

System-Level Programming

5 Language Overview

J. Kleinöder, D. Lohmann, V. Sieh, P. Wägemann

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Erlangen-Nürnberg

Summer Term 2024

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



Structure of a C Program – General

```
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 }
```

- A C-program (usually) consists of
 - a set of **global variables**
 - a set of **(sub-)functions**
 - a set of **local variables**
 - a set of **instructions**
 - the function **main()**, which is the entry point for any execution



Structure of a C Program – an Example

```
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 }
```

■ A C-program (usually) consists of

- a set of **global variables** nextLED, line 5
- a set of **(sub-)functions** wait(), line 15
 - a set of **local variables** i, line 16
 - a set of **instructions** for-loop, line 17
- the function **main()**, which is the entry point for any execution



```
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 }
```

- **Names** given by the developer for certain elements of the program
 - element: type, variable, constant, function, jump mark
 - structure: [A-Z, a-z, _] [A-Z, a-z, 0-9, _]*
 - one letter, followed by a combination of letters, numbers and underscores
 - **underscore** can be used **as a first symbol**, however, this is usually reserved for compiler manufacturers
 - every identifier has to be **declared** prior to being used



```

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 }

```

■ Reserved words of the language
(~ shall never be used as an identifier)

- embedded (*primitive*) types unsigned int, void
- type modifiers volatile
- control structures for, while
- elementary instructions 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 }
```

■ (Expression of) constants in the code

- For every primitive data type, there is at least one literal form.
 - for integers: decimal (base 10: 65535), hexadecimal (base 16, leading 0x: 0xffff), octal (base 8, leading 0: 0177777)
- The programmer can then choose the best suited form.
 - 0xffff is more handy than 65535 to represent the maximal value of a 16-bit integer



```
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 }
```

- Outline the actual **procedure** of the program
- They are hierarchically made up from three basic forms
 - single instruction – **expression** followed by **;**
 - single semicolon \mapsto empty instruction
 - **block** – sequence of instructions, wrapped in **{...}**
 - **control structures**, followed by instructions



```
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 }
```

- Outline the actual **procedure** of the program
- They are hierarchically made up from three basic forms
 - single instruction – **expression** followed by **;**
 - single semicolon \mapsto empty instruction
 - **block** – sequence of instructions, wrapped in **{...}**
 - **control structures**, followed by instructions




```
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 }
```

- Outline the actual **procedure** of the program
- They are hierarchically made up from three basic forms
 - single instruction – **expression** followed by **;**
 - single semicolon \mapsto empty instruction
 - **block** – sequence of instructions, wrapped in **{...}**
 - **control structures**, followed by instructions



```
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 }
```

- Outline the actual **procedure** of the program
- They are hierarchically made up from three basic forms
 - single instruction – **expression** followed by `;`
 - single semicolon \mapsto empty instruction
 - **block** – sequence of instructions, wrapped in `{...}`
 - **control structures**, followed by instructions



```
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 }
```

■ Valid combination of operators, literals, and identifiers

- “valid” in the sense of syntax and types
- priority rules for operators determine the order, in which the expressions get handled
 - order of execution can be explicitly forced with the help of brackets ()
 - the compiler is allowed to evaluate partial expressions in the most efficient order

↪ 7-23

