

Angewandte Informatik

Prof. Dr. Peter Gerwinski

18. Oktober 2012

2 Einführung in C

2.7 Seiteneffekte

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

```
int main (void)
```

```
{  
    int a = printf ("%d\n", 42);  
    printf ("%d\n", a);  
    return 0;  
}
```

- `printf()` ist eine Funktion.
- „Haupteffekt“: Wert zurückliefern
(hier: Anzahl der ausgegebenen Zeichen)
- *Seiteneffekt*: Ausgabe

2 Einführung in C

2.7 Seiteneffekte bei Operatoren

Unäre Operatoren:

- Negation: `-foo`
- Funktionsaufruf: `foo ()`
- Post-Dekrement: `foo--`
- Post-Inkrement: `foo++`
- Prä-Dekrement: `--foo`
- Prä-Inkrement: `++foo`

Binäre Operatoren:

- Rechnen: `+ - * / %`
- Vergleich: `== != < > <= >=`
- Zuweisung: `= += -= *= /= %=`

2 Einführung in C

2.7 Seiteneffekte bei Operatoren

Unäre Operatoren:

- Negation: `-foo`
- Funktionsaufruf: `foo ()`
- Post-Dekrement: `foo--`
- Post-Inkrement: `foo++`
- Prä-Dekrement: `--foo`
- Prä-Inkrement: `++foo`

Binäre Operatoren:

- Rechnen: `+` `-` `*` `/` `%`
- Vergleich: `==` `!=` `<` `>` `<=` `>=`
- Zuweisung: `=` `+=` `-=` `*=` `/=` `%=`

rot = mit Seiteneffekt

2 Einführung in C

2.7 Seiteneffekte bei Operatoren

Unäre Operatoren:

- Negation: `—foo`
- Funktionsaufruf: `foo ()`
- Post-Dekrement: `foo—`
- Post-Inkrement: `foo++`
- Prä-Dekrement: `—foo`
- Prä-Inkrement: `++foo`

```
int i;
```

```
i = 0;
```

```
while (i < 10)
```

```
{
```

```
    printf ("%d\n", i);
```

```
    i += 1;
```

```
}
```

Binäre Operatoren:

- Rechnen: `+ — * / %`
- Vergleich: `== != < > <= >=`
- Zuweisung: `= += -= *= /= %=`

rot = mit Seiteneffekt

2 Einführung in C

2.7 Seiteneffekte bei Operatoren

Unäre Operatoren:

- Negation: `—foo`
- Funktionsaufruf: `foo ()`
- Post-Dekrement: `foo—`
- Post-Inkrement: `foo++`
- Prä-Dekrement: `—foo`
- Prä-Inkrement: `++foo`

Binäre Operatoren:

- Rechnen: `+ — * / %`
- Vergleich: `== != < > <= >=`
- Zuweisung: `= += -= *= /= %=`

rot = mit Seiteneffekt

```
int i;
```

```
i = 0;
```

```
while (i < 10)
```

```
{
```

```
    printf ("%d\n", i);
```

```
    i += 1;
```

```
}
```

```
for (i = 0; i < 10; i++)
```

```
    printf ("%d\n", i);
```

2 Einführung in C

2.7 Seiteneffekte bei Operatoren

Unäre Operatoren:

- Negation: `¬foo`
- Funktionsaufruf: `foo ()`
- Post-Dekrement: `foo--`
- Post-Inkrement: `foo++`
- Prä-Dekrement: `--foo`
- Prä-Inkrement: `++foo`

Binäre Operatoren:

- Rechnen: `+` `-` `*` `/` `%`
- Vergleich: `==` `!=` `<` `>` `<=` `>=`
- Zuweisung: `=` `+=` `-=` `*=` `/=` `%=`

rot = mit Seiteneffekt

```
int i;
```

```
i = 0;  
while (i < 10)  
{  
    printf ("%d\n", i);  
    i += 1;  
}
```

```
for (i = 0; i < 10; i++)  
    printf ("%d\n", i);
```

```
i = 0;  
while (i < 10)  
    printf ("%d\n", i++);
```

2 Einführung in C

2.7 Seiteneffekte bei Operatoren

Unäre Operatoren:

- Negation: `—foo`
- Funktionsaufruf: `foo ()`
- Post-Dekrement: `foo—`
- Post-Inkrement: `foo++`
- Prä-Dekrement: `—foo`
- Prä-Inkrement: `++foo`

Binäre Operatoren:

- Rechnen: `+ — * / %`
- Vergleich: `== != < > <= >=`
- Zuweisung: `= += -= *= /= %=`

rot = mit Seiteneffekt

```
int i;
```

```
i = 0;
while (i < 10)
{
    printf ("%d\n", i);
    i += 1;
}
```

```
for (i = 0; i < 10; i++)
    printf ("%d\n", i);
```

```
i = 0;
while (i < 10)
    printf ("%d\n", i++);
```

```
for (i = 0; i < 10; printf ("%d\n", i++));
```


2 Einführung in C

```
int i;
```

```
i = 0;
```

```
while (i < 10)
```

```
{
```

```
    printf ("%d\n", i);
```

```
    i += 1;
```

```
}
```

```
for (i = 0; i < 10; i++)
```

```
    printf ("%d\n", i);
```

```
i = 0;
```

```
while (i < 10)
```

```
    printf ("%d\n", i++);
```

```
for (i = 0; i < 10; printf ("%d\n", i++));
```

2 Einführung in C

2.8 Strukturierte Programmierung

```
int i;
```

```
i = 0;
```

```
while (i < 10)
```

```
{
```

```
    printf ("%d\n", i);
```

```
    i += 1;
```

```
}
```

```
for (i = 0; i < 10; i++)
```

```
    printf ("%d\n", i);
```

```
i = 0;
```

```
while (i < 10)
```

```
    printf ("%d\n", i++);
```

```
for (i = 0; i < 10; printf ("%d\n", i++));
```

2 Einführung in C

2.8 Strukturierte Programmierung

```
i = 0;  
while (1)  
{  
    if (i >= 10)  
        break;  
    printf ("%d\n", i++);  
}
```

```
int i;
```

```
i = 0;  
while (i < 10)  
{  
    printf ("%d\n", i);  
    i += 1;  
}
```

```
for (i = 0; i < 10; i++)  
    printf ("%d\n", i);
```

```
i = 0;  
while (i < 10)  
    printf ("%d\n", i++);
```

```
for (i = 0; i < 10; printf ("%d\n", i++));
```

2 Einführung in C

2.8 Strukturierte Programmierung

```
i = 0;
while (1)
{
    if (i >= 10)
        break;
    printf ("%d\n", i++);
}
```

```
i = 0;
loop:
if (i >= 10)
    goto endloop;
printf ("%d\n", i++);
goto loop;
endloop:
```

```
int i;

i = 0;
while (i < 10)
{
    printf ("%d\n", i);
    i += 1;
}
```

```
for (i = 0; i < 10; i++)
    printf ("%d\n", i);
```

```
i = 0;
while (i < 10)
    printf ("%d\n", i++);
```

```
for (i = 0; i < 10; printf ("%d\n", i++));
```

2 Einführung in C

2.8 Strukturierte Programmierung

```
i = 0;
while (1)      fragwürdig
{
    if (i >= 10)
        break;
    printf ("%d\n", i++);
}
```

```
i = 0;
loop:         sehr fragwürdig
if (i >= 10)
    goto endloop;
printf ("%d\n", i++);
goto loop;
endloop:
```

```
int i;

i = 0;
while (i < 10)    gut
{
    printf ("%d\n", i);
    i += 1;
}
```

```
for (i = 0; i < 10; i++)
    printf ("%d\n", i);
```

nur, wenn
Sie wissen,
was Sie tun

```
i = 0;
while (i < 10)
    printf ("%d\n", i++);
```

```
for (i = 0; i < 10; printf ("%d\n", i++));
```

2 Einführung in C

2.9 Funktionen

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

```
int answer (void)
```

```
{  
    return 42;  
}
```

```
void foo (void)
```

```
{  
    printf ("%d\n", answer ());  
}
```

```
int main (void)
```

```
{  
    foo ();  
    return 0;  
}
```

2 Einführung in C

2.9 Funktionen

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

```
int answer (void)
{
    return 42;
}
```

```
void foo (void)
{
    printf ("%d\n", answer ());
}
```

```
int main (void)
{
    foo ();
    return 0;
}
```

- Funktionsdeklaration:
Typ Name (Parameterliste)
{
 Anweisungen
}

2 Einführung in C

2.9 Funktionen

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

```
void add_verbose (int a, int b)
{
    printf ("%d_+_%d=_%d\n", a, b, a + b);
}
```

```
int main (void)
{
    add_verbose (3, 7);
    return 0;
}
```

- Funktionsdeklaration:
Typ Name (Parameterliste)
{
 Anweisungen
}

2 Einführung in C

2.9 Funktionen

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

```
void add_verbose (int a, int b)
{
    printf ("%d_+_%d=_%d\n", a, b, a + b);
}
```

```
int main (void)
{
    add_verbose (3, 7);
    return 0;
}
```

- Funktionsdeklaration:
Typ Name (Parameterliste)
{
 Anweisungen
}
- Der Datentyp **void** steht für „nichts“

2 Einführung in C

2.9 Funktionen

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

```
void add_verbose (int a, int b)
{
    printf ("%d_+_%d=_%d\n", a, b, a + b);
}
```

```
int main (void)
{
    add_verbose (3, 7);
    return 0;
}
```

- Funktionsdeklaration:
Typ Name (Parameterliste)
{
 Anweisungen
}
- Der Datentyp **void**
steht für „nichts“
und kann ignoriert werden.

2 Einführung in C

2.9 Funktionen

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

```
void add_verbose (int a, int b)
{
    printf ("%d_+_%d=_%d\n", a, b, a + b);
}
```

```
int main (void)
{
    add_verbose (3, 7);
    return 0;
}
```

- Funktionsdeklaration:
Typ Name (Parameterliste)
{
 Anweisungen
}
- Der Datentyp **void** steht für „nichts“ und muß ignoriert werden.

2 Einführung in C

2.9 Funktionen

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

```
int a, b = 3;
```

```
void foo (void)
```

```
{  
    b++;  
    static int a = 5;  
    int b = 7;  
    printf ("foo():_"  
           "a=_%d,_b=_%d\n",  
           a, b);  
    a++;  
}
```

```
int main (void)
```

```
{  
    printf ("main():_"  
           "a=_%d,_b=_%d\n",  
           a, b);  
    foo ();  
    printf ("main():_"  
           "a=_%d,_b=_%d\n",  
           a, b);  
    a = b = 12;  
    printf ("main():_"  
           "a=_%d,_b=_%d\n",  
           a, b);  
    foo ();  
    printf ("main():_"  
           "a=_%d,_b=_%d\n",  
           a, b);  
    return 0;  
}
```

2 Einführung in C

2.10 Zeiger

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

```
void calc_answer (int *a)  
{  
    *a = 42;  
}
```

```
int main (void)  
{  
    int answer;  
    calc_answer (&answer);  
    printf ("The_answer_is_%d.\n", answer);  
    return 0;  
}
```

2 Einführung in C

2.10 Zeiger

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

```
void calc_answer (int *a)
```

```
{  
    *a = 42;  
}
```

- *a ist eine **int**.

```
int main (void)
```

```
{  
    int answer;  
    calc_answer (&answer);  
    printf ("The_answer_is_%d.\n", answer);  
    return 0;  
}
```

2 Einführung in C

2.10 Zeiger

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

```
void calc_answer (int *a)
{
    *a = 42;
}
```

- `*a` ist eine **int**.
- unärer Operator `*`:
Pointer-Dereferenzierung

```
int main (void)
{
    int answer;
    calc_answer (&answer);
    printf ("The_answer_is_%d.\n", answer);
    return 0;
}
```

2 Einführung in C

2.10 Zeiger

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

```
void calc_answer (int *a)
{
    *a = 42;
}
```

- `*a` ist eine **int**.
- unärer Operator `*`:
Pointer-Dereferenzierung

→ `a` ist ein Zeiger (Pointer) auf eine **int**.

```
int main (void)
{
    int answer;
    calc_answer (&answer);
    printf ("The_answer_is_%d.\n", answer);
    return 0;
}
```


2 Einführung in C

2.10 Zeiger

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

```
void calc_answer (int *a)
{
    *a = 42;
}
```

- `*a` ist eine **int**.
- unärer Operator `*`:
Pointer-Dereferenzierung

→ `a` ist ein Zeiger (Pointer) auf eine **int**.

```
int main (void)
{
    int answer;
    calc_answer (&answer);
    printf ("The_answer_is_%d.\n", answer);
    return 0;
}
```

- unärer Operator `&`: Adresse

2 Einführung in C

2.11 Arrays und Strings

Ein Zeiger zeigt auf eine Variable.

2 Einführung in C

2.11 Arrays und Strings

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

2 Einführung in C

2.11 Arrays und Strings

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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

```
int main (void)
```

```
{
```

```
    int prime[5] = { 2, 3, 5, 7, 11 };
```

```
    int *p = prime, i = 0;
```

```
    while (i < 5)
```

```
        printf ("%d\n", *(p + i++));
```

```
    return 0;
```

```
}
```

2 Einführung in C

2.11 Arrays und Strings

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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

```
int main (void)
```

```
{
```

```
    int prime[5] = { 2, 3, 5, 7, 11 };
```

```
    int *p = prime, i = 0;
```

```
    while (i < 5)
```

```
        printf ("%d\n", *(p + i++));
```

```
    return 0;
```

```
}
```

- `prime` ist eine Ansammlung von fünf ganzen Zahlen.

2 Einführung in C

2.11 Arrays und Strings

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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

```
int main (void)
```

```
{
```

```
    int prime[5] = { 2, 3, 5, 7, 11 };
```

```
    int *p = prime, i = 0;
```

```
    while (i < 5)
```

```
        printf ("%d\n", *(p + i++));
```

```
    return 0;
```

```
}
```

- `prime` ist ein Array von fünf ganzen Zahlen.

2 Einführung in C

2.11 Arrays und Strings

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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

```
int main (void)
```

```
{  
    int prime[5] = { 2, 3, 5, 7, 11 };  
    int *p = prime, i = 0;  
    while (i < 5)  
        printf ("%d\n", *(p + i++));  
    return 0;  
}
```

- `prime` ist ein Array von fünf ganzen Zahlen.
- `prime` ist ein Zeiger auf eine `int`.

2 Einführung in C


2.11 Arrays und Strings

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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

```
int main (void)
```

```
{  
    int prime[5] = { 2, 3, 5, 7, 11 };  
    int *p = prime, i = 0;  
    while (i < 5)  
        printf ("%d\n", *(p + i++));  
    return 0;  
}
```

- `prime` ist ein Array von fünf ganzen Zahlen.
- `prime` ist ein Zeiger auf eine `int`. 
- `p + i` ist ein Zeiger auf den `i`-ten Nachbarn von `*p`.

2 Einführung in C


2.11 Arrays und Strings

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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

```
int main (void)
```

```
{  
    int prime[5] = { 2, 3, 5, 7, 11 };  
    int *p = prime, i = 0;  
    while (i < 5)  
        printf ("%d\n", *(p + i++));  
    return 0;  
}
```

- `prime` ist ein Array von fünf ganzen Zahlen.
- `prime` ist ein Zeiger auf eine `int`. 
- `p + i` ist ein Zeiger auf den `i`-ten Nachbarn von `*p`.
- `*(p + i)` ist der `i`-te Nachbar von `*p`.

2 Einführung in C


2.11 Arrays und Strings

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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

```
int main (void)
```

```
{  
    int prime[5] = { 2, 3, 5, 7, 11 };  
    int *p = prime, i = 0;  
    while (i < 5)  
        printf ("%d\n", *(p + i++));  
    return 0;  
}
```

- `prime` ist ein Array von fünf ganzen Zahlen.
- `prime` ist ein Zeiger auf eine `int`. 
- `p + i` ist ein Zeiger auf den `i`-ten Nachbarn von `*p`.
- `*(p + i)` ist der `i`-te Nachbar von `*p`.
- Andere Schreibweise: `p[i]` statt `*(p + i)`

2 Einführung in C


2.11 Arrays und Strings

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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

```
int main (void)
```

```
{  
    int prime[5] = { 2, 3, 5, 7, 11 };  
    int i = 0;  
    while (i < 5)  
        printf ("%d\n", prime[i++]);  
    return 0;  
}
```

- `prime` ist ein Array von fünf ganzen Zahlen.
- `prime` ist ein Zeiger auf eine `int`. 
- `p + i` ist ein Zeiger auf den `i`-ten Nachbarn von `*p`.
- `*(p + i)` ist der `i`-te Nachbar von `*p`.
- Andere Schreibweise: `p[i]` statt `*(p + i)`

2 Einführung in C

2.11 Arrays und Strings

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

```
int main (void)
```

```
{
```

```
    char hello_world[] = "Hello,_world!\n";
```

```
    int i = 0;
```

```
    while (hello_world[i] != 0)
```

```
        printf ("%d", hello_world[i++]);
```

```
    return 0;
```

```
}
```

2 Einführung in C

2.11 Arrays und Strings

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

```
int main (void)
```

```
{
```

```
    char hello_world[] = "Hello,_world!\n";
```

```
    int i = 0;
```

```
    while (hello_world[i])
```

```
        printf ("%d", hello_world[i++]);
```

```
    return 0;
```

```
}
```

2 Einführung in C

2.11 Arrays und Strings

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

```
int main (void)
```

```
{
```

```
    char hello_world[] = "Hello,_world!\n";
```

```
    char *p = hello_world;
```

```
    while (*p)
```

```
        printf ("%d", *p++);
```

```
    return 0;
```

```
}
```

2 Einführung in C

2.11 Arrays und Strings

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

```
int main (void)
```

```
{
```

```
    char hello_world[] = "Hello,_world!\n";
```

```
    char *p = hello_world;
```

```
    while (*p)
```

```
        printf ("%c", *p++);
```

```
    return 0;
```

```
}
```

2 Einführung in C

2.11 Arrays und Strings

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

```
int main (void)
```

```
{
```

```
    char hello_world[] = "Hello,_world!\n";
```

```
    char *p = hello_world;
```

```
    while (*p)
```

```
        printf ("%c", *p++);
```

```
    return 0;
```

```
}
```

- Ein **char** ist eine kleinere **int**.

2 Einführung in C

2.11 Arrays und Strings

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

```
int main (void)
```

```
{
```

```
    char hello_world[] = "Hello,_world!\n";
```

```
    char *p = hello_world;
```

```
    while (*p)
```

```
        printf ("%c", *p++);
```

```
    return 0;
```

```
}
```

- Ein **char** ist eine kleinere **int**.
- Ein „String“ in C ist ein Array von **chars**.

2 Einführung in C

2.11 Arrays und Strings

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

```
int main (void)
```

```
{
```

```
    char hello_world[] = "Hello,_world!\n";
```

```
    char *p = hello_world;
```

```
    while (*p)
```

```
        printf ("%c", *p++);
```

```
    return 0;
```

```
}
```

- Ein **char** ist eine kleinere **int**.
- Ein „String“ in C ist ein Array von **chars**, also ein Zeiger auf **chars**.

2 Einführung in C

2.11 Arrays und Strings

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

```
int main (void)
```

```
{
```

```
    char hello_world[] = "Hello,_world!\n";
```

```
    char *p = hello_world;
```

```
    while (*p)
```

```
        printf ("%c", *p++);
```

```
    return 0;
```

```
}
```

- Ein **char** ist eine kleinere **int**.
- Ein „String“ in C ist ein Array von **chars**, also ein Zeiger auf **chars**
also ein Zeiger auf (kleinere) Integer.

2 Einführung in C

2.11 Arrays und Strings

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

```
int main (void)
```

```
{
```

```
    char hello_world[] = "Hello,_world!\n";
```

```
    char *p = hello_world;
```

```
    while (*p)
```

```
        printf ("%c", *p++);
```

```
    return 0;
```

```
}
```

- Ein **char** ist eine kleinere **int**.
- Ein „String“ in C ist ein Array von **chars**, also ein Zeiger auf **chars** also ein Zeiger auf (kleinere) Integer.
- Die Formatspezifikation entscheidet über die Ausgabe:
 - %d** dezimal
 - %x** hexadezimal
 - %c** Zeichen

2 Einführung in C

2.11 Arrays und Strings

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

```
int main (void)
```

```
{
```

```
    char hello_world[] = "Hello,_world!\n";
```

```
    char *p = hello_world;
```

```
    while (*p)
```

```
        printf ("%c", *p++);
```

```
    return 0;
```

```
}
```

- Ein **char** ist eine kleinere **int**.
- Ein „String“ in C ist ein Array von **chars**, also ein Zeiger auf **chars** also ein Zeiger auf (kleinere) Integer.
- Die Formatspezifikation entscheidet über die Ausgabe:
 - %d** dezimal
 - %x** hexadezimal
 - %c** Zeichen
 - %s** String