Structs

A presentation by

Christoph B., Leander D., Jakob G. and

Mario N.

Overview

- What is a struct?
- Operators
- Initialization
- Acessing structs
- Size of structs

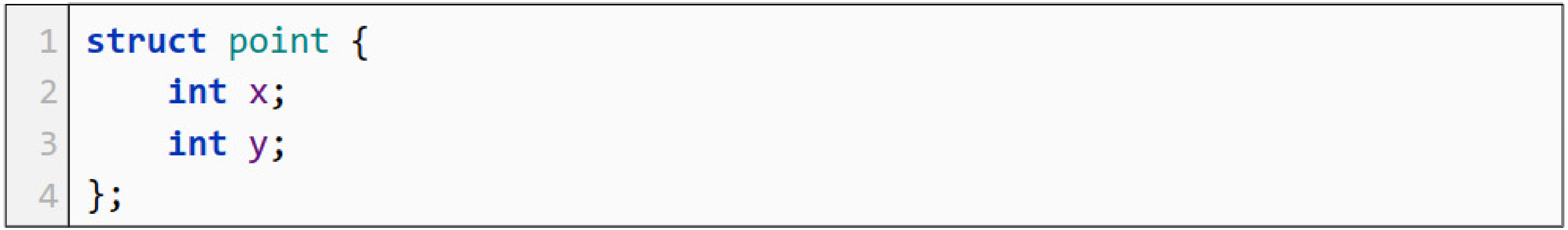
- Typedef
- Functions with structs
- Addition: Bitfields and Unions
- Task one: Elastic collisions
- Task two: Rocket equation

What is a Struct ?

- A structure is a user-defined composite data type.

- It consists of components (members), each having a type and a name.

- At least one member.



```c
struct point {
    int x;
    int y;
};
```

Operators

- Assignment operator (=)
- Address operator (&)
- Dot operator (.) and arrow operator (->)
- sizeof operator

Initialization

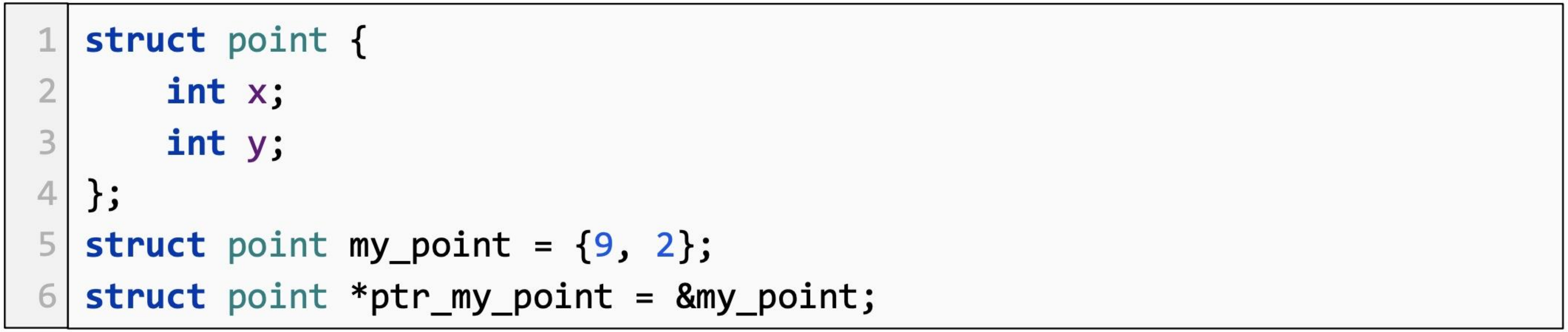
- Structure members can be initialized during definition

- Members are typically initialized in the order of declaration.

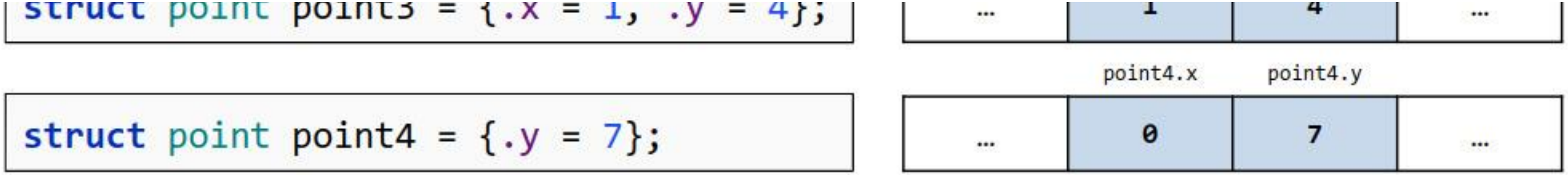
- Alternatively, members can be initialized by name (designated initializers).

- Uninitialized members are automatically set to the default zero value of their data type.

Example: Initialization



```
struct point {
    int x;
    int y;
};
struct point my_point = {9, 2};
struct point *ptr_my_point = &my_point;
```



```
struct point point3 = {.x = 1, .y = 4};
```

...	1	4	...

```
struct point point4 = {.y = 7};
```

	point4.x	point4.y	
...	0	7	...

Accessing Structs

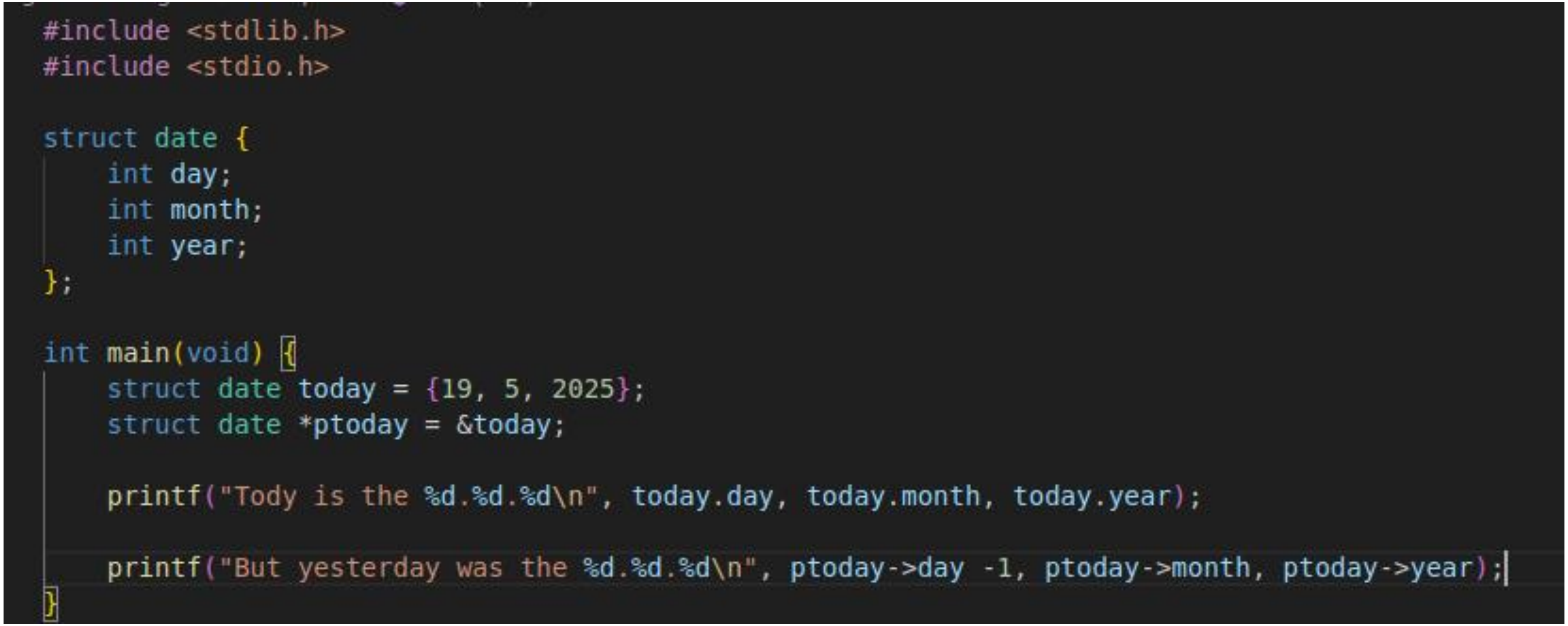
(.) And (->) Operator

- Binary operators

- Point-Operator is used for a struct – type

- Arrow-Operator is used for Pointers to struct types

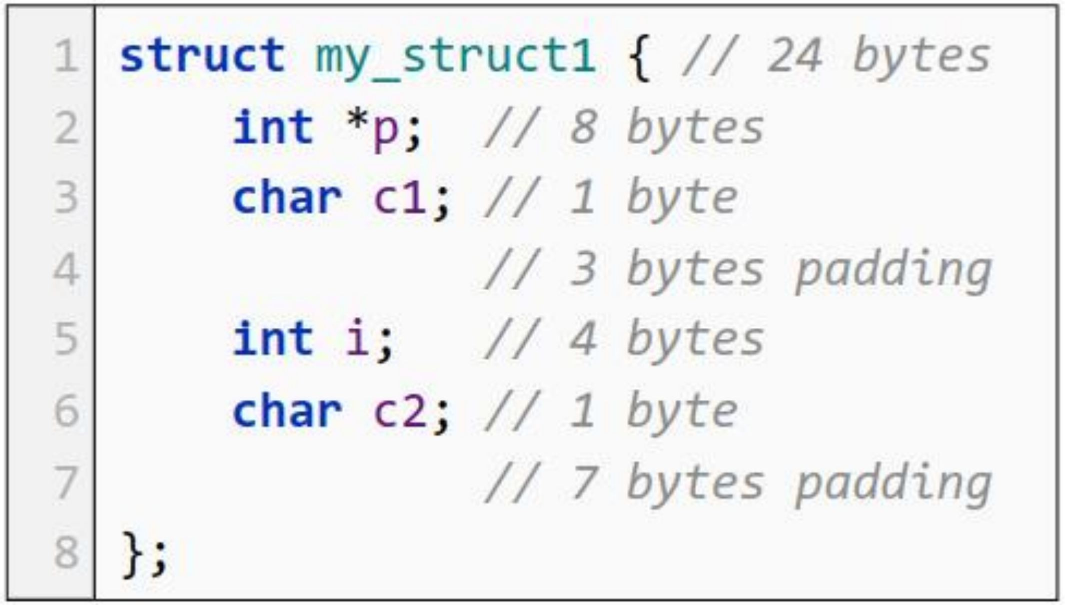
Example: Accessing structs



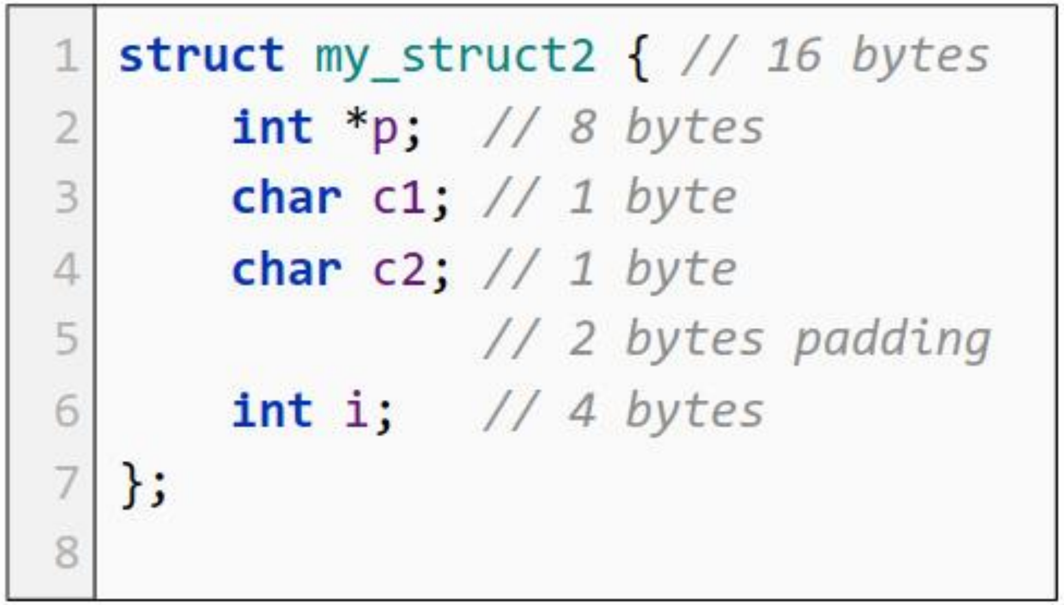
```
#include <stdlib.h>
#include <stdio.h>

struct date {
    int day;
    int month;
    int year;
};

int main(void) {
    struct date today = {19, 5, 2025};
    struct date *ptoday = &today;

    printf("Tody is the %d.%d.%d\n", today.day, today.month, today.year);

    printf("But yesterday was the %d.%d.%d\n", ptoday->day -1, ptoday->month, ptoday->year);
}
```

Size of structs

- Size depends on order and computer architecture



```c
struct my_struct1 { // 24 bytes
    int *p;  // 8 bytes
    char c1; // 1 byte
             // 3 bytes padding
    int i;   // 4 bytes
    char c2; // 1 byte
             // 7 bytes padding
};
```



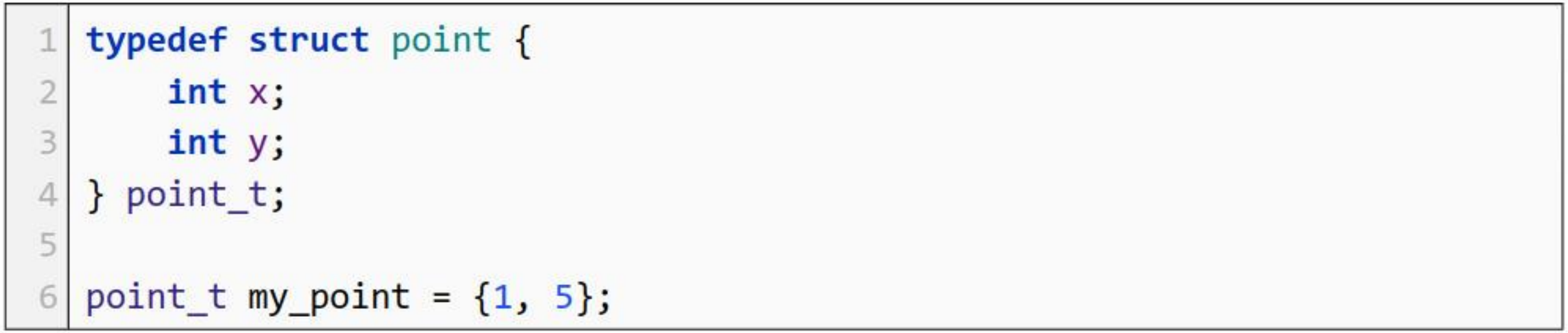
```c
struct my_struct2 { // 16 bytes
    int *p;  // 8 bytes
    char c1; // 1 byte
    char c2; // 1 byte
             // 2 bytes padding
    int i;   // 4 bytes
};
```

Typedef

- Typedef in a C structure gives a custom name (alias) to a struct

- Simplicity

- Can be used for other data types



```c
typedef struct point {
    int x;
    int y;
} point_t;

point_t my_point = {1, 5};
```

Functions with structs

Input (Arguments)

- Members one at a time
- Structs themselves
- Pointer to structs

Return Values

- Return Struct
- Return member

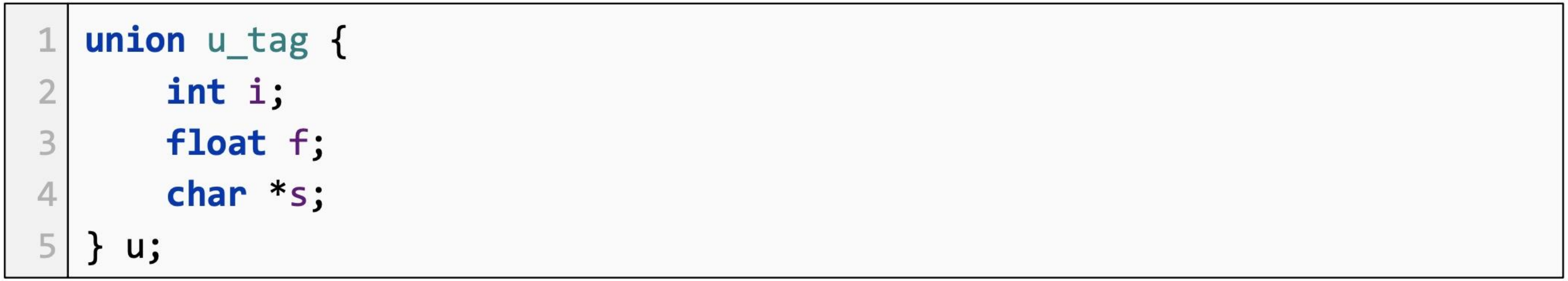
Unions

- All members share the same memory
- Only one member can hold a value at a time
- Implementation very similar to structs

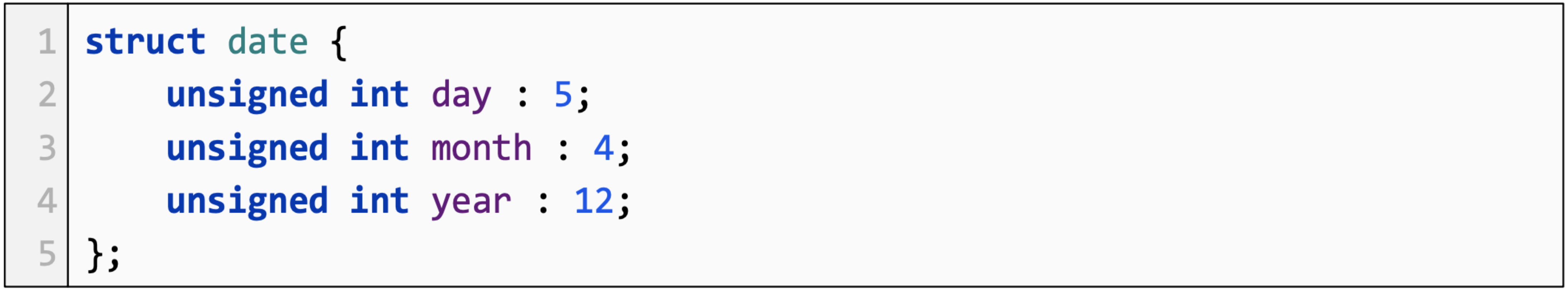
Bitfields

- Part of struct or union
- disk space optimization
- Declare bit size
- Many downsides

Example: Unions and Bitfields



```c
union u_tag {
    int i;
    float f;
    char *s;
} u;
```



```c
struct date {
    unsigned int day : 5;
    unsigned int month : 4;
    unsigned int year : 12;
};
```

Task 1: Elastic collision

- Collision between two point-masses
- Program takes in starting conditions
- Computation and output of end state

Task 1: Elastic collision

- Implementation with structures due to its convenient properties for this problem
- Input via command line
- Initialization of struct content
- Actual computation in separate function with distinction of cases

Task 1: Elastic collision

- Function takes struct as input parameter

- Function checks if certain condition actually takes place

- Computed end state is returned to main function and displayed

- Verification of implementation with special starting conditions

Task 2: Rocket equation

- Acceleration of rocket

- Program takes in starting values

- Calculating end speed, distance travelled and time of acceleration

Thank you for your attention

A presentation by: Christoph B., Leander D., Jakob G. and

Mario N.