Hardwarenahe Software-Entwicklung

Hardwarenahe Software-Entwicklung

1 Wiederholung: Einführung in C

Hardwarenahe Software-Entwicklung

1 Wiederholung: Einführung in C

- Zeiger

Hardwarenahe Software-Entwicklung

1 Wiederholung: Einführung in C

- Zeiger

```
void set_value (int *a)
{
   *a = 137;
}
```

Hardwarenahe Software-Entwicklung

1 Wiederholung: Einführung in C

- Zeiger

```
void set_value (int *a)
{
   *a = 137;
}
```

```
int b;
set_value (&b);
```

Hardwarenahe Software-Entwicklung

1 Wiederholung: Einführung in C

- Zeiger

```
void set_value (int *a)
{
   *a = 137;
}
```

```
int b;
set_value (&b);
```

- Zeiger-Arithmetik

Hardwarenahe Software-Entwicklung

1 Wiederholung: Einführung in C

- Zeiger

```
void set_value (int *a)
{
   *a = 137;
}
```

```
int b;
set_value (&b);
```

- Zeiger-Arithmetik

```
char hello[] = "Hello";
char *p = hello;
while (*p)
   printf ("%c", *p++);
```

Hardwarenahe Software-Entwicklung

1 Wiederholung: Einführung in C

- Zeiger

```
void set_value (int *a)
{
   *a = 137;
}
```

```
int b;
set_value (&b);
```

- Zeiger-Arithmetik

```
char hello[] = "Hello";
char *p = hello;
while (*p)
   printf ("%c", *p++);
```

- Arrays

Hardwarenahe Software-Entwicklung

1 Wiederholung: Einführung in C

- Zeiger

```
void set_value (int *a)
{
   *a = 137;
}
```

```
int b;
set_value (&b);
```

- Zeiger-Arithmetik

```
char hello[] = "Hello";
char *p = hello;
while (*p)
   printf ("%c", *p++);
```

- Arrays

```
char hello[] = "Hello";
int i = 0;
while (hello[i])
   printf ("%c", hello[i++]);
```

Hardwarenahe Software-Entwicklung

1 Wiederholung: Einführung in C

- Zeiger

```
void set_value (int *a)
{
   *a = 137;
}
```

```
int b;
set_value (&b);
```

- Zeiger-Arithmetik

```
char hello[] = "Hello";
char *p = hello;
while (*p)
   printf ("%c", *p++);
```

- Arrays

```
char hello[] = "Hello";
int i = 0;
while (hello[i])
   printf ("%c", hello[i++]);
```

p[i] ist dasselbe wie *(p + i)

Hardwarenahe Software-Entwicklung

1 Wiederholung: Einführung in C

- Zeiger

```
void set_value (int *a)
{
   *a = 137;
}
```

```
int b;
set_value (&b);
```

- Zeiger-Arithmetik

```
char hello[] = "Hello";
char *p = hello;
while (*p)
   printf ("%c", *p++);
```

- Arrays

```
char hello[] = "Hello";
int i = 0;
while (hello[i])
   printf ("%c", hello[i++]);
```

p[i] ist dasselbe wie *(p + i)

- String = Array von **char**s

Hardwarenahe Software-Entwicklung

1 Wiederholung: Einführung in C

- Zeiger

```
void set_value (int *a)
{
   *a = 137;
}
```

```
int b;
set_value (&b);
```

- Zeiger-Arithmetik

```
char hello[] = "Hello";
char *p = hello;
while (*p)
   printf ("%c", *p++);
```

- Arrays

```
char hello[] = "Hello";
int i = 0;
while (hello[i])
   printf ("%c", hello[i++]);
```

p[i] ist dasselbe wie *(p + i)

- String = Array von **char**s (ganze Zahlen)

Hardwarenahe Software-Entwicklung

1 Wiederholung: Einführung in C

- Zeiger

```
void set_value (int *a)
{
   *a = 137;
}
```

```
int b;
set_value (&b);
```

- Zeiger-Arithmetik

```
char hello[] = "Hello";
char *p = hello;
while (*p)
   printf ("%c", *p++);
```

- Arrays

```
char hello[] = "Hello";
int i = 0;
while (hello[i])
   printf ("%c", hello[i++]);
```

p[i] ist dasselbe wie *(p + i)

- String = Array von **char**s (ganze Zahlen)

letzter Eintrag: 0

1 Wiederholung: Einführung in C

- String = Array von **char**s (ganze Zahlen) letzter Eintrag: 0

- Parameter des Hauptprogramms

1 Wiederholung: Einführung in C

- String = Array von **char**s (ganze Zahlen) letzter Eintrag: 0

- Parameter des Hauptprogramms

```
int main (int argc,
          char **argv)
{ ... }
```

1 Wiederholung: Einführung in C

- String = Array von **char**s (ganze Zahlen)

  letzter Eintrag: 0

- Parameter des Hauptprogramms

```
int main (int argc,
          char **argv)
{ ... }
```

  ← Anzahl der Parameter

1 Wiederholung: Einführung in C

- String = Array von **char**s (ganze Zahlen) letzter Eintrag: 0

- Parameter des Hauptprogramms

```
int main (int argc,          ← Anzahl der Parameter
          char **argv)       ← Array von Arrays
{ ... }
```

1 Wiederholung: Einführung in C

- String = Array von **char**s (ganze Zahlen) letzter Eintrag: 0

- Parameter des Hauptprogramms

```
int main (int argc,          <- Anzahl der Parameter
          char **argv)       <- Array von Arrays
{ ... }                        letzter Eintrag: NULL
```

1 Wiederholung: Einführung in C

- String = Array von **char**s (ganze Zahlen)
  letzter Eintrag: 0

- Parameter des Hauptprogramms

```
int main (int argc,
          char **argv)
{ ... }
```

  ← Anzahl der Parameter
  ← Array von Arrays
  letzter Eintrag: NULL
  = Zeiger auf „nichts“

1 Wiederholung: Einführung in C

- String = Array von **char**s (ganze Zahlen) — letzter Eintrag: 0

- Parameter des Hauptprogramms

```
int main (int argc,
          char **argv)
{ ... }
```

← Anzahl der Parameter
← Array von Arrays
letzter Eintrag: NULL
= Zeiger auf „nichts“

- Strukturen

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

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

```
date today;
get_date (&today);
printf ("%d.%d.%d\n",
   today.day, today.month,
   today.year);
```

1 Wiederholung: Einführung in C

- String = Array von **char**s (ganze Zahlen)

  letzter Eintrag: 0

- Parameter des Hauptprogramms

```
int main (int argc,          ← Anzahl der Parameter
          char **argv)       ← Array von Arrays
{ ... }                        letzter Eintrag: NULL
                               = Zeiger auf „nichts“
```

- Strukturen

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

Komponenten

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

```
date today;
get_date (&today);
printf ("%d.%d.%d\n",
   today.day, today.month,
   today.year);
```

1 Wiederholung: Einführung in C

- String = Array von **char**s (ganze Zahlen) — letzter Eintrag: 0

- Parameter des Hauptprogramms

```
int main (int argc,          ← Anzahl der Parameter
          char **argv)       ← Array von Arrays
{ ... }                        letzter Eintrag: NULL
                               = Zeiger auf „nichts“
```

- Strukturen

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

Komponenten

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

```
date today;
get_date (&today);
printf ("%d.%d.%d\n",
   today.day, today.month,
   today.year);
```

Zugriff auf Komponenten

1 Wiederholung: Einführung in C

- String = Array von **char**s (ganze Zahlen)

  letzter Eintrag: 0

- Parameter des Hauptprogramms

```
int main (int argc,
          char **argv)
{ ... }
```

  ← Anzahl der Parameter

  ← Array von Arrays
  letzter Eintrag: NULL
  = Zeiger auf „nichts“

- Strukturen

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

Komponenten

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

dasselbe wie (*d).year

```
date today;
get_date (&today);
printf ("%d.%d.%d\n",
   today.day, today.month,
   today.year);
```

Zugriff auf Komponenten

1 Einführung in C

Sprachelemente weitgehend komplett

1 Einführung in C

Sprachelemente weitgehend komplett

Es fehlen:

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

1 Einführung in C

Sprachelemente weitgehend komplett

Es fehlen:

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

1 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

1 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)

1 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)

⟶ Literatur

1 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)

⟶ Literatur

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

1 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)

⟶ Literatur

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

- Praxiserfahrung

1 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)

⟶ Literatur

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

- Praxiserfahrung

⟶ Praktikum

1 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)

⟶ Literatur

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

- Praxiserfahrung

⟶ Praktikum: nur Einstieg

1 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)

⟶ Literatur

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

- Praxiserfahrung

⟶ Praktikum: nur Einstieg

⟶ selbständig arbeiten

2 Bibliotheken

2.1 Der Präprozessor

#include

2 Bibliotheken

2.1 Der Präprozessor

#include: Text einbinden

2 Bibliotheken

2.1 Der Präprozessor

#include: Text einbinden

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

2 Bibliotheken

2.1 Der Präprozessor

#include: Text einbinden

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

2 Bibliotheken

2.1 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

2 Bibliotheken

2.1 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!

2 Bibliotheken

2.1 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)

2 Bibliotheken

2.1 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

2 Bibliotheken

2.2 Bibliotheken einbinden

Inhalt der Header-Datei: externe Deklarationen

2 Bibliotheken

2.2 Bibliotheken einbinden

Inhalt der Header-Datei: externe Deklarationen

extern int answer (**void**);

2 Bibliotheken

2.2 Bibliotheken einbinden

Inhalt der Header-Datei: externe Deklarationen

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

2 Bibliotheken

2.2 Bibliotheken einbinden

Inhalt der Header-Datei: externe Deklarationen

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

Funktion wird „anderswo“ definiert

2 Bibliotheken

2.2 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

2 Bibliotheken

2.2 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`

2 Bibliotheken

2.2 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

2.3 Bibliothek verwenden (Beispiel: OpenGL)

- Include-Dateien:

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
```

2.3 Bibliothek verwenden (Beispiel: OpenGL)

- Include-Dateien:

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
```

- Compiler-Aufruf:

```
gcc -Wall -O cube.c -lGL -lGLU -lglut -o cube
```

2.3 Bibliothek verwenden (Beispiel: OpenGL)

- Include-Dateien:

  ```
  #include <GL/gl.h>
  #include <GL/glu.h>
  #include <GL/glut.h>
  ```

- Compiler-Aufruf:

  ```
  gcc -Wall -O cube.c -lGL -lGLU -lglut -o cube
  ```

- Funktionen aufrufen: glutInit (&argc, argv);

2.3 Bibliothek verwenden (Beispiel: OpenGL)

- Include-Dateien:

  ```
  #include <GL/gl.h>
  #include <GL/glu.h>
  #include <GL/glut.h>
  ```

- Compiler-Aufruf:

  ```
  gcc -Wall -O cube.c -lGL -lGLU -lglut -o cube
  ```

- Funktionen aufrufen: glutInit (&argc, argv);
- Konstante: GLUT_RGBA

2.3 Bibliothek verwenden (Beispiel: OpenGL)

- Include-Dateien:

  ```
  #include <GL/gl.h>
  #include <GL/glu.h>
  #include <GL/glut.h>
  ```

- Compiler-Aufruf:

  ```
  gcc -Wall -O cube.c -lGL -lGLU -lglut -o cube
  ```

- Funktionen aufrufen: glutInit (&argc, argv);
- Konstante: GLUT_RGBA
- Datentypen: GLfloat

2.3 Bibliothek verwenden (Beispiel: OpenGL)

- Include-Dateien:

  ```
  #include <GL/gl.h>
  #include <GL/glu.h>
  #include <GL/glut.h>
  ```

- Compiler-Aufruf:

  ```
  gcc -Wall -O cube.c -lGL -lGLU -lglut -o cube
  ```

- Funktionen aufrufen: glutInit (&argc, argv);
- Konstante: GLUT_RGBA
- Datentypen: GLfloat
- Array übergeben:

  ```
  GLfloat light0_position[] = {1.0, 2.0, −2.0, 1.0};
  glLightfv (GL_LIGHT0, GL_POSITION, light0_position);
  ```

2.3 Bibliothek verwenden (Beispiel: OpenGL)

- Include-Dateien:

  ```
  #include <GL/gl.h>
  #include <GL/glu.h>
  #include <GL/glut.h>
  ```

- Compiler-Aufruf:

  ```
  gcc -Wall -O cube.c -lGL -lGLU -lglut -o cube
  ```

- Funktionen aufrufen: glutInit (&argc, argv);
- Konstante: GLUT_RGBA
- Datentypen: GLfloat
- Array übergeben:

  ```
  GLfloat light0_position[] = {1.0, 2.0, −2.0, 1.0};
  glLightfv (GL_LIGHT0, GL_POSITION, light0_position);
  ```

- Funktion übergeben (Callbacks):

  ```
  void draw_cube (void)
  { ... }

  glutDisplayFunc (draw_cube);
  ```

2.3 Bibliothek verwenden (Beispiel: OpenGL)

Aufgabe:

Für welche elementaren geometrischen Körper
stellt die GLUT-Bibliothek Zeichenroutinen zur Verfügung?

2.3 Bibliothek verwenden (Beispiel: OpenGL)

Aufgabe:

Für welche elementaren geometrischen Körper
stellt die GLUT-Bibliothek Zeichenroutinen zur Verfügung?

Lösung:

glutSolidCube (GLdouble size);

2.3 Bibliothek verwenden (Beispiel: OpenGL)

Aufgabe:

Für welche elementaren geometrischen Körper
stellt die GLUT-Bibliothek Zeichenroutinen zur Verfügung?

Lösung:

```
glutSolidCube (GLdouble size);
glutWireCube (GLdouble size);
```

2.3 Bibliothek verwenden (Beispiel: OpenGL)

Aufgabe:

Für welche elementaren geometrischen Körper
stellt die GLUT-Bibliothek Zeichenroutinen zur Verfügung?

Lösung:

glutSolidCube (GLdouble size);
glutWireCube (GLdouble size);
glutSolidSphere (GLdouble radius, GLint slices, GLint stacks); ... Wire ...

2.3 Bibliothek verwenden (Beispiel: OpenGL)

Aufgabe:

Für welche elementaren geometrischen Körper
stellt die GLUT-Bibliothek Zeichenroutinen zur Verfügung?

Lösung:

```
glutSolidCube (GLdouble size);
glutWireCube (GLdouble size);
glutSolidSphere (GLdouble radius, GLint slices, GLint stacks); ... Wire ...
glutSolidCone (GLdouble base, GLdouble height, GLint slices, GLint stacks);
glutSolidTorus (GLdouble innerRadius, GLdouble outerRadius,
                GLint sides, GLint rings);
glutSolidDodecahedron (void);
glutSolidOctahedron (void);
glutSolidTetrahedron (void);
glutSolidIcosahedron (void);
```

2.3 Bibliothek verwenden (Beispiel: OpenGL)

Aufgabe:

Für welche elementaren geometrischen Körper
stellt die GLUT-Bibliothek Zeichenroutinen zur Verfügung?

Lösung:

```
glutSolidCube (GLdouble size);
glutWireCube (GLdouble size);
glutSolidSphere (GLdouble radius, GLint slices, GLint stacks); ... Wire ...
glutSolidCone (GLdouble base, GLdouble height, GLint slices, GLint stacks);
glutSolidTorus (GLdouble innerRadius, GLdouble outerRadius,
                GLint sides, GLint rings);
glutSolidDodecahedron (void);
glutSolidOctahedron (void);
glutSolidTetrahedron (void);
glutSolidIcosahedron (void);
glutSolidTeapot (GLdouble size);
```

2.3 Bibliothek verwenden (Beispiel: OpenGL)

Aufgabe:

Für welche elementaren geometrischen Körper
stellt die GLUT-Bibliothek Zeichenroutinen zur Verfügung?

Lösung:

```
glutSolidCube (GLdouble size);
glutWireCube (GLdouble size);
glutSolidSphere (GLdouble radius, GLint slices, GLint stacks); ... Wire ...
glutSolidCone (GLdouble base, GLdouble height, GLint slices, GLint stacks);
glutSolidTorus (GLdouble innerRadius, GLdouble outerRadius,
                GLint sides, GLint rings);
glutSolidDodecahedron (void);
glutSolidOctahedron (void);
glutSolidTetrahedron (void);
glutSolidIcosahedron (void);
glutSolidTeapot (GLdouble size);
glutSolidRhombicDodecahedron (void);
glutSolidSierpinskiSponge (int num_levels, GLdouble offset[3], GLdouble scale);
glutSolidCylinder (GLdouble radius, GLdouble height, GLint slices, GLint stacks);
```

Praktikumsaufgabe

Schreiben Sie 3d-„Basketball“-Programm.

Ein schief abgeworfener Ball fliegt
entlang einer Wurfparabel
durch einen horizontalen Ring.

Hinweis:

$$
\begin{aligned}
x(t) &= x_0 + v_{0x} \cdot t \\
y(t) &= y_0 + v_{0y} \cdot t - \tfrac{1}{2} g t^2 \\
g &= 9.81 \frac{m}{s^2}
\end{aligned}
$$