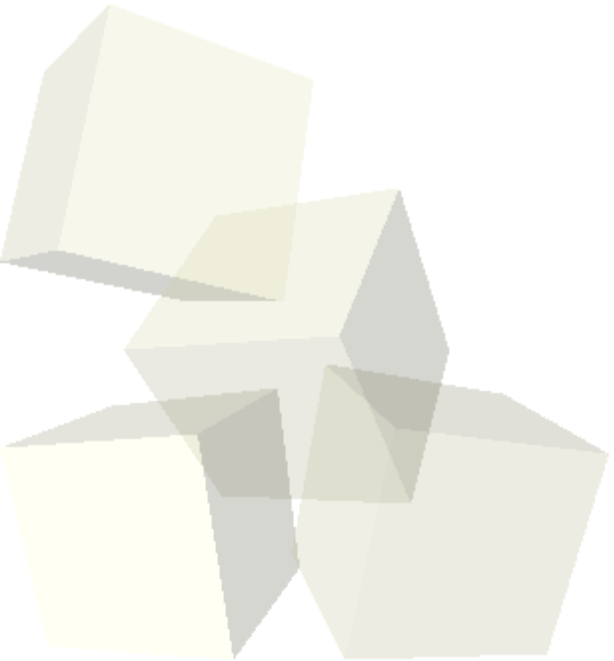
Herzlich willkommen





Agenda

- Zeiger
- Arrays und Zeiger
- Lebensdauer
- sizeof



Swap

```
#include <stdio.h>

void swap(int a, int b)
{
    int t;
    t = a;
    a = b;
    b = t;
}

int main(void)
{
    int x = 2;
    int y = 4;

    swap(x, y);

    printf("%d %d\n", x, y);
    return 0;
}
```

Swap

```
#include <stdio.h>

void swap(int a, int b)
{
    int t;
    t = a;
    a = b;
    b = t;
}

int main(void)
{
    int x = 2;
    int y = 4;

    swap(x, y);

    printf("%d %d\n", x, y);
    return 0;
}
```

Name	Typ	Adresse	Wert
x	int	0022ff1c	2
y	int	0022ff18	4

Swap

```
#include <stdio.h>

void swap(int a, int b)
{
    int t;
    t = a;
    a = b;
    b = t;
}

int main(void)
{
    int x = 2;
    int y = 4;

    swap(x, y);

    printf("%d %d\n", x, y);
    return 0;
}
```

Name	Typ	Adresse	Wert
t	int	0022feec	
a	int	0022ff00	2
b	int	0022ff04	4

Name	Typ	Adresse	Wert
x	int	0022ff1c	2
y	int	0022ff18	4

Swap

```
#include <stdio.h>

void swap(int a, int b)
{
    int t;
    t = a;
    a = b;
    b = t;
}

int main(void)
{
    int x = 2;
    int y = 4;

    swap(x, y);

    printf("%d %d\n", x, y);
    return 0;
}
```

Name	Typ	Adresse	Wert
t	int	0022feec	2
a	int	0022ff00	2
b	int	0022ff04	4

Name	Typ	Adresse	Wert
x	int	0022ff1c	2
y	int	0022ff18	4

Swap

```
#include <stdio.h>

void swap(int a, int b)
{
    int t;
    t = a;
    a = b;
    b = t;
}

int main(void)
{
    int x = 2;
    int y = 4;

    swap(x, y);

    printf("%d %d\n", x, y);
    return 0;
}
```

Name	Typ	Adresse	Wert
t	int	0022feec	2
a	int	0022ff00	4
b	int	0022ff04	4

Name	Typ	Adresse	Wert
x	int	0022ff1c	2
y	int	0022ff18	4

Swap

```
#include <stdio.h>

void swap(int a, int b)
{
    int t;
    t = a;
    a = b;
    b = t;
}

int main(void)
{
    int x = 2;
    int y = 4;

    swap(x, y);

    printf("%d %d\n", x, y);
    return 0;
}
```

Name	Typ	Adresse	Wert
t	int	0022feec	2
a	int	0022ff00	4
b	int	0022ff04	2

Name	Typ	Adresse	Wert
x	int	0022ff1c	2
y	int	0022ff18	4

Swap

```
#include <stdio.h>

void swap(int a, int b)
{
    int t;
    t = a;
    a = b;
    b = t;
}

int main(void)
{
    int x = 2;
    int y = 4;

    swap(x, y);

    printf("%d %d\n", x, y);
    return 0;
}
```

Name	Typ	Adresse	Wert
x	int	0022ff1c	2
y	int	0022ff18	4

Variablen

- Eine Variable hat sechs Eigenschaften:
  - Name
  - Typ
  - Gültigkeitsbereich
  - Adresse
  - Wert
  - Lebensdauer
- Wenn eine Variable als aktueller Parameter auftritt, dann wird ihr aktueller Wert übergeben.
- Um die Variable innerhalb der Funktion verändern zu können, müssen wir statt ihres (getypten) Wertes ihre (getypte) Adresse übergeben.
- Getypte Adressen heißen *Zeiger*.

Swap

```
#include <stdio.h>

void swap(int * a, int * b)
{
    int t;
    t = *a;
    *a = *b;
    *b = t;
}

int main(void)
{
    int x = 2;
    int y = 4;

    swap(&x, &y);

    printf("%d %d\n", x, y);
    return 0;
}
```

Swap

```
#include <stdio.h>

void swap(int * a, int * b)
{
    int t;
    t = *a;
    *a = *b;
    *b = t;
}

int main(void)
{
    int x = 2;
    int y = 4;

    swap(&x, &y);

    printf("%d %d\n", x, y);
    return 0;
}
```

Name	Typ	Adresse	Wert
x	int	0022ff1c	2
y	int	0022ff18	4

Swap

```
#include <stdio.h>

void swap(int * a, int * b)
{
    int t;
    t = *a;
    *a = *b;
    *b = t;
}

int main(void)
{
    int x = 2;
    int y = 4;

    swap(&x, &y);

    printf("%d %d\n", x, y);
    return 0;
}
```

Name	Typ	Adresse	Wert
t	int	0022feec	
a	int *	0022ff00	0022ff1c
b	int *	0022ff04	0022ff18

Name	Typ	Adresse	Wert
x	int	0022ff1c	2
y	int	0022ff18	4

Swap

```
#include <stdio.h>

void swap(int * a, int * b)
{
    int t;
    t = *a;
    *a = *b;
    *b = t;
}

int main(void)
{
    int x = 2;
    int y = 4;

    swap(&x, &y);

    printf("%d %d\n", x, y);
    return 0;
}
```

Name	Typ	Adresse	Wert
t	int	0022feec	2
a	int *	0022ff00	0022ff1c
b	int *	0022ff04	0022ff18

Name	Typ	Adresse	Wert
x	int	0022ff1c	2
y	int	0022ff18	4

Swap

```
#include <stdio.h>

void swap(int * a, int * b)
{
    int t;
    t = *a;
    *a = *b;
    *b = t;
}

int main(void)
{
    int x = 2;
    int y = 4;

    swap(&x, &y);

    printf("%d %d\n", x, y);
    return 0;
}
```

Name	Typ	Adresse	Wert
t	int	0022feec	2
a	int *	0022ff00	0022ff1c
b	int *	0022ff04	0022ff18

Name	Typ	Adresse	Wert
x	int	0022ff1c	4
y	int	0022ff18	4

Swap

```
#include <stdio.h>

void swap(int * a, int * b)
{
    int t;
    t = *a;
    *a = *b;
    *b = t;
}

int main(void)
{
    int x = 2;
    int y = 4;

    swap(&x, &y);

    printf("%d %d\n", x, y);
    return 0;
}
```

Name	Typ	Adresse	Wert
t	int	0022feec	2
a	int *	0022ff00	0022ff1c
b	int *	0022ff04	0022ff18

Name	Typ	Adresse	Wert
x	int	0022ff1c	4
y	int	0022ff18	2

Swap

```
#include <stdio.h>

void swap(int * a, int * b)
{
    int t;
    t = *a;
    *a = *b;
    *b = t;
}

int main(void)
{
    int x = 2;
    int y = 4;

    swap(&x, &y);

    printf("%d %d\n", x, y);
    return 0;
}
```

Name	Typ	Adresse	Wert
x	int	0022ff1c	4
y	int	0022ff18	2

Die zwei Gesichter des Sterns

- Der Stern hat zwei verschiedene Bedeutungen.
- In Deklarationen markiert er einen Zeigertyp:
  - `int * a;  // a ist ein Zeiger auf int`
- In Ausdrücken dereferenziert er einen Zeiger:
  - `*a = 42;  // speichere 42 in dem int, auf den a zeigt`

Leerzeichen in Deklarationen

Leerzeichen haben bei Zeigerdeklarationen keinerlei Auswirkungen:

```
int*a;     // a ist ein int *
int *b;    // b ist ein int *
int* c;    // c ist ein int *
int * d;   // d ist ein int *
```

Der Compiler sieht in allen vier Fällen “int”, “Stern”, “Bezeichner”.

Falls man mehrere Zeiger auf einmal deklariert, gilt jeder Stern immer nur für die unmittelbar folgende Variable:

```
int* a, b;   // a ist ein int *, b ist ein int
int *a, b;   // a ist ein int *, b ist ein int
```

Da die zweite Schreibweise bei Mehrfachdeklarationen weniger Verwirrungspotential bietet, wird sie von C-Programmierern meist bevorzugt. Dem Compiler sind solche Konventionen völlig egal.

Arrays und Zeiger

- Wir hatten bereits festgestellt, dass Arrays nicht per Wert übergeben (d.h. nicht kopiert) werden.
- Stattdessen wird implizit ein Zeiger auf die erste Zelle des Arrays übergeben.
- char[ ] a ist **innerhalb einer Parameterliste** nur syntaktischer Zucker für char * a
  - So konnten wir bereits drei Tage lang mit Zeigern programmieren, ohne den Mechanismus zu kennen.
- Wieso können wir das Array in der Funktion indizieren, obwohl wir nur einen Zeiger haben?
  - p[i] ist nur syntaktischer Zucker für *(p+i)
  - &p[i] ist nur syntaktischer Zucker für p+i
  - p+i liefert einen Zeiger, der i Zellen weiter zeigt als p.

Arrays und Zeiger

```
#include <stdio.h>

int main(void)
{
    char a[ ] = "Hallo Welt.\n";
    printf(a);
    char * p = a;

    a[5] = ',';
    p[10] = '!';

    printf(p);
    return 0;
}
```

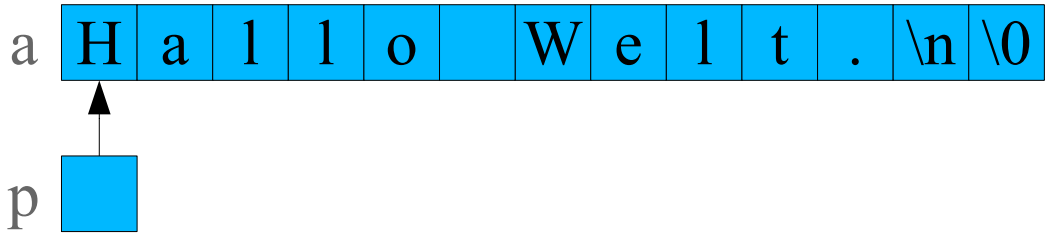
Arrays und Zeiger

```
#include <stdio.h>

int main(void)
{
    char a[ ] = "Hallo Welt.\n";
    printf(a);
    char * p = a;

    a[5] = ',';
    p[10] = '!';

    printf(p);
    return 0;
}
```



Name	Typ	Adresse	Wert
a	char[13]	0022ff0f	Hallo Welt.
p	char *	0022ff1c	0022ff0f

Lebensdauer

Die Lebensdauer lokaler Variablen endet normalerweise beim Verlassen ihrer Funktion. Deswegen darf man keine Zeiger auf solche Variablen zurückliefern:

```
char * message()
{
    char a[ ] = "Hallo Welt.\n";
    return a;   /* warning: function returns address of local variable */
}
```

Die Lebensdauer von Stringliteralen erstreckt sich dagegen über die gesamte Laufzeit. Daher kann man Zeiger auf Stringliterale problemlos herumreichen:

```
char * message()
{
    char * p = "Hallo Welt.\n";
    return p;
    // oder noch einfacher:
    return "Hallo Welt.\n";
}
```

Lebensdauer

Die Lebensdauer globaler Variablen erstreckt sich auch über die gesamte Laufzeit:

```
char a[ ] = "Hallo Welt.\n";

char * message()
{
    return a;
}
```

Für lokale Variablen kann dies mit dem Schlüsselwort static erreicht werden:

```
char * message()
{
    static char a[ ] = "Hallo Welt.\n";
    return a;
}
```

Achtung: sämtliche Aktivierungen von message teilen sich dann dasselbe a!

Funktionen mit Langzeitgedächtnis

```
unsigned random_number(unsigned bound)
{
    static unsigned seed = 1234567890;
    seed = seed * 1103515245 + 12345;
    return seed % bound;
} //https://en.wikipedia.org/wiki/Linear_congruential_generator

int main(void)
{
    int i;
    for (i = 1; i <= 6; ++i)
        printf("%d. Lottozahl: %2d\n", i, random_number(49) + 1);
    return 0;
}
```

1. Lottozahl: 17
2. Lottozahl: 46
3. Lottozahl: 27
4. Lottozahl: 21
5. Lottozahl: 28
6. Lottozahl: 30

sizeof

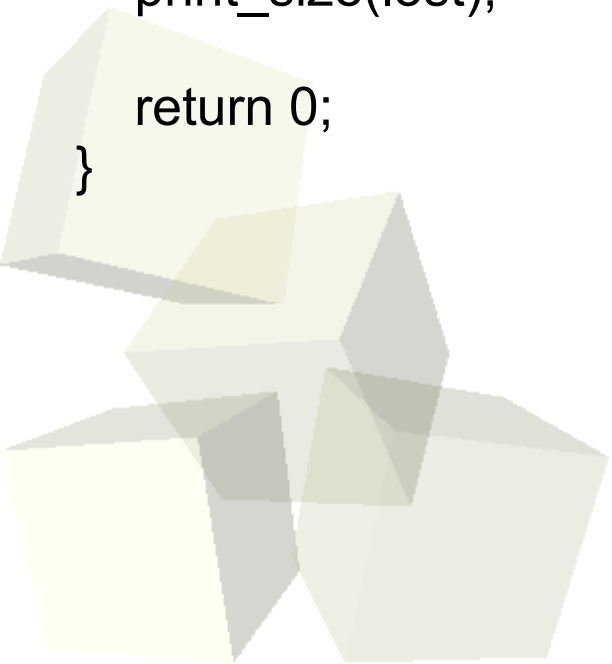
```
void print_size(int numbers[ ])
{
    printf("%lu\n", sizeof(numbers));
}

int main(void)
{
    int lost[ ] = {4, 8, 15, 16, 23, 42};

    printf("%lu\n", sizeof(lost));

    print_size(lost);

    return 0;
}
```



```
void print_size(int * numbers)   /* Umschreibregel in Parameterlisten */
{
    printf("%lu\n", sizeof(int *));
}

int main(void)
{
    int lost[6] = {4, 8, 15, 16, 23, 42};

    printf("%lu\n", sizeof(int[6]));

    print_size(lost);   // Implizite Typumwandlung int[6] → int *

    return 0;
}
```

sizeof

```
void print_size(int * numbers)
{
    printf("%lu\n", 4);
}

int main(void)
{
    int lost[6] = {4, 8, 15, 16, 23, 42};

    printf("%lu\n", 24);

    print_size(lost);

    return 0;
}
```

sizeof(ausdruck) wird bereits zur Übersetzungszeit ausgewertet. Außerhalb des Sichtbarkeitsbereichs eines Arrays ist seine Größe mittels sizeof nicht feststellbar!

Zusammenfassung

- Zeiger erlauben den indirekten Zugriff auf Variablen aus anderen Sichtbarkeitsbereichen.
- Zeiger können auf Zellen von Arrays zeigen:
  - int * p = &a[0];
- Überall, wo ein Zeiger erwartet wird, kann ein Array angegeben werden. Dabei wird implizit ein Zeiger auf die erste Zelle des Arrays geliefert:
  - int * p = a; // implizite Typumwandlung int[n] → int *
- Man kann keine Arrays an Funktionen übergeben, sondern nur Zeiger auf Zellen eines Arrays.
- Über einen Zeiger, der auf eine Zelle eines Array zeigt, ist dessen Größe nicht ermittelbar.
  - sizeof(array) vs. sizeof(pointer)
- Niemals Zeiger auf Stackvariablen zurückliefern!