1 Programmieren in C

1 Programmieren in C

- Variable

1 Programmieren in C

- Variable
- Zuweisungsoperatoren

1 Programmieren in C

- Variable
- Zuweisungsoperatoren:
  binäre Operatoren mit Seiteneffekt

1 Programmieren in C

- Variable
- Zuweisungsoperatoren:
  binäre Operatoren mit Seiteneffekt
- Funktionen schreiben

1 Programmieren in C

- Variable
- Zuweisungsoperatoren:
  binäre Operatoren mit Seiteneffekt
- Funktionen schreiben:
  ISO- vs. K&R-C

1 Programmieren in C

- Variable
- Zuweisungsoperatoren:
  binäre Operatoren mit Seiteneffekt
- Funktionen schreiben:
  ISO- vs. K&R-C
  Datentyp **void**

1 Programmieren in C

- Variable
- Zuweisungsoperatoren:
  binäre Operatoren mit Seiteneffekt
- Funktionen schreiben:
  ISO- vs. K&R-C
  Datentyp **void**
- Zeiger

1 Programmieren in C

- Variable
- Zuweisungsoperatoren:
  binäre Operatoren mit Seiteneffekt
- Funktionen schreiben:
  ISO- vs. K&R-C
  Datentyp **void**
- Zeiger
  Zeiger-Arithmetik

1 Programmieren in C

- Variable
- Zuweisungsoperatoren:
  binäre Operatoren mit Seiteneffekt
- Funktionen schreiben:
  ISO- vs. K&R-C
  Datentyp **void**
- Zeiger
  Zeiger-Arithmetik
- Arrays

1 Programmieren in C

- Variable
- Zuweisungsoperatoren:
  binäre Operatoren mit Seiteneffekt
- Funktionen schreiben:
  ISO- vs. K&R-C
  Datentyp **void**
- Zeiger
  Zeiger-Arithmetik
- Arrays
  . . . sind i. w. dasselbe wie Zeiger

1 Programmieren in C

- Variable
- Zuweisungsoperatoren:
  binäre Operatoren mit Seiteneffekt
- Funktionen schreiben:
  ISO- vs. K&R-C
  Datentyp **void**
- Zeiger
  Zeiger-Arithmetik
- Arrays
  . . . sind i. w. dasselbe wie Zeiger:
  foo[i] ist Abkürzung für $*$(foo + i)

1 Programmieren in C

- Variable
- Zuweisungsoperatoren:
  binäre Operatoren mit Seiteneffekt
- Funktionen schreiben:
  ISO- vs. K&R-C
  Datentyp **void**
- Zeiger
  Zeiger-Arithmetik
- Arrays
  . . . sind i. w. dasselbe wie Zeiger:
  foo[i] ist Abkürzung für $*$(foo + i)
- Strings

1 Programmieren in C

- Variable
- Zuweisungsoperatoren:
  binäre Operatoren mit Seiteneffekt
- Funktionen schreiben:
  ISO- vs. K&R-C
  Datentyp **void**
- Zeiger
  Zeiger-Arithmetik
- Arrays
  . . . sind i. w. dasselbe wie Zeiger:
  foo[i] ist Abkürzung für $*$(foo + i)
- Strings
  . . . sind $\left\{ \begin{array}{c} \text{Arrays von} \\ \text{Zeiger auf} \end{array} \right\}$ **char**s

1 Programmieren in C

- Variable
- Zuweisungsoperatoren:
  binäre Operatoren mit Seiteneffekt
- Funktionen schreiben:
  ISO- vs. K&R-C
  Datentyp **void**
- Zeiger
  Zeiger-Arithmetik
- Arrays
  . . . sind i. w. dasselbe wie Zeiger:
  foo[i] ist Abkürzung für $*(foo + i)$
- Strings
  . . . sind $\left\{ \begin{array}{c} \text{Arrays von} \\ \text{Zeiger auf} \end{array} \right\}$ **char**s,
  d. h. auf ganze Zahlen

1 Programmieren in C

- Variable
- Zuweisungsoperatoren:
  binäre Operatoren mit Seiteneffekt
- Funktionen schreiben:
  ISO- vs. K&R-C
  Datentyp **void**
- Zeiger
  Zeiger-Arithmetik
- Arrays
  . . . sind i. w. dasselbe wie Zeiger:
  foo[i] ist Abkürzung für $*(foo + i)$
- Strings
  . . . sind $\left\{ \begin{array}{c} \text{Arrays von} \\ \text{Zeiger auf} \end{array} \right\}$ **char**s,
  d. h. auf ganze Zahlen

1.8 Arrays und Strings

```
#include <stdio.h>

void foo (int *x)
{
   static int counter = 1;
   printf ("foo(%d): %d\n", counter++, *x);
}

int main (void)
{
   int a[3] = { 13, 14, 15 };
   int *b = a;
   foo (a);
   foo (&a);
   foo (b);
   foo (&b);
   return 0;
}
```

1.8 Arrays und Strings

```
#include <stdio.h>

void foo (int *x)
{
   static int counter = 1;
   printf ("foo(%d): %d\n", counter++, *x);
}

int main (void)
{
   int a[3] = { 13, 14, 15 };
   int *b = a;
   foo (a);    <-- Adresse des Arrays, also der 13
   foo (&a);
   foo (b);
   foo (&b);
   return 0;
}
```

1.8 Arrays und Strings

```
#include <stdio.h>

void foo (int *x)
{
   static int counter = 1;
   printf ("foo(%d): %d\n", counter++, *x);
}

int main (void)
{
   int a[3] = { 13, 14, 15 };
   int *b = a;
   foo (a);    <-- Adresse des Arrays, also der 13
   foo (&a);
   foo (b);    <-- dasselbe wie a: Adresse der 13
   foo (&b);
   return 0;
}
```

1.8 Arrays und Strings

```
#include <stdio.h>

void foo (int *x)
{
   static int counter = 1;
   printf ("foo(%d): %d\n", counter++, *x);
}

int main (void)
{
   int a[3] = { 13, 14, 15 };
   int *b = a;
   foo (a);   <-- Adresse des Arrays, also der 13
   foo (&a);
   foo (b);   <-- dasselbe wie a: Adresse der 13
   foo (&b); <-- Adresse von b
   return 0;
}
```

1.8 Arrays und Strings

```
#include <stdio.h>

void foo (int *x)
{
   static int counter = 1;
   printf ("foo(%d): %d\n", counter++, *x);
}

int main (void)
{
   int a[3] = { 13, 14, 15 };
   int *b = a;
   foo (a);   <-- Adresse des Arrays, also der 13
   foo (&a); <-- dasselbe wie a: Adresse der 13
   foo (b);   <-- dasselbe wie a: Adresse der 13
   foo (&b); <-- Adresse von b
   return 0;
}
```

1.8 Arrays und Strings

Parameter des Hauptprogramms

```
#include <stdio.h>

int main (int argc, char **argv)
{
   int i = 0;
   printf ("argc = %d\n", argc);
   while (*argv)
      printf ("argv[%d] = \"%s\"\n", i++, *argv++);
   return 0;
}
```

1.8 Arrays und Strings

Parameter des Hauptprogramms

```
#include <stdio.h>

int main (int argc, char **argv)
{
   int i = 0;
   char **p = argv;
   printf ("argc = %d\n", argc);
   while (*p)
      printf ("argv[%d] = \"%s\"\n", i++, *p++);
   return 0;
}
```

1.8 Arrays und Strings

Parameter des Hauptprogramms

```
#include <stdio.h>

int main (int argc, char **argv)
{
   int i = 0;
   printf ("argc = %d\n", argc);
   while (argv[i])
      printf ("argv[%d] = \"%s\"\n", i, argv[i++]);
   return 0;
}
```

1.8 Arrays und Strings

Parameter des Hauptprogramms

```
#include <stdio.h>

int main (int argc, char **argv)
{
   printf ("argc = %d\n", argc);
   for (int i = 0; argv[i]; i++)
      printf ("argv[%d] = \"%s\"\n", i, argv[i]);
   return 0;
}
```

1.9 Strukturen

```
#include <stdio.h>

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

int main (void)
{
   date today = { 11, 4, 2012 };
   printf ("%d.%d.%d\n", today.day, today.month, today.year);
   return 0;
}
```

1.9 Strukturen

```
#include <stdio.h>

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

void get_date (date *d)
{
   (*d).day = 11;
   (*d).month = 4;
   (*d).year = 2012;
}
```

```
int main (void)
{
   date today;
   get_date (&today);
   printf ("%d.%d.%d\n", today.day,
            today.month, today.year);
   return 0;
}
```

1.9 Strukturen

```
#include <stdio.h>

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

void get_date (date *d)
{
   d->day = 11;
   d->month = 4;
   d->year = 2012;
}
```

foo->bar ist Abkürzung für (*foo).bar

```
int main (void)
{
   date today;
   get_date (&today);
   printf ("%d.%d.%d\n", today.day,
            today.month, today.year);
   return 0;
}
```

1.9 Strukturen

```
#include <stdio.h>

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

void get_date (date *d)
{
   d->day = 11;
   d->month = 4;
   d->year = 2012;
}
```

Aufgabe

Schreiben Sie eine Funktion
add_date (date *d, **int** days),
die auf ein gegebenes Datum d
days Tage addiert.

```
int main (void)
{
   date today;
   get_date (&today);
   printf ("%d.%d.%d\n", today.day,
            today.month, today.year);
   return 0;
}
```

Tag-Nr. seit 1. 1. 1980 in Jahr + Tag umrechnen

```
year = ORIGINYEAR; /* = 1980 */

while (days > 365)
  {
    if (IsLeapYear (year))
      {
        if (days > 366)
          {
            days -= 366;
            year += 1;
          }
      }
    else
      {
        days -= 365;
        year += 1;
      }
  }
```

Tag-Nr. seit 1. 1. 1980 in Jahr + Tag umrechnen

```
year = ORIGINYEAR; /* = 1980 */

while (days > 365)
  {
    if (IsLeapYear (year))
      {
        if (days > 366)
          {
            days -= 366;
            year += 1;
          }
      }
    else
      {
        days -= 365;
        year += 1;
      }
  }
```

Kunden des ersten mobilen Media-Players von Microsoft erlebten zum Jahresende eine böse Überraschung: Am 31. Dezember 2008 fielen weltweit alle Zune-Geräte der ersten Generation aus. Ursache war ein interner Fehler bei der Handhabung von Schaltjahren.

http://heise.de/-193332, 3. 1. 2009

1.9 Strukturen

```
#include <stdio.h>

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

int main (void)
{
   date today = { 11, 4, 2012 };
   printf ("%d.%d.%d\n", today.day, today.month, today.year);
   return 0;
}
```

1.9 Strukturen

ISO-C

```
#include <stdio.h>

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

int main (void)
{
   date today = { 11, 4, 2012 };
   printf ("%d.%d.%d\n", today.day, today.month, today.year);
   return 0;
}
```

1.9 Strukturen

K&R-C

```
#include <stdio.h>

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


int main (void)
{
   struct date today = { 11, 4, 2012 };
   printf ("%d.%d.%d\n", today.day, today.month, today.year);
   return 0;
}
```