1 Programmieren in C

1 Programmieren in C

- Unterschied zwischen Arrays und Zeigern: Adresse

1 Programmieren in C

- Unterschied zwischen Arrays und Zeigern: Adresse
- Parameter des Hauptprogramms: **int** argc, **char** $**$argv)

1 Programmieren in C

- Unterschied zwischen Arrays und Zeigern: Adresse
- Parameter des Hauptprogramms: **int** argc, **char** ∗∗argv)
- Strukturen:

```
typedef struct
{
   char day, month;
   int year;
}
date;

date today;
date *d = &today;

today.day = 18;
d->day++;
```

foo−>bar ist Abkürzung für (∗foo).bar

1 Programmieren in C

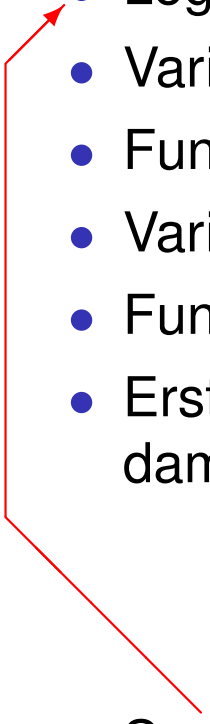
Programmiertips

- Richtig Einrücken.
- Logik sortieren, z. B. vom Komplizierten zum Einfachen.
- Variable kosten nichts.
- Funktionen auch nichts. Code-Verdopplung vermeiden!
- Variable sinnvoll benennen („sprechende Namen“).
- Funktionen auch.
- Erst konservativ, dann erst ambitioniert programmieren, damit man durch Vergleichen testen kann.

1 Programmieren in C

Programmiertips

- Richtig Einrücken.
- Logik sortieren, z. B. vom Komplizierten zum Einfachen.
- Variable kosten nichts.
- Funktionen auch nichts. Code-Verdopplung vermeiden!
- Variable sinnvoll benennen („sprechende Namen“).
- Funktionen auch.
- Erst konservativ, dann erst ambitioniert programmieren, damit man durch Vergleichen testen kann.

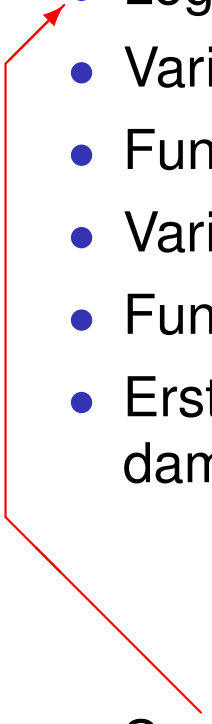


- Speziallfall des *Top-Down-Ansatzes*

1 Programmieren in C

Programmiertips

- Richtig Einrücken.
- Logik sortieren, z. B. vom Komplizierten zum Einfachen.
- Variable kosten nichts.
- Funktionen auch nichts. Code-Verdopplung vermeiden!
- Variable sinnvoll benennen („sprechende Namen“).
- Funktionen auch.
- Erst konservativ, dann erst ambitioniert programmieren, damit man durch Vergleichen testen kann.



- Speziallfall des *Top-Down-Ansatzes*
- Beispiel: Datum um 1 Tag inkrementieren

1.10 Dynamischer Speicher

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

int main (void)
{
   int *buffer = NULL;
   int count = 7;
   buffer = malloc (count * sizeof (int));
   for (int i = 0; i < count; i++)
      buffer[i] = i;
   for (int i = 0; i < count; i++)
      printf ("%d ", buffer[i]);
   printf ("\n");
   free (buffer);
   return 0;
}
```

1.10 Dynamischer Speicher

```c
#include <stdio.h>
#include <stdlib.h>

int main (void)
{
   int *buffer = NULL;
   int count = 7;
   buffer = malloc (count * sizeof (int));
   for (int i = 0; i < count; i++)
      buffer[i] = i;
   for (int i = 0; i < count; i++)
      printf ("%d ", buffer[i]);
   printf ("\n");
   free (buffer);
   return 0;
}
```

Man sieht buffer nicht an,
wie groß das Array ist.

1.10 Dynamischer Speicher

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

int main (void)
{
  int *buffer = NULL;
  int count = 7;
  buffer = malloc (count * sizeof (int));
  for (int i = 0; i < count; i++)
    buffer[i] = i;
  for (int i = 0; i < count; i++)
    printf ("%d ", buffer[i]);
  printf ("\n");
  free (buffer);
  return 0;
}
```

Man sieht buffer nicht an,
wie groß das Array ist.

$\longrightarrow$ buffer und count gemeinsam
in einem **struct** speichern

1.10 Dynamischer Speicher

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

int main (void)
{
  int *buffer = NULL;
  int count = 7;
  buffer = malloc (count * sizeof (int));
  for (int i = 0; i < count; i++)
    buffer[i] = i;
  for (int i = 0; i < count; i++)
    printf ("%d ", buffer[i]);
  printf ("\n");
  free (buffer);
  return 0;
}
```

Man sieht buffer nicht an,
wie groß das Array ist.

$\longrightarrow$ buffer und count gemeinsam
in einem **struct** speichern

Aufgabe

Schreiben Sie eine Funktion,
die in eine derartige „Container-Klasse“
ein Element in der Mitte einfügt.