

Angewandte Informatik

Prof. Dr. Peter Gerwinski

8. November 2012

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>
```

```
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

Gültigkeitsbereiche von Variablen

```
#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;
}
```

- `*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 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));  
        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]);  
        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 };  
    for (int i = 0; i < 5; i++)  
        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] != 0)
```

```
        printf ("%c", 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 ("%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, abgeschlossen mit dem **char 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;
```

```
}
```

- Die Formatspezifikation entscheidet über die Ausgabe:

%d dezimal

%x hexadezimal

%c Zeichen

- 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, abgeschlossen mit dem **char 0**.

2 Einführung in C

2.11 Arrays und Strings

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

```
int main (void)
{
    char *p = "Hello, world!";
    printf ("%s\n", p);
    return 0;
}
```

- Die Formatspezifikation entscheidet über die Ausgabe:

%d dezimal

%x hexadezimal

%c Zeichen

%s String

- 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, abgeschlossen mit dem **char 0**.

2 Einführung in C

2.11 Arrays und Strings

Parameter des Hauptprogramms

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

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

- **int argc**: Anzahl der Parameter
- **char **argv**: Zeiger auf Zeiger auf **char**
= Array von Arrays von **char**
= Array von Strings

2 Einführung in C

2.11 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] != NULL; i++)
        printf ("argv[%d] = \"%s\"\n", i, argv[i]);
    return 0;
}
```

- **int argc**: Anzahl der Parameter
- **char **argv**: Zeiger auf Zeiger auf **char**
= Array von Arrays von **char**
= Array von Strings,
abgeschlossen mit dem String **NULL**
(= Zeiger auf „nichts“)

2 Einführung in C

2.12 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;  
}
```

2 Einführung in C

2.12 Strukturen

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

```
typedef struct
```

```
{  
    char day, month;  
    int year;  
}
```

```
date;
```

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

```
int main (void)
```

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

2 Einführung in C

2.12 Strukturen

foo->bar ist Abkürzung für (*foo).bar

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

```
typedef struct
```

```
{  
    char day, month;  
    int year;  
}
```

```
date;
```

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

```
int main (void)
```

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

2 Einführung in C

2.12 Strukturen

foo->bar ist Abkürzung für (*foo).bar

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

```
typedef struct
```

```
{  
    char day, month;  
    int year;  
}
```

```
date;
```

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

```
int main (void)
```

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

Aufgabe

Schreiben Sie eine Funktion `inc_date (date *d)`, die ein gegebenes Datum `d` auf den nächsten Tag setzt. Schaltjahre berücksichtigen.

2 Einführung in C

Sprachelemente weitgehend komplett

2 Einführung in C

Sprachelemente weitgehend komplett

Es fehlen:

- Ergänzungen (z. B. ternärer Operator, **union**, **unsigned**, **volatile**)

2 Einführung in C

Sprachelemente weitgehend komplett

Es fehlen:

- Ergänzungen (z. B. ternärer Operator, **union**, **unsigned**, **volatile**)
- Bibliotheksfunktionen (z. B. **malloc()**)

2 Einführung in C

Sprachelemente weitgehend komplett

Es fehlen:

- Ergänzungen (z. B. ternärer Operator, **union**, **unsigned**, **volatile**)
- Bibliotheksfunktionen (z. B. **malloc()**)

→ werden eingeführt, wenn wir sie brauchen

2 Einführung in C

Sprachelemente weitgehend komplett

Es fehlen:

- Ergänzungen (z. B. ternärer Operator, **union**, **unsigned**, **volatile**)
- Bibliotheksfunktionen (z. B. **malloc()**)

→ werden eingeführt, wenn wir sie brauchen

- Konzepte (z. B. rekursive Datenstrukturen, Klassen selbst bauen)

2 Einführung in C

Sprachelemente weitgehend komplett

Es fehlen:

- Ergänzungen (z. B. ternärer Operator, **union**, **unsigned**, **volatile**)
- Bibliotheksfunktionen (z. B. **malloc()**)

→ werden eingeführt, wenn wir sie brauchen

- Konzepte (z. B. rekursive Datenstrukturen, Klassen selbst bauen)

→ werden eingeführt, wenn wir sie brauchen, oder:

→ Literatur

2 Einführung in C

Sprachelemente weitgehend komplett

Es fehlen:

- Ergänzungen (z. B. ternärer Operator, **union**, **unsigned**, **volatile**)
- Bibliotheksfunktionen (z. B. **malloc()**)

→ werden eingeführt, wenn wir sie brauchen

- Konzepte (z. B. rekursive Datenstrukturen, Klassen selbst bauen)

→ werden eingeführt, wenn wir sie brauchen, oder:

→ Literatur

(z. B. Wikibooks: C-Programmierung,
Dokumentation zu Compiler und Bibliotheken)

2 Einführung in C

Sprachelemente weitgehend komplett

Es fehlen:

- Ergänzungen (z. B. ternärer Operator, **union**, **unsigned**, **volatile**)
- Bibliotheksfunktionen (z. B. **malloc()**)

→ werden eingeführt, wenn wir sie brauchen

- Konzepte (z. B. rekursive Datenstrukturen, Klassen selbst bauen)

→ werden eingeführt, wenn wir sie brauchen, oder:

→ Literatur

(z. B. Wikibooks: C-Programmierung,
Dokumentation zu Compiler und Bibliotheken)

- Praxiserfahrung

2 Einführung in C

Sprachelemente weitgehend komplett

Es fehlen:

- Ergänzungen (z. B. ternärer Operator, **union**, **unsigned**, **volatile**)
- Bibliotheksfunktionen (z. B. **malloc()**)

→ werden eingeführt, wenn wir sie brauchen

- Konzepte (z. B. rekursive Datenstrukturen, Klassen selbst bauen)

→ werden eingeführt, wenn wir sie brauchen, oder:

→ Literatur

(z. B. Wikibooks: C-Programmierung,
Dokumentation zu Compiler und Bibliotheken)

- Praxiserfahrung

→ Praktikum

2 Einführung in C

Sprachelemente weitgehend komplett

Es fehlen:

- Ergänzungen (z. B. ternärer Operator, **union**, **unsigned**, **volatile**)
- Bibliotheksfunktionen (z. B. **malloc()**)

→ werden eingeführt, wenn wir sie brauchen

- Konzepte (z. B. rekursive Datenstrukturen, Klassen selbst bauen)

→ werden eingeführt, wenn wir sie brauchen, oder:

→ Literatur

(z. B. Wikibooks: C-Programmierung,
Dokumentation zu Compiler und Bibliotheken)

- Praxiserfahrung

→ Praktikum: nur Einstieg

2 Einführung in C

Sprachelemente weitgehend komplett

Es fehlen:

- Ergänzungen (z. B. ternärer Operator, **union**, **unsigned**, **volatile**)
- Bibliotheksfunktionen (z. B. **malloc()**)

→ werden eingeführt, wenn wir sie brauchen

- Konzepte (z. B. rekursive Datenstrukturen, Klassen selbst bauen)

→ werden eingeführt, wenn wir sie brauchen, oder:

→ Literatur

(z. B. Wikibooks: C-Programmierung,
Dokumentation zu Compiler und Bibliotheken)

- Praxiserfahrung

→ Praktikum: nur Einstieg

→ selbständig arbeiten

3 Bibliotheken

3.1 Der Präprozessor

`#include`

3 Bibliotheken

3.1 Der Präprozessor

#include: Text einbinden

3 Bibliotheken

3.1 Der Präprozessor

#include: Text einbinden

- **#include** `<stdio.h>`: Standard-Verzeichnisse – Standard-Header

3 Bibliotheken

3.1 Der Präprozessor

#include: Text einbinden

- **#include** <stdio.h>: Standard-Verzeichnisse – Standard-Header
- **#include** "answer.h": auch aktuelles Verzeichnis – eigene Header

3 Bibliotheken

3.1 Der Präprozessor

#include: Text einbinden

- **#include** <stdio.h>: Standard-Verzeichnisse – Standard-Header
- **#include** "answer.h": auch aktuelles Verzeichnis – eigene Header

#define VIER 4: Text ersetzen lassen – Konstante definieren

3 Bibliotheken

3.1 Der Präprozessor

#include: Text einbinden

- **#include** <stdio.h>: Standard-Verzeichnisse – Standard-Header
- **#include** "answer.h": auch aktuelles Verzeichnis – eigene Header

#define VIER 4: Text ersetzen lassen – Konstante definieren

- Kein Semikolon!

3 Bibliotheken

3.1 Der Präprozessor

#include: Text einbinden

- **#include** <stdio.h>: Standard-Verzeichnisse – Standard-Header
- **#include** "answer.h": auch aktuelles Verzeichnis – eigene Header

#define VIER 4: Text ersetzen lassen – Konstante definieren

- Kein Semikolon!
- Berechnungen in Klammern setzen:
#define VIER (2 + 2)

3 Bibliotheken

3.1 Der Präprozessor

#include: Text einbinden

- **#include** <stdio.h>: Standard-Verzeichnisse – Standard-Header
- **#include** "answer.h": auch aktuelles Verzeichnis – eigene Header

#define VIER 4: Text ersetzen lassen – Konstante definieren

- Kein Semikolon!
- Berechnungen in Klammern setzen:
#define VIER (2 + 2)
- Konvention: Großbuchstaben

3 Bibliotheken

3.2 Bibliotheken einbinden

Inhalt der Header-Datei: externe Deklarationen

3 Bibliotheken

3.2 Bibliotheken einbinden

Inhalt der Header-Datei: externe Deklarationen

```
extern int answer (void);
```

3 Bibliotheken

3.2 Bibliotheken einbinden

Inhalt der Header-Datei: externe Deklarationen

```
extern int answer (void);
```

```
extern int printf (__const char *__restrict __format, ...);
```

3 Bibliotheken

3.2 Bibliotheken einbinden

Inhalt der Header-Datei: externe Deklarationen

```
extern int answer (void);
```

```
extern int printf (__const char *__restrict __format, ...);
```

Funktion wird „anderswo“ definiert

3 Bibliotheken

3.2 Bibliotheken einbinden

Inhalt der Header-Datei: externe Deklarationen

```
extern int answer (void);
```

```
extern int printf (__const char *__restrict __format, ...);
```

Funktion wird „anderswo“ definiert

- separater C-Quelltext: mit an `gcc` übergeben

3 Bibliotheken

3.2 Bibliotheken einbinden

Inhalt der Header-Datei: externe Deklarationen

```
extern int answer (void);
```

```
extern int printf (__const char *__restrict __format, ...);
```

Funktion wird „anderswo“ definiert

- separater C-Quelltext: mit an `gcc` übergeben
- vorcompilierte Bibliothek: `-lfoo`

3 Bibliotheken

3.2 Bibliotheken einbinden

Inhalt der Header-Datei: externe Deklarationen

```
extern int answer (void);
```

```
extern int printf (__const char *__restrict __format, ...);
```

Funktion wird „anderswo“ definiert

- separater C-Quelltext: mit an `gcc` übergeben
- vorcompilierte Bibliothek: `-lfoo`
= Datei `libfoo.a` in Standard-Verzeichnis