

Systemnahe Programmierung in C

5 Sprachüberblick

J. Kleinöder, D. Lohmann, V. Sieh

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Erlangen-Nürnberg

Sommersemester 2024

<http://sys.cs.fau.de/lehre/ss24>



Struktur eines C-Programms – allgemein

```
1 // include files
2 #include ...
3
4 // global variables
5 ... variable1 = ...
6
7 // subfunction 1
8 ... subfunction_1(...) {
9     // local variables
10    ... variable1 = ...
11    // statements
12    ...
13 }
14 // subfunction n
15 ... subfunction_n(...) {
16
17     ...
18
19 }
20
21 // main function
22 ... main(...) {
23
24     ...
25
26 }
```

- Ein C-Programm besteht (üblicherweise) aus
 - Menge von globalen Variablen
 - Menge von (Sub-)Funktionen
 - Menge von lokalen Variablen
 - Menge von Anweisungen
 - Der Funktion `main()`, in der die Ausführung beginnt



Struktur eines C-Programms – am Beispiel

```
1 // include files
2 #include <led.h>
3
4 // global variables
5 LED nextLED = RED0;
6
7 // subfunction 1
8 LED lightLED(void) {
9     if (nextLED <= BLUE1) {
10         sb_led_on(nextLED++);
11     }
12     return nextLED;
13 }
14 // subfunction 2
15 void wait(void) {
16     volatile unsigned int i;
17     for (i = 0; i < 0xffff; i++)
18         ;
19 }
20
21 // main function
22 void main(void) {
23     while (lightLED() < 8) {
24         wait();
25     }
26 }
```

■ Ein C-Programm besteht (üblicherweise) aus

- Menge von **globalen Variablen** nextLED, Zeile 5
- Menge von **(Sub-)Funktionen** wait(), Zeile 15
 - Menge von **lokalen Variablen** i, Zeile 16
 - Menge von **Anweisungen** for-Schleife, Zeile 17
- Der Funktion **main()**, in der die Ausführung beginnt



```
1 // include files
2 #include <led.h>
3
4 // global variables
5 LED nextLED = RED0;
6
7 // subfunction 1
8 LED lightLED(void) {
9     if (nextLED <= BLUE1) {
10         sb_led_on(nextLED++);
11     }
12     return nextLED;
13 }
14 // subfunction 2
15 void wait(void) {
16     volatile unsigned int i;
17     for (i = 0; i < 0xffff; i++)
18         ;
19 }
20
21 // main function
22 void main() {
23     while (lightLED() < 8) {
24         wait();
25     }
26 }
```

- Vom Entwickler vergebener **Name** für ein Element des Programms
 - Element: Typ, Variable, Konstante, Funktion, Sprungmarke
 - Aufbau: [A-Z, a-z, _] [A-Z, a-z, 0-9, _]*
 - Buchstabe gefolgt von Buchstaben, Ziffern und Unterstrichen
 - **Unterstrich als erstes Zeichen** möglich, aber reserviert für Compilerhersteller
 - Ein Bezeichner muss vor Gebrauch **deklariert** werden



```
1 // include files
2 #include <led.h>
3
4 // global variables
5 LED nextLED = RED0;
6
7 // subfunction 1
8 LED lightLED(void) {
9     if (nextLED <= BLUE1) {
10         sb_led_on(nextLED++);
11     }
12     return nextLED;
13 }
14 // subfunction 2
15 void wait(void) {
16     volatile unsigned int i;
17     for (i = 0; i < 0xffff; i++)
18         ;
19 }
20
21 // main function
22 void main(void) {
23     while (lightLED() < 8) {
24         wait();
25     }
26 }
```

■ Reservierte Wörter der Sprache

(↪ dürfen nicht als Bezeichner verwendet werden)

- Eingebaute (*primitive*) Datentypen unsigned int, void
- Typmodifizierer volatile
- Kontrollstrukturen for, while
- Elementaranweisungen return



```
1 // include files
2 #include <led.h>
3
4 // global variables
5 LED nextLED = RED0;
6
7 // subfunction 1
8 LED lightLED(void) {
9     if (nextLED <= BLUE1) {
10         sb_led_on(nextLED++);
11     }
12     return nextLED;
13 }
14 // subfunction 2
15 void wait(void) {
16     volatile unsigned int i;
17     for (i = 0; i < 0xffff; i++)
18         ;
19 }
20
21 // main function
22 void main() {
23     while (lightLED() < 8) {
24         wait();
25     }
26 }
```

■ (Darstellung von) Konstanten im Quelltext

- Für jeden primitiven Datentyp gibt es eine oder mehrere Literalformen
 - Bei Integertypen: dezimal (Basis 10: 65535), hexadezimal (Basis 16, führendes 0x: 0xffff), oktal (Basis 8, führende 0: 0177777)
- Der Programmierer kann jeweils die am besten geeignete Form wählen
 - 0xffff ist handlicher als 65535, um den Maximalwert einer vorzeichenlosen 16-Bit-Ganzzahl darzustellen



```
1 // include files
2 #include <led.h>
3
4 // global variables
5 LED nextLED = RED0;
6
7 // subfunction 1
8 LED lightLED(void) {
9     if (nextLED <= BLUE1) {
10         sb_led_on(nextLED++);
11     }
12     return nextLED;
13 }
14 // subfunction 2
15 void wait(void) {
16     volatile unsigned int i;
17     for (i = 0; i < 0xffff; i++)
18         ;
19 }
20
21 // main function
22 void main() {
23     while (lightLED() < 8) {
24         wait();
25     }
26 }
```

- Beschreiben den eigentlichen **Ablauf** des Programms
- Werden hierarchisch komponiert aus drei Grundformen
 - Einzelanweisung – **Ausdruck** gefolgt von `;`;
 - einzelnes Semikolon \mapsto leere Anweisung
 - **Block** – Sequenz von Anweisungen, geklammert durch `{...}`
 - **Kontrollstruktur**, gefolgt von Anweisung



```
1 // include files
2 #include <led.h>
3
4 // global variables
5 LED nextLED = RED0;
6
7 // subfunction 1
8 LED lightLED(void) {
9     if (nextLED <= BLUE1) {
10         sb_led_on(nextLED++);
11     }
12     return nextLED;
13 }
14 // subfunction 2
15 void wait(void) {
16     volatile unsigned int i;
17     for (i = 0; i < 0xffff; i++)
18         ;
19 }
20
21 // main function
22 void main() {
23     while (lightLED() < 8) {
24         wait();
25     }
26 }
```

- Beschreiben den eigentlichen **Ablauf** des Programms
- Werden hierarchisch komponiert aus drei Grundformen
 - Einzelanweisung – **Ausdruck** gefolgt von **;**
 - einzelnes Semikolon ↪ leere Anweisung
 - **Block** – Sequenz von Anweisungen, geklammert durch **{...}**
 - **Kontrollstruktur**, gefolgt von Anweisung




```
1 // include files
2 #include <led.h>
3
4 // global variables
5 LED nextLED = RED0;
6
7 // subfunction 1
8 LED lightLED(void) {
9     if (nextLED <= BLUE1) {
10         sb_led_on(nextLED++);
11     }
12     return nextLED;
13 }
14 // subfunction 2
15 void wait(void) {
16     volatile unsigned int i;
17     for (i = 0; i < 0xffff; i++)
18         ;
19 }
20
21 // main function
22 void main() {
23     while (lightLED() < 8) {
24         wait();
25     }
26 }
```

- Beschreiben den eigentlichen **Ablauf** des Programms
- Werden hierarchisch komponiert aus drei Grundformen
 - Einzelanweisung – **Ausdruck** gefolgt von `;`;
 - einzelnes Semikolon \mapsto leere Anweisung
 - **Block** – Sequenz von Anweisungen, geklammert durch `{...}`
 - **Kontrollstruktur**, gefolgt von Anweisung



```
1 // include files
2 #include <led.h>
3
4 // global variables
5 LED nextLED = RED0;
6
7 // subfunction 1
8 LED lightLED(void) {
9     if (nextLED <= BLUE1) {
10         sb_led_on(nextLED++);
11     }
12     return nextLED;
13 }
14 // subfunction 2
15 void wait(void) {
16     volatile unsigned int i;
17     for (i = 0; i < 0xffff; i++)
18         ;
19 }
20
21 // main function
22 void main() {
23     while (lightLED() < 8) {
24         wait();
25     }
26 }
```

- Beschreiben den eigentlichen **Ablauf** des Programms
- Werden hierarchisch komponiert aus drei Grundformen
 - Einzelanweisung – **Ausdruck** gefolgt von `;`;
 - einzelnes Semikolon \mapsto leere Anweisung
 - **Block** – Sequenz von Anweisungen, geklammert durch `{...}`
 - **Kontrollstruktur**, gefolgt von Anweisung



```
1 // include files
2 #include <led.h>
3
4 // global variables
5 LED nextLED = RED0;
6
7 // subfunction 1
8 LED lightLED(void) {
9     if (nextLED <= BLUE1) {
10         sb_led_on(nextLED++);
11     }
12     return nextLED;
13 }
14 // subfunction 2
15 void wait(void) {
16     volatile unsigned int i;
17     for (i = 0; i < 0xffff; i++)
18         ;
19 }
20
21 // main function
22 void main() {
23     while (lightLED() < 8) {
24         wait();
25     }
26 }
```

■ Gültige Kombination von Operatoren, Literalen und Bezeichnern

- „Gültig“ im Sinne von Syntax und Typsystem
- Vorrangregeln für Operatoren legen die Reihenfolge fest, ↪ 7-14
in der Ausdrücke abgearbeitet werden
 - Auswertungsreihenfolge kann mit Klammern () explizit bestimmt werden
 - Der Compiler darf Teilausdrücke in möglichst effizienter Folge auswerten

