Hochschule Bochum
Bochum University
of Applied Sciences

Angewandte Informatik

Prof. Dr. Peter Gerwinski

15. November 2012

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 **char**s, also ein Zeiger auf **char**s, 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 **char**s, also ein Zeiger auf **char**s, 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

2.11 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]; 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())

$\longrightarrow$ 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())

$\longrightarrow$ 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