System-Level Programming

14 Composite Data Types

Peter Wägemann

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Erlangen-Nürnberg (FAU)

Summer Term 2025

http://sys.cs.fau.de/lehre/ss25

Structs: Motivation

- If variables "somehow" belong together,
  - intuitive approach to structure together
  - problem-specific abstraction ↪ 4–1
  - separation of concerns ↪ 12–4
- C makes this possible with composite types: structs

```
// Structure declaration
struct Student {
  char  lastname[64];
  char  firstname[64];
  long  matnum;
  int   passed;
};

// Variable definition
struct Student stud;

// Pointer definition
struct Student *pstud;
```

A structure type combines a set of data with a common type.

The elements are placed subsequently in memory.

Structs: Motivation

- If variables “somehow” belong together,
  - intuitive approach to structure together
  - problem-specific abstraction ↪ 4–1
  - separation of concerns ↪ 12–4
- C makes this possible with composite types: structs

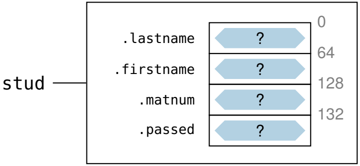
```c
// Structure declaration
struct Student {
  char  lastname[64];
  char  firstname[64];
  long  matnum;
  int   passed;
};

// Variable definition
struct Student stud;

// Pointer definition
struct Student *pstud;
```

A structure type combines a set of data with a common type.

The elements are placed subsequently in memory.



Structs: Variable Definitions and Initialization

- Similar to an array, a structure variable is initialized during definition ↪ 13–8

```
struct Student {
  char  lastname[64];
  char  firstname[64];
  long  matnum;
  int   passed;
};
```

```
struct Student stud = { "Meier", "Hans",
                        4711, 0 };
```

The initializers are only assigned by order, not by their identifier. ⇝ potential source of errors when changing code!

Structs: Variable Definitions and Initialization

- Similar to an array, a structure variable is initialized during definition ↪ 13–8

```
struct Student {
  char   lastname[64];
  char   firstname[64];
  long   matnum;
  int    passed;
};
```

```
struct Student stud = { "Meier", "Hans",
                        4711, 0 };
```

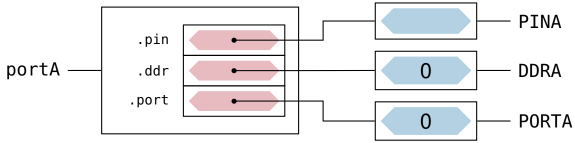
The initializers are only assigned by order, not by their identifier. ⇝ potential source of errors when changing code!

- In analogy to the definition of `enum` types, the use is simplified with the keyword `typedef` ↪ 6–8

```
typedef struct {
  volatile uint8_t *pin;
  volatile uint8_t *ddr;
  volatile uint8_t *port;
} port_t;
```

```
port_t portA = { &PINA, &DDRA, &PORTA };
port_t portD = { &PIND, &DDRD, &PORTD };
```

Structs: Access to Elements



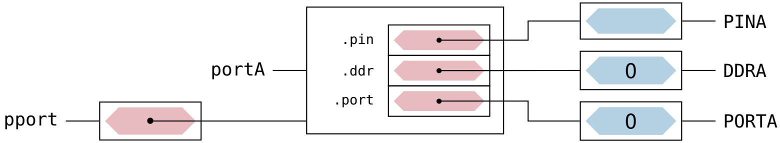
- The elements of a structure is accessed with the `.`-operator [≈Python]

```
port_t portA = { &PINA, &DDRA, &PORTA };

*portA.port = 0;    // clear all pins
*portA.ddr  = 0xff; // set all to output
```

Note: `.` has higher precedence than `*`

Structs: Access to Elements

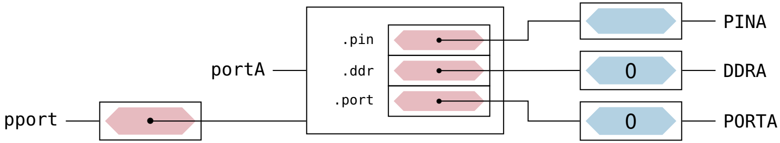


- Parentheses are required when working with pointers to structures

```
port_t *pport = &portA; // p --> portA

*(*pport).port = 0;     // clear all pins
*(*pport).ddr  = 0xff;  // set all to output
```

Structs: Access to Elements



- Parentheses are required when working with pointers to structures

```
port_t *pport = &portA; // p --> portA

*(*pport).port = 0;     // clear all pins
*(*pport).ddr  = 0xff;  // set all to output
```

- The **->** operator simplifies this access: `s->m` ≡ `(*s).m`

```
port_t *pport = &portA; // p --> portA

*pport->port = 0;       // clear all pins
*pport->ddr  = 0xff;    // set all to output
```

-> **also** has a higher precedence than *

Structs as Function Arguments

- In contrast to arrays, structs are passed *by value*

```
void initPort(port_t p) {
  *p.port = 0;         // clear all pins
  *p.ddr  = 0xff;      // set all to output
  p.port  = &PORTD;    // no effect, p is local variable
}

void main(void) { initPort(portA); ··· }
```

Structs as Function Arguments

- In contrast to arrays, structs are passed *by value*

```
void initPort(port_t p) {
  *p.port = 0;        // clear all pins
  *p.ddr  = 0xff;     // set all to output
  p.port  = &PORTD;   // no effect, p is local variable
}

void main(void) { initPort(portA); ··· }
```

- This is **highly inefficient** when working with larger structures
  - e. g., `Student` (↪ 14–1): 134 byte have to be allocated and copied with each function call
  - better solution: pass a pointer to the constant structure

```
void initPort(const port_t *p){
  *p->port = 0;        // clear all pins
  *p->ddr  = 0xff;     // set all to output
  // p->port  = &PORTD;   compile-time error, *p is const!
}

void main(void) { initPort(&portA); ··· }
```

Bit-Structures: Bit Arrays

- Single structure elements can be (un)set with granularity of bits
  - the compiler combines bit fields as suitable integer types
  - useful for accessing a range of bits of a register
- Example
  - `EICRA` **External Interrupt Control Register A**
    Controls triggers for external interrupt sources `INT0` und `INT1`. [1]

7	6	5	4	3	2	1	0
				ISC11	ISC10	ISC01	ISC00
				R/W	R/W	R/W	R/W

```
typedef struct {
  uint8_t ISC0     : 2;     // bit 0-1: interrupt sense control INT0
  uint8_t ISC1     : 2;     // bit 2-3: interrupt sense control INT1
  uint8_t reserved : 4;     // bit 4-7: reserved for future use
} EICRA_t;
```

References

[1] *ATmega328PB 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash*. Atmel Corporation. Okt. 2015. URL: `https://sys.cs.fau.de/extern/lehre/ss25/spic/uebung/spicboard/Atmel-42397-8-bit-AVR-Microcontroller-ATmega328PB_Datasheet.pdf`.