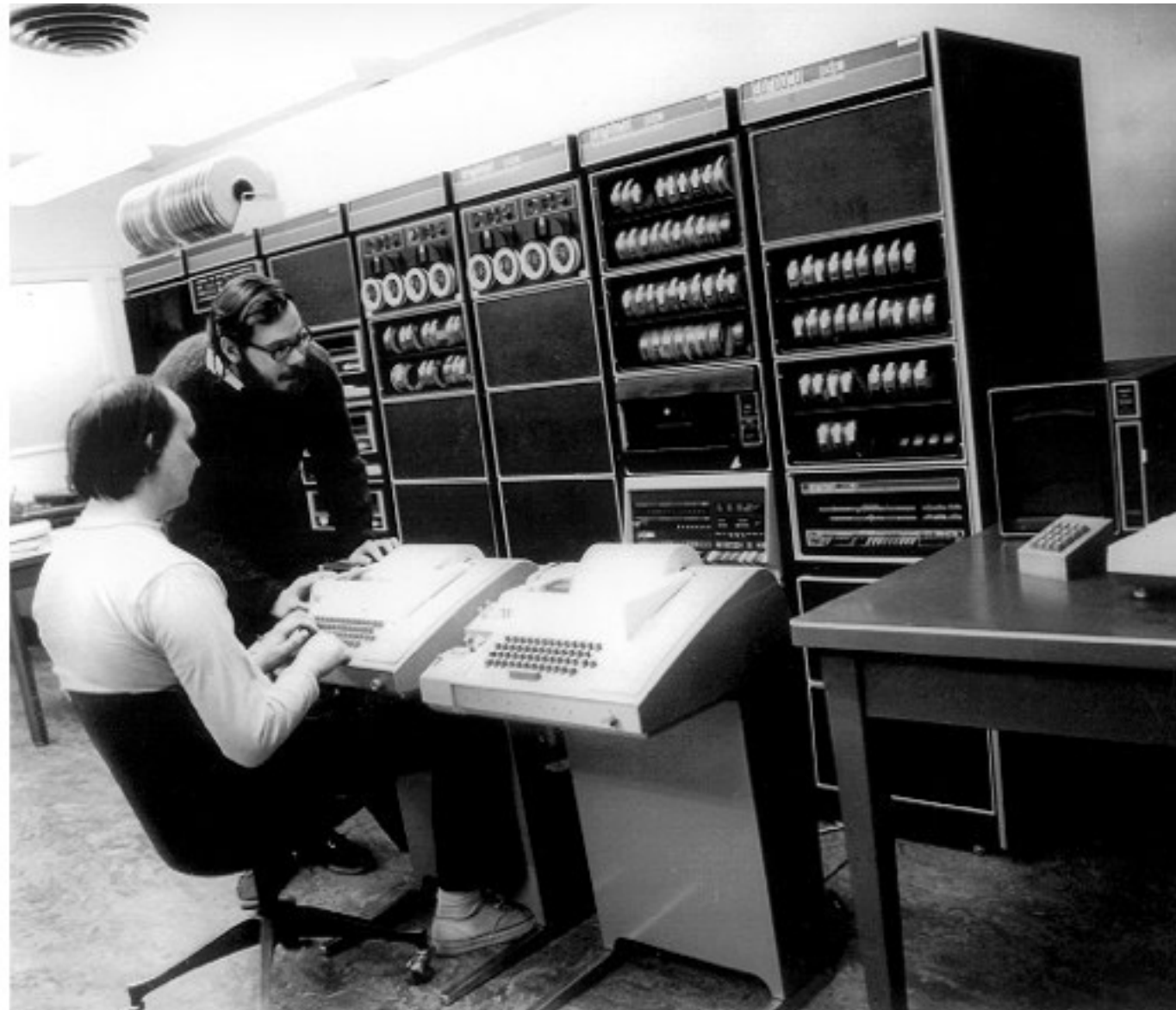


# Structures

“to get a deeper understanding of the language”



Deep C - a 3 day course  
Jon Jagger & Olve Maudal

# structs

- a struct definition introduces a new type
- data members may be different types (unlike an array)
- structs may contain other structs

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


```
struct project
{
    const char * name;
    struct date deadline;
    ...
};
```

# struct typedef

- a struct's tagname and its typedef name are the same type
- giving them different names is misleading

*date.h*

```
struct date_tag
{
    ...
};
typedef struct date_tag date;
```



```
#include "date.h"
```

```
void delay(struct date_tag deadline);
```

```
#include "date.h"
```

```
void delay(date deadline);
```


 equivalent! ?

# struct typedef

- a struct's tagname and its typedef name\* should be the same

*date.h*

```
struct date
{
    ...
};
typedef struct date date;
```



```
#include "date.h"
```

```
void delay(struct date deadline);
```

```
#include "date.h"
```

```
void delay(date deadline);
```

↕ equivalent ✓

*\*if it has a typedef*

# initialization

- structs support a convenient { aggregate } initialisation
- allows const struct variables
- missing fields are default initialised
- not permitted for assignment
- list cannot be empty

```
struct date deadline = { 2015, may, 1 };
```



```
const struct project fubar =  
    { "fubar", { 2015, may, 1 } };
```



```
struct date deadline;  
deadline = { 2015, may, 1 };
```

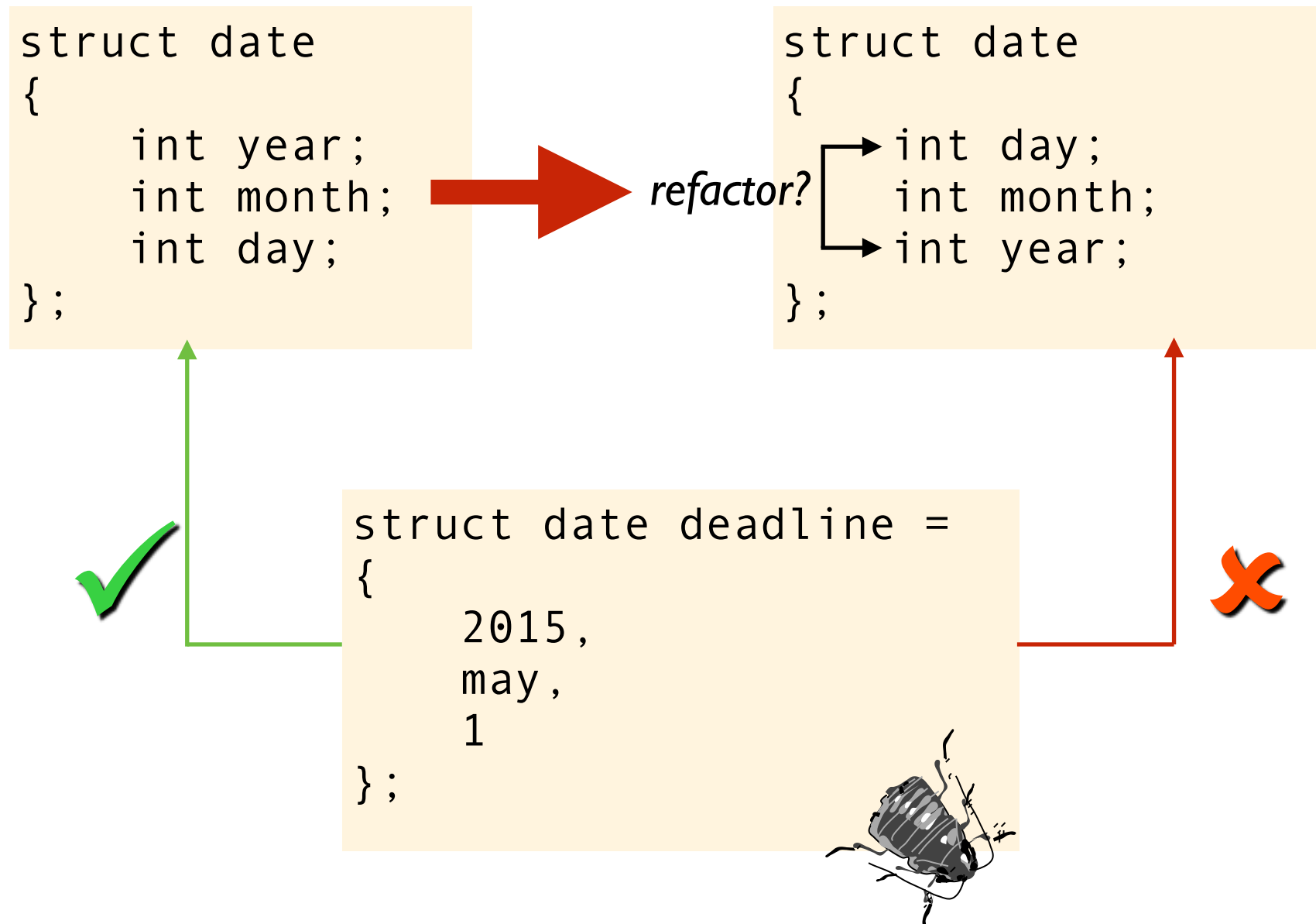


```
struct date deadline = { };
```



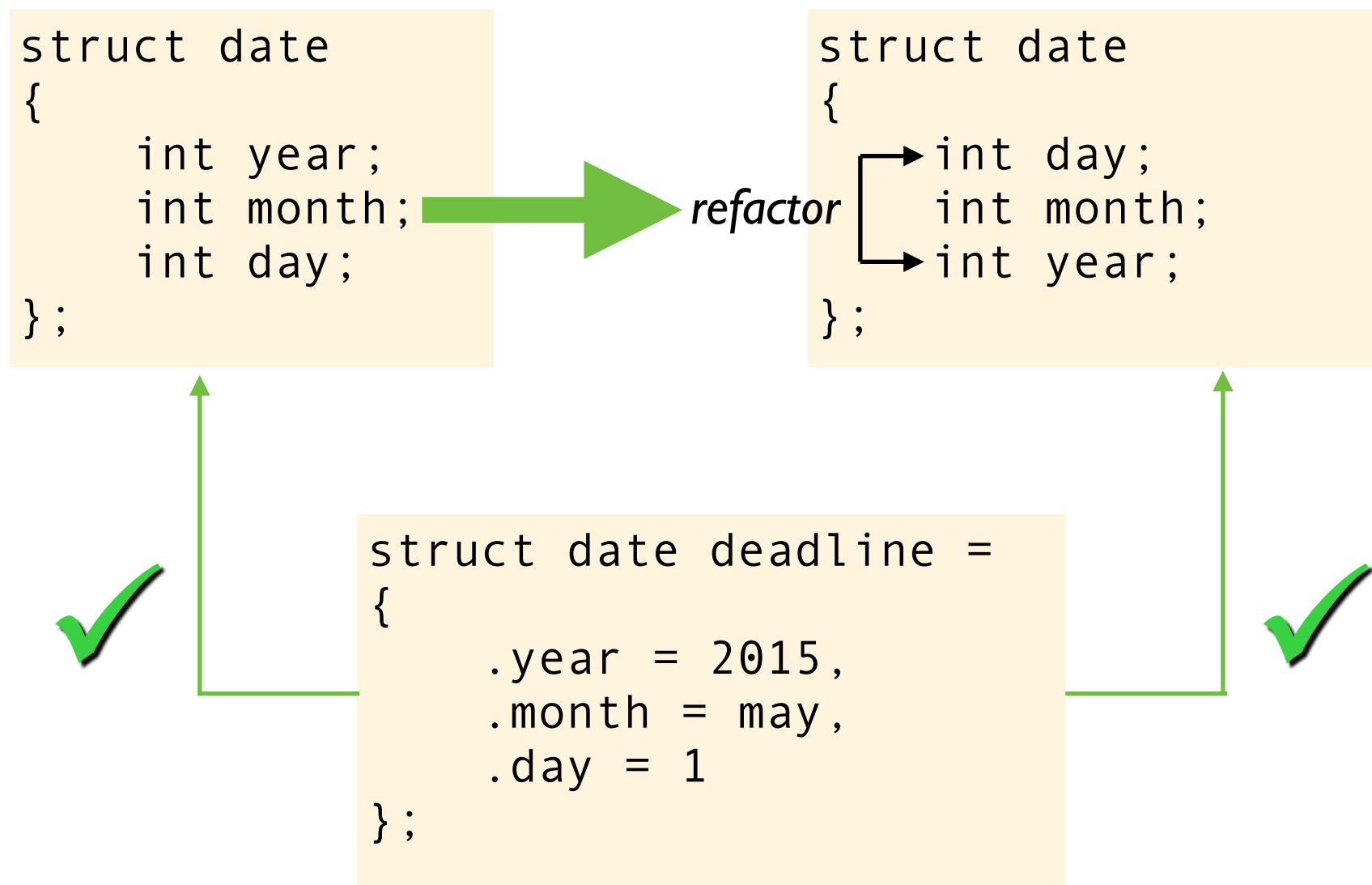
# representation dependency

- initialization in member order is a representation dependency :-(
- makes the representation harder to change



# dot designators c99

- structs support designator identifiers
- allows struct data members to be changed
- missing members are still default initialised



# compound literals c99

- aggregate initialization list can be "cast"
- "cast" type becomes type of expression
- assignment works with the "cast" :-)

*works for plain initialiser lists*

```
deadline = (struct date){ 2015, may, 1 };
```

*works for dot designators*

```
deadline =  
    (struct date){ .year = 2015, .month = may, .day = 1 };
```

*allows creation of anonymous objects*

```
void f(struct date when);
```

```
f((struct date){ .year = 2015, .month = may, .day = 1 });  
f((const struct date){ 2015, may, 1 });
```



# compound literals c99

- you can even take the address of an anonymous object.
- the anonymous object pointed to...
  - has automatic storage class
  - is scoped to the enclosing block

```
int delay(const struct date * deadline);
```



```
delay(&(struct date){ .year = 2015, .month = may, .day = 1 });
```



```
delay(&(const struct date){ 2015, may, 1 });
```



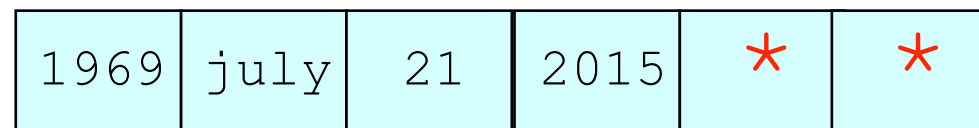
# arrays and structs

- `[int]` and `.identifier` designators can be combined

*array of structs*

```
struct date milestones[] =  
{  
    [0] = { 1969, july, 21 },  
    [1].year = 2015  
};
```

c99



milestones[0]

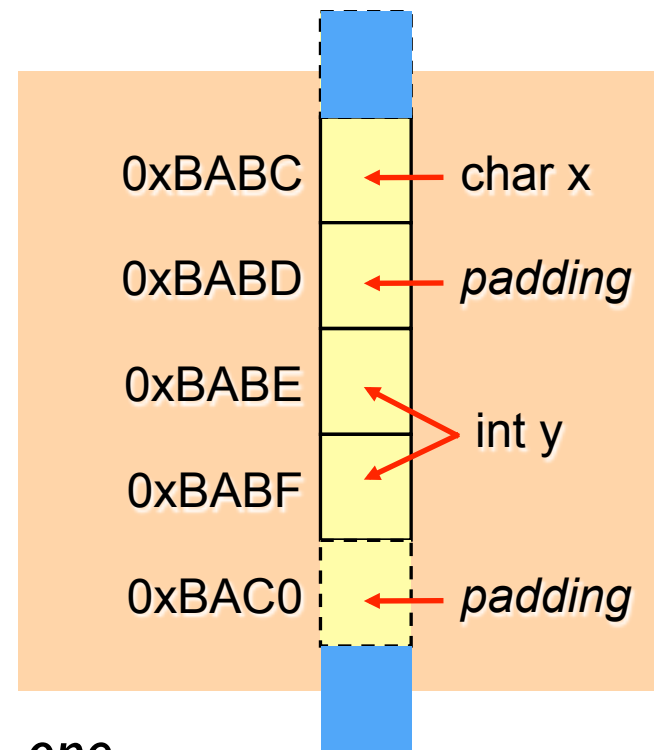
milestones[1]

\* default value = 0

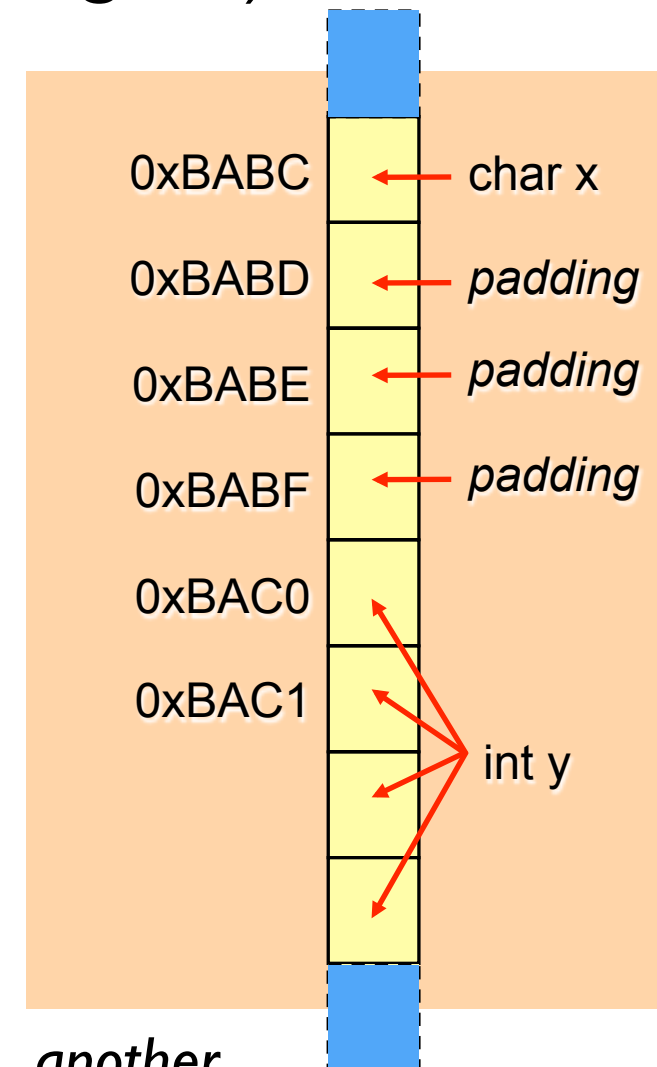
# struct alignment-padding

- types can have alignment restrictions
- first struct member determines struct's alignment
- padding can be added between struct members
- padding can be added after the last struct member  
(to ensure following struct in an array is aligned)

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



one  
alternative



another  
alternative

## struct =

- struct assignment *might* perform a simple bitwise copy
- struct assignment *might* not copy padding bytes

```
void some_func(const struct date * deadline)
{
    struct date delayed;
    memcpy(&delayed, deadline, sizeof delayed);
    ...
}
```



```
void some_func(const struct date * deadline)
{
    struct date delayed = *deadline;
    ...
}
```



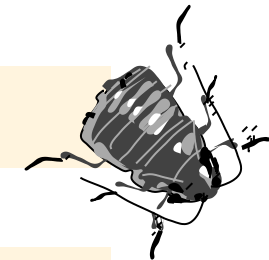
## struct ==

- structs do not support == or != operators
- using memcmp is flawed because of padding!
- only safe approach is to compare each member

```
struct date now = { ... };  
struct date due = { ... };
```

```
if (now == due) ❌ compile time error
```

```
if (memcmp(&now, &due, sizeof now) == 0)
```




```
bool date_equal(const struct date * lhs,  
               const struct date * rhs)  
{  
    return lhs->year == rhs->year &&  
           lhs->month == rhs->month &&  
           lhs->day == rhs->day;  
}
```



# struct hack c99

- last member may be an incomplete array type
- can't be the only member of the struct
- as if size of array is zero?!

```
struct small_message
{
    char size;
    char letters[];
};
```

 *incomplete array type*

```
struct small_message var;
assert(sizeof var == 1);
assert(sizeof(struct small_message) == 1);

struct small_message array[42];
assert(sizeof array == 42);
assert(sizeof(struct small_message[42]) == 42);
```

# struct hack

- allows struct type to overlay memory
- as if variable-length array was last member

```
...  
int main(void)  
{  
    char buffer[] =  
    {  
        5, 'h', 'e', 'l', 'l', 'o',  
        2, ' ', ' ',  
        5, 'w', 'o', 'r', 'l', 'd',  
        1, '\n',  
    };  
    const struct small_message * msg;  
    int n = 0;  
    do {  
        msg = (const struct small_message *)&buffer[n++];  
        n += printf("%.*s", msg->size, msg->letters);  
    } while (msg->size != 0);  
}
```

hello, world

# summary

- make struct's tagname and typedef name the same
- structs are the most common user defined type
- structs support = assignment
- structs do not support == equality (beware padding)
- struct typically pad between members
- structs have a rich compound literal syntax (c99)
- struct hack allows variable length struct (c99)