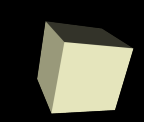
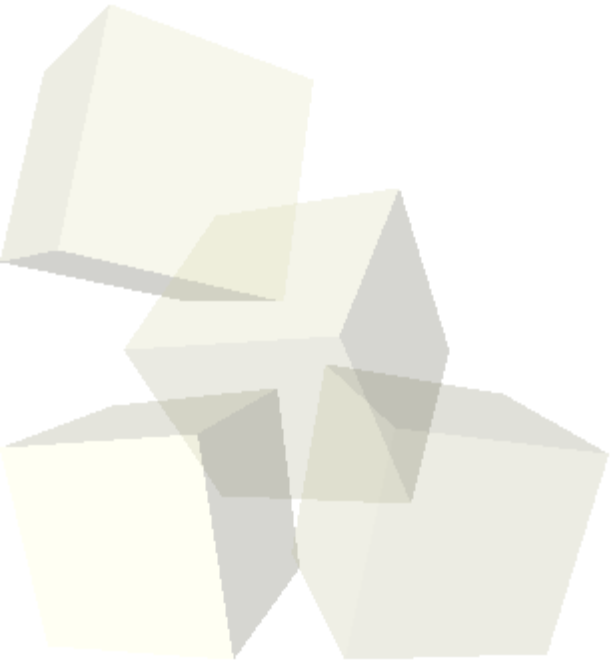


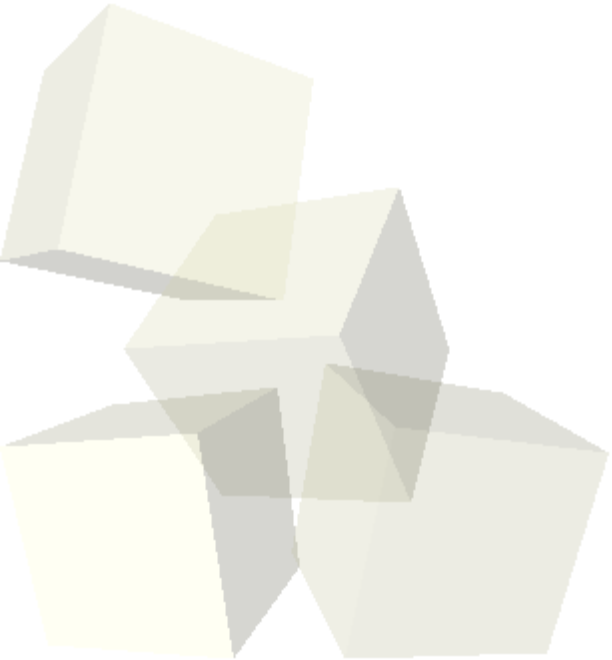
C-Schulung Tag 6

Herzlich willkommen



Agenda

- Formatierte Eingabe
- Zeilenweise Eingabe
- Ströme
- Dateien
- Dynamische Speicheranforderung



Formatierte Eingabe

```
#include <stdio.h>

int main(void)
{
    int a, b;
    printf("please enter two numbers: ");
    scanf("%d%d", &a, &b);
    printf("%d + %d = %d\n", a, b, a + b);
    return 0;
}
```

```
please enter two numbers: 1 2
1 + 2 = 3
```

```
please enter two numbers: 1
2
1 + 2 = 3
```

```
please enter two numbers:
1
2
1 + 2 = 3
```

```
please enter two numbers: quatsch
2147340288 + 4199280 = -2143427728
```

Formatierte Eingabe

```
#include <stdio.h>

int main(void)
{
    int a, b, n;
    printf("please enter two numbers: ");
    n = scanf("%d%d", &a, &b);
    if (n != 2)
        printf("%s argument invalid!\n",
            n ? "2nd" : "1st");
    else
        printf("%d + %d = %d\n",
            a, b, a + b);
    return 0;
}
```

```
please enter two numbers: 1 2
1 + 2 = 3
```

```
please enter two numbers: quatsch
1st argument invalid!
```

```
please enter two numbers: 1 zwei
2nd argument invalid!
```

```
please enter two numbers: 1 2 drei
1 + 2 = 3
```

Formatierte Eingabe

```
#include <stdio.h>

int main(void)
{
    int a, sum = 0;
    printf("please enter numbers: ");
    while (scanf("%d", &a) == 1)
        sum += a;
    printf("sum: %d\n", sum);
    return 0;
}
```

```
please enter numbers: 1 2 3
4
 5
  6
7 8 9 joghurt
sum: 45
```

Zeilenweise Eingabe

```c
#include <stdio.h>

int main(void)
{
    char name[20];
    printf("name: ");
    scanf("%s", name);
    printf("hello %s!\n", name);
    return 0;
}
```

```
name: Horst
hello Horst!

name: Max
hello Max!

name: Max Mustermann
hello Max!

name: Max_Mustermann
hello Max_Mustermann!

name: .....|.....|.....|
hello .....|.....|.....|!
```

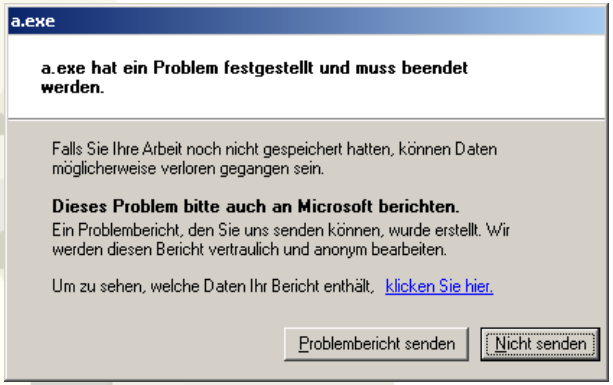
Zeilenweise Eingabe

```
#include <stdio.h>

int main(void)
{
    int guard = 0;
    char name[20];
    printf("name: ");
    scanf("%s", name);
    printf("hello %s!\n", name);
    printf("guard: %d\n", guard);
    return 0;
}
```

```
name: .........|.........|.........|
hello .........|.........|.........|!
guard: 0

name: .........|.........|.........|.........|
hello .........|.........|.........|.........|!
guard: 0

name: .........|.........|.........|.........|.........|
hello .........|.........|.........|.........|.........|!
guard: 774778414

name: <60>
hello <60>!
guard: 774778414
name: nanu
hello nanu!
guard: 0
```

Zeilenweise Eingabe

```
#include <stdio.h>

int main(void)
{
    int guard = 0;
    char name[20];
    printf("name: ");
    scanf("%s", name);
    printf("hello %s!\n", name);
    printf("guard: %d\n", guard);
    return 0;
}
```




```
name: .....|.....|.....|
hello .....|.....|.....|!
guard: 0

name: .....|.....|.....|.....|
hello .....|.....|.....|.....|!
guard: 0

name: .....|.....|.....|.....|.....|
hello .....|.....|.....|.....|.....|!
guard: 774778414

name: <60>
hello <60>!
guard: 774778414
name: nanu
hello nanu!
guard: 0

name: <70>
hello <70>!
```

Zeilenweise Eingabe

```c
#include <stdio.h>

int main(void)
{
    char name[20];
    printf("name: ");
    scanf("%19s", name);
    printf("hello %s!\n", name);
    return 0;
}
```

```
name: .....|.....|.....|.....|.....|
hello .....|.....!
```

```
name: Max Mustermann
hello Max!
```

Zeilenweise Eingabe

```
#include <stdio.h>

int main(void)
{
    char name[20];
    printf("name: ");
    fgets(name, 20, stdin);
    printf("hello %s!\n", name);
    return 0;
}
```

```
name: .........|.........|.........|.........|.........|
hello .........|.........!
```

```
name: Max Mustermann
hello Max Mustermann
!
```

Pufferüberläufe

- Im Gegensatz zu den meisten Ausgaben können Eingaben fehlerhaft sein.
- Bei zu langen Texteingaben kann es zu Pufferüberläufen kommen.
  - Diese gehören zu den häufigsten Sicherheitslücken und machen C zu einer relativ unsicheren Sprache.
- Zum Einlesen ganzer Zeilen ist fgets(s, n, stdin) die sicherste Alternative, unsicher sind dagegen:
  - scanf("%s", s)
  - gets(s)

Ströme

- stdin ist ein Strom, der Benutzereingaben für das Programm zugänglich macht.
- Viele Funktionen benutzen stdin implizit:
  - getchar() ⇒ getc(stdin)
  - scanf(format, ...) ⇒ fscanf(stdin, format, ...)
  - ...
- Für Ausgaben gibt es den Strom stdout:
  - putchar(c) ⇒ putc(c, stdout)
  - printf(format, ...) ⇒ fprintf(stdout, format, ...)
  - ...
- Für Fehlermeldungen gibt es stderr:
  - Ausgaben über stderr sind ungepuffert.
  - stderr wird nicht gepipet.

Dateien

- Neben den Standardströmen kann man auch neue Ströme schaffen, die auf Dateien arbeiten.
- Der aktuelle Zustand eines Stroms wird in einem Objekt des Datentyps FILE festgehalten.
- fopen gibt einen Zeiger auf ein FILE zurück.
- Andere Operationen arbeiten auf diesem FILE.
- Manche Operationen gibt es in 2 Ausführungen:
  - fgetc(FILE *) ist eine Funktion
  - getc(FILE *) ist eine Funktion oder ein Makro

In Textdatei schreiben

```
#include <stdio.h>

int main(void)
{
    FILE * datei;
    int i;

    if (datei = fopen("square.txt", "w"))
    {
        fprintf(datei, "Quadratzahlen\n\n");

        for (i = 0; i < 100; ++i)
            fprintf(datei, "%2d -> %4d\n", i, i * i);

        fclose(datei);
    }
    else
        printf("unable to open file\n");

    return 0;
}
```

Aus Textdatei lesen

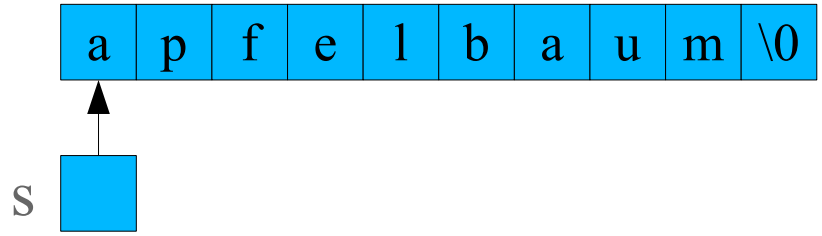
```
#include <stdio.h>

int main(void)
{
    FILE * datei;
    int n = 0;
    char buffer[1000];

    if (datei = fopen("precount.c", "r"))
    {
        while (fgets(buffer, sizeof(buffer), datei))
            if (buffer[0] == '#') ++n;
        fclose(datei);

        printf("preprocessor directives: %d\n", n);
    }
    else
        printf("unable to open file\n");

    return 0;
}
```

Dynamische Arrays

- Lokale Arrays haben zwei Einschränkungen:
  - Ihre Größe muss zur Übersetzungszeit feststehen.
  - Beim Rücksprung aus einer Funktion werden sie zerstört. (Ausnahme: statische Arrays)
- Dynamisch angeforderte Arrays unterliegen nicht diesen Einschränkungen:
  - Ihre Größe kann zur Laufzeit berechnet werden.
  - Sie überleben den Rücksprung aus der Funktion, in welcher sie erzeugt wurden.
- Beide Aspekte werden im Folgenden beleuchtet.

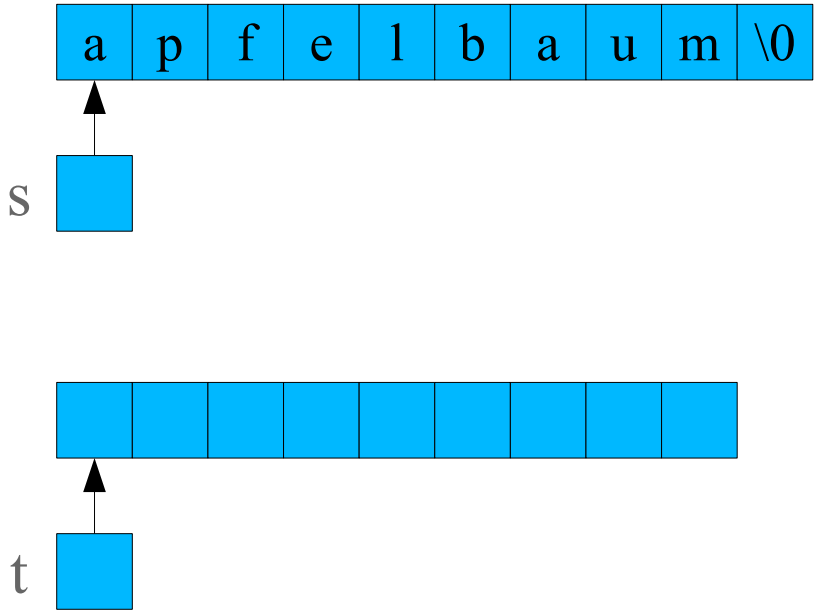
Dynamische Arrays in Funktionen

```
#include <stdlib.h>
#include <string.h>

void reverse(char * s)
{
    int i, k;

    int len = strlen(s);
    char * t = malloc(len);

    memcpy(t, s, len);

    for (i = 0, k = len - 1; i < len; ++i, --k)
    {
        s[i] = t[k];
    }

    free(t);
}
```



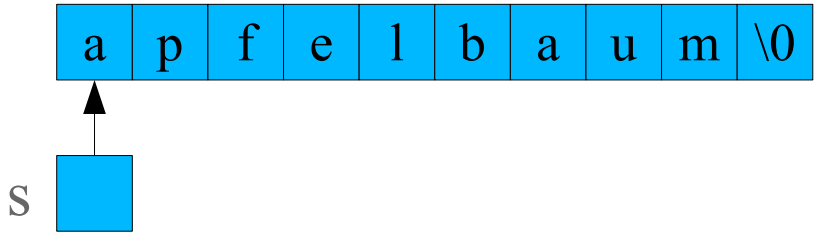
Dynamische Arrays in Funktionen

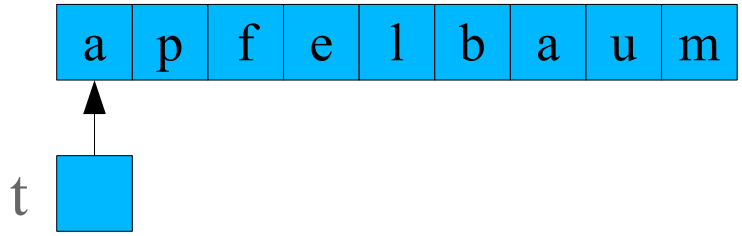
```
#include <stdlib.h>
#include <string.h>

void reverse(char * s)
{
    int i, k;

    int len = strlen(s);
    char * t = malloc(len);

    memcpy(t, s, len);

    for (i = 0, k = len - 1; i < len; ++i, --k)
    {
        s[i] = t[k];
    }

    free(t);
}
```



Dynamische Arrays in Funktionen

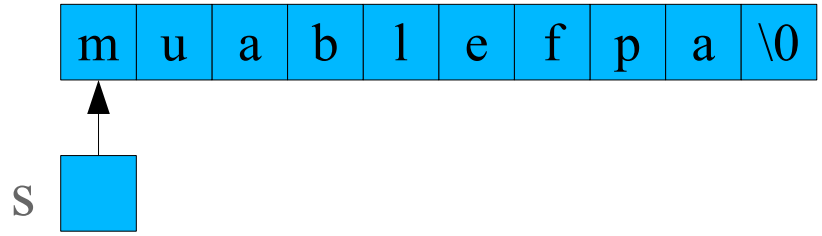
```
#include <stdlib.h>
#include <string.h>

void reverse(char * s)
{
    int i, k;

    int len = strlen(s);
    char * t = malloc(len);

    memcpy(t, s, len);

    for (i = 0, k = len - 1; i < len; ++i, --k)
    {
        s[i] = t[k];
    }

    free(t);
}
```

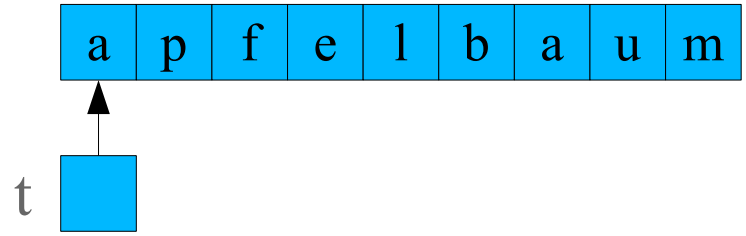




Dynamische Arrays in Funktionen

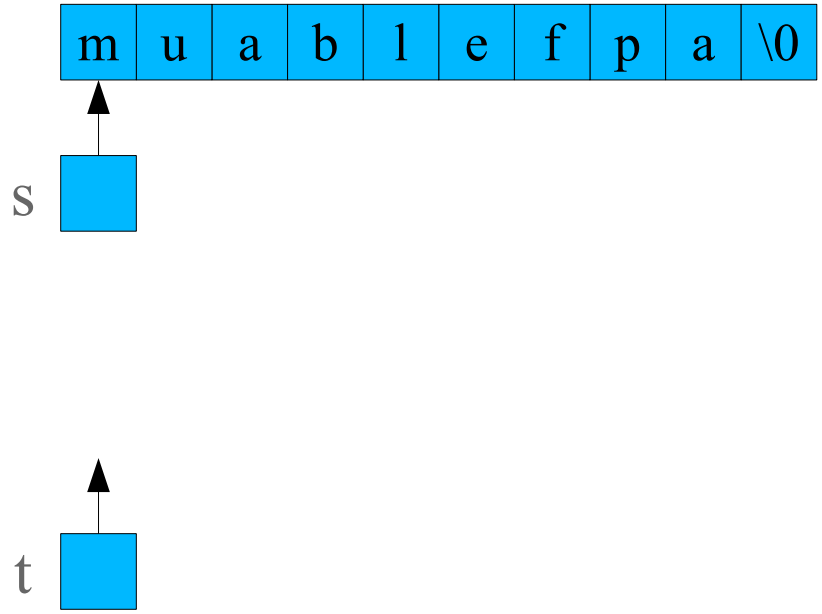
```
#include <stdlib.h>
#include <string.h>

void reverse(char * s)
{
    int i, k;

    int len = strlen(s);
    char * t = malloc(len);

    memcpy(t, s, len);

    for (i = 0, k = len - 1; i < len; ++i, --k)
    {
        s[i] = t[k];
    }

    free(t);
}
```





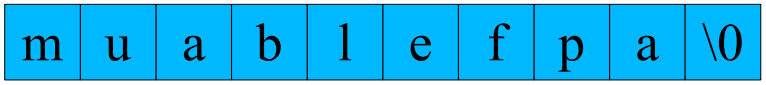
Dynamische Arrays in Funktionen

```
#include <stdlib.h>
#include <string.h>

void reverse(char * s)
{
    int i, k;

    int len = strlen(s);
    char * t = malloc(len);

    memcpy(t, s, len);

    for (i = 0, k = len - 1; i < len; ++i, --k)
    {
        s[i] = t[k];
    }

    free(t);
}
```



Dynamische Arrays in Funktionen

```
#include <stdlib.h>
#include <string.h>

void reverse(char * s)
{
    int i, k;

    int len = strlen(s);
    char * t = malloc(len);

    memcpy(t, s, len);

    for (i = 0, k = len - 1; i < len; ++i, --k)
    {
        s[i] = t[k];
    }

    free(t);
}
```



Dynamische Arrays zurückgeben

Da die Lebensdauer dynamisch angeforderter Arrays von Sichtbarkeitsbereichen entkoppelt ist, können diese Arrays aus Funktionen zurückgegeben werden:

```
int * berechne_quadratzahlen(int n)
{
    int * a = malloc(n * sizeof(int));
    int i;
    for (i = 0; i < n; ++i) a[i] = i * i;
    return a;
}
```

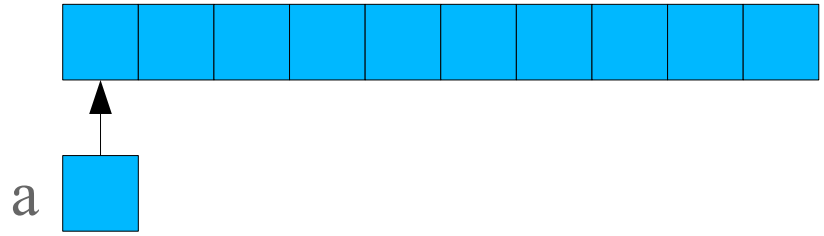
Der Aufrufer ist dann dafür verantwortlich, das Array wieder freizugeben, wenn es nicht mehr benötigt wird:

```
void test_quadratzahlen()
{
    int * q = berechne_quadratzahlen(10);
    int i;
    for (i = 0; i < 10; ++i) printf("%d ", q[i]);
    free(q);
}
```

Dynamische Arrays zurückgeben

Da die Lebensdauer dynamisch angeforderter Arrays von Sichtbarkeitsbereichen entkoppelt ist, können diese Arrays aus Funktionen zurückgegeben werden:

```
int * berechne_quadratzahlen(int n)
{
    int * a = malloc(n * sizeof(int));
    int i;
    for (i = 0; i < n; ++i) a[i] = i * i;
    return a;
}
```



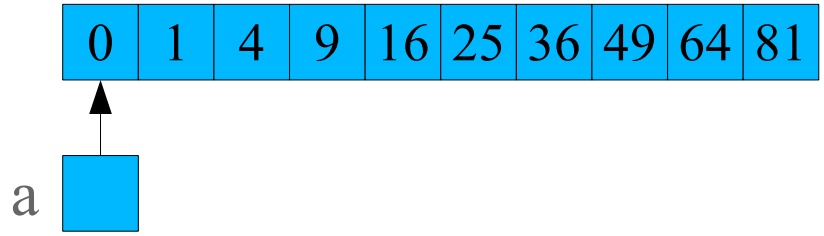
Der Aufrufer ist dann dafür verantwortlich, das Array wieder freizugeben, wenn es nicht mehr benötigt wird:

```
void test_quadratzahlen()
{
    int * q = berechne_quadratzahlen(10);
    int i;
    for (i = 0; i < 10; ++i) printf("%d ", q[i]);
    free(q);
}
```

Dynamische Arrays zurückgeben

Da die Lebensdauer dynamisch angeforderter Arrays von Sichtbarkeitsbereichen entkoppelt ist, können diese Arrays aus Funktionen zurückgegeben werden:

```
int * berechne_quadratzahlen(int n)
{
    int * a = malloc(n * sizeof(int));
    int i;
    for (i = 0; i < n; ++i) a[i] = i * i;
    return a;
}
```



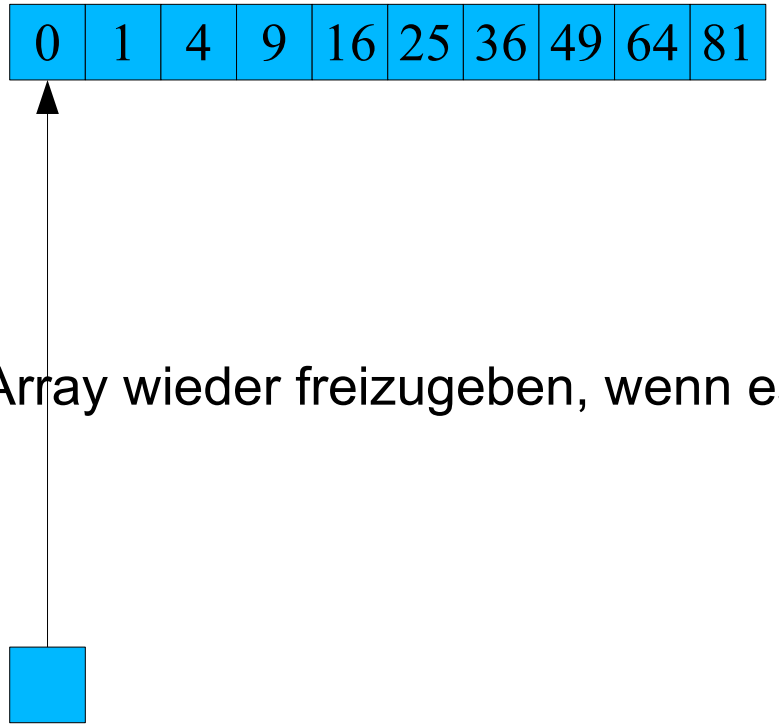
Der Aufrufer ist dann dafür verantwortlich, das Array wieder freizugeben, wenn es nicht mehr benötigt wird:

```
void test_quadratzahlen()
{
    int * q = berechne_quadratzahlen(10);
    int i;
    for (i = 0; i < 10; ++i) printf("%d ", q[i]);
    free(q);
}
```

Dynamische Arrays zurückgeben

Da die Lebensdauer dynamisch angeforderter Arrays von Sichtbarkeitsbereichen entkoppelt ist, können diese Arrays aus Funktionen zurückgegeben werden:

```
int * berechne_quadratzahlen(int n)
{
    int * a = malloc(n * sizeof(int));
    int i;
    for (i = 0; i < n; ++i) a[i] = i * i;
    return a;
}
```



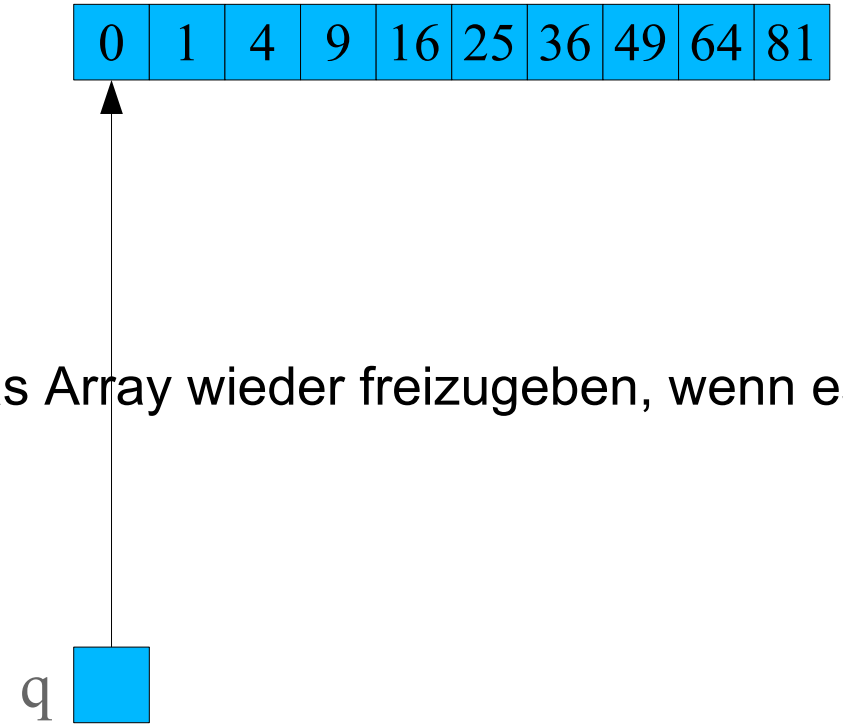
Der Aufrufer ist dann dafür verantwortlich, das Array wieder freizugeben, wenn es nicht mehr benötigt wird:

```
void test_quadratzahlen()
{
    int * q = berechne_quadratzahlen(10);
    int i;
    for (i = 0; i < 10; ++i) printf("%d ", q[i]);
    free(q);
}
```

Dynamische Arrays zurückgeben

Da die Lebensdauer dynamisch angeforderter Arrays von Sichtbarkeitsbereichen entkoppelt ist, können diese Arrays aus Funktionen zurückgegeben werden:

```
int * berechne_quadratzahlen(int n)
{
    int * a = malloc(n * sizeof(int));
    int i;
    for (i = 0; i < n; ++i) a[i] = i * i;
    return a;
}
```



Der Aufrufer ist dann dafür verantwortlich, das Array wieder freizugeben, wenn es nicht mehr benötigt wird:

```
void test_quadratzahlen()
{
    int * q = berechne_quadratzahlen(10);
    int i;
    for (i = 0; i < 10; ++i) printf("%d ", q[i]);
    free(q);
}
```

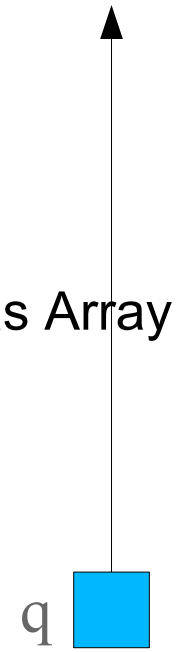
Dynamische Arrays zurückgeben

Da die Lebensdauer dynamisch angeforderter Arrays von Sichtbarkeitsbereichen entkoppelt ist, können diese Arrays aus Funktionen zurückgegeben werden:

```
int * berechne_quadratzahlen(int n)
{
    int * a = malloc(n * sizeof(int));
    int i;
    for (i = 0; i < n; ++i) a[i] = i * i;
    return a;
}
```

Der Aufrufer ist dann dafür verantwortlich, das Array wieder freizugeben, wenn es nicht mehr benötigt wird:

```
void test_quadratzahlen()
{
    int * q = berechne_quadratzahlen(10);
    int i;
    for (i = 0; i < 10; ++i) printf("%d ", q[i]);
    free(q);
}
```



Dynamische Arrays zurückgeben

Da die Lebensdauer dynamisch angeforderter Arrays von Sichtbarkeitsbereichen entkoppelt ist, können diese Arrays aus Funktionen zurückgegeben werden:

```
int * berechne_quadratzahlen(int n)
{
    int * a = malloc(n * sizeof(int));
    int i;
    for (i = 0; i < n; ++i) a[i] = i * i;
    return a;
}
```

Der Aufrufer ist dann dafür verantwortlich, das Array wieder freizugeben, wenn es nicht mehr benötigt wird:

```
void test_quadratzahlen()
{
    int * q = berechne_quadratzahlen(10);
    int i;
    for (i = 0; i < 10; ++i) printf("%d ", q[i]);
    free(q);
}
```

Relevante Funktionen in <stdlib.h>

```
void * malloc(int size_in_bytes);

int * p = malloc(100 * sizeof(int));


void * calloc(int number_of_elements, int size_of_one_element);

int * p = calloc(100, sizeof(int));


void * realloc(void * start_address, int new_size_in_bytes);

p = realloc(p, 200 * sizeof(int));


void free(void * start_address);

free(p);
```

Explizite Freigabe

- Jeder dynamisch angeforderte Speichbereich muss irgendwann explizit freigegeben werden.
  - Ansonsten gibt es ein Speicherleck.
- Oft geben eigene Funktion Zeiger auf dynamisch angeforderte Speicherbereiche zurück.
- Der Aufrufer ist anschließend verantwortlich dafür, den Speicher mittels free wieder freizugeben.
- Solche Verantwortlichkeiten müssen in C-Programmen gut dokumentiert sein, da es keine Form der automatischen Speicherfreigabe gibt!