



# Structs and Enums

- these are the two recommended styles
  - ◆ any deviation from these two is unhelpful

tag name 

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

 typedef name

if you use a typedef  
make the typedef  
name the same as  
the tag name.

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

or don't typedef.

- making the tag\_name and the typedef name different creates a name difference when there is no actual type difference

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



```
struct date_tag deadline;  
...  
date delay = deadline;
```



- omit the tag name?
  - ◆ this forces a single syntax
  - ◆ struct on a declaration smells a bit hungarian?

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



```
struct date_tag deadline;  
...  
date delay = deadline;
```



```
date deadline;  
...  
date delay = deadline;
```



- omit the tag name?
  - ◆ also changes the namespace
  - ◆ you can't use the type name as an object name

```
struct date
{
    ...
};

struct date date;
```



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

date date;
```

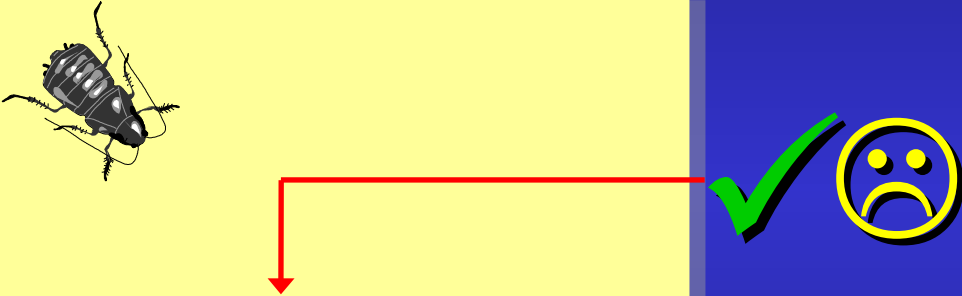


- omit the tag name?
  - ◆ also prevents a bizarre bug
  - ◆ `struct X;` means `X` does not have to already exist
  - ◆ `struct X;` forces `X` to name a possibly new type!

```

struct date
{
    ...
};

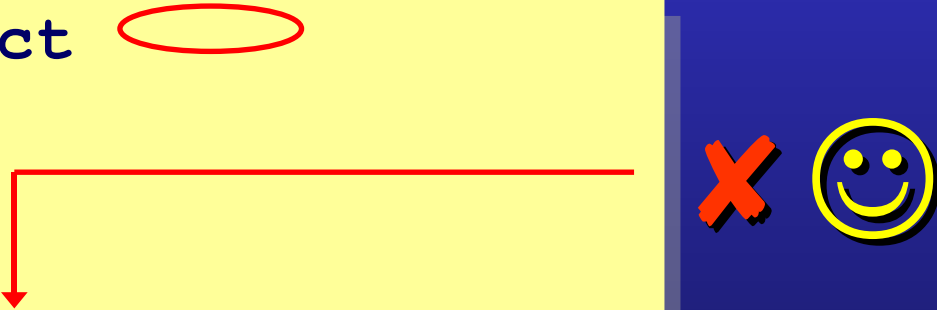
void func(struct date_t *);
  
```



```

typedef struct
{
    ...
} date;

void func(date_t *);
  
```



- omit the tag name?
  - ◆ is not an option for cyclically dependent types

```
struct link
{
    struct link * next;
    ...
};
```



```
typedef struct  
{
    struct link * next;
} link;
```



- omit the tag name?
  - ◆ prevents forward declarations
  - ◆ forces unnecessary #includes

date.h

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

use.h

```
#include "date.h"  
void f(date d);
```



- writing a tag name?
  - ◆ allows forward declarations
  - ◆ allows omission of unnecessary #includes



date.h

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


```
typedef struct date date;
void f(date d);
```

date.h


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

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

- these are the two recommended styles
  - ◆ any variation is unhelpful

tag name 

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

 typedef name

if you use a typedef  
make the typedef  
name the same as  
the tag name.

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

or don't typedef.

- a typedef does not create a new type
  - ◆ perhaps it should have been named typedefcl ?

```
typedef int mile;
```

```
typedef int kilometer;
```

```
void weak(mile lhs, kilometer rhs)
{
    lhs = rhs;
    ...
}
```



- a struct does introduce a new type name
  - ◆ use a wrapper type instead...

```
typedef struct mile
{
    int value;
} mile;
```



```
typedef struct kilometer
{
    int value;
} kilometer;
```



```
void strong(mile lhs, kilometer rhs)
{
    lhs = rhs;
}
```



- a struct wrapper does not introduces any extra runtime overhead

mile1.c

```
typedef struct mile
{
    int value;
} mile;

void f(int n)
{
    mile m = { n };
    if (m.value % 2)
        ...
}
```

mile2.c

```
typedef int mile;


void f(int n)
{
    mile m = n;
    if (m % 2)
        ...
}
```

```
>gcc -O2 -S mile1.c
>gcc -O2 -S mile2.c
>diff mile1.s mile2.s
→ no difference
```

- struct layout – the compiler...
  - ◆ can insert padding between members and after the last member
  - ◆ cannot insert padding before the first element or re-order the members

```
#include <stddef.h> // offsetof

const size_t co = offsetof(wibble, c);
const size_t cs = offsetof(wibble, s);
const size_t ci = offsetof(wibble, i);
```



```
co == 0
cs >= co + sizeof(char)
ci >= cs + sizeof(short)
```

```
typedef struct wibble
{
    char c;
    short s;
    int m;
} wibble;
```

- **structs support initialization and assignment**
  - ◆ not necessarily implemented as a memcpy
  - ◆ some other implementation might be faster!  
e.g. member by member copying avoiding memcpy of many padding bytes
  - ◆ hence == and != are not supported

```
typedef struct wibble  
{  
    ...  
} wibble;
```


```
wibble w = { ... };  
wibble copy = w;  
    copy = w;
```

```
if (copy == w) ...
```




- a struct can contain an array
  - ◆ struct assignment copies the whole array!
  - ◆ no extra overhead compared to a memcpy


```
typedef struct name
{
    char letters[64];
} name;
```



```
void strong(name * dst, const name * src)
{
    *dst = *src;
}
```



```
void strong(name * dst, const char * src)
{
    assert(strlen(src) < sizeof *dst);
    strcpy(dst->letters, src);
}
```



- **structs support designator identifiers**
  - ◆ allows for a small useful degree of ignorance

```
<stdlib.h>
```

```
7.20.6.2 The div ... Functions
```

```
div_t div(int numer, int denom);
```

The div...functions return a structure of type div\_t...The structures shall contains

(in either order)

the members quot (the quotient) and rem (the remainder)...

```
struct div_t d = { 9, 6 }; —————
```



```
struct div_t d;
d.quot = 9;
d.rem = 6;
```



```
struct div_t d = { .quot = 9, .rem = 6 }; —————
```



```
#define MAX_LEN (1024)
char buffer[MAX_LEN];
```

Avoid the  
preprocessor  
when possible

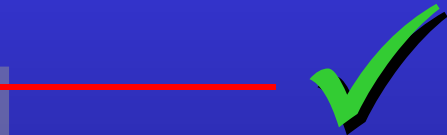
worse

```
enum { MAX_LEN = 1024 };
char buffer[MAX_LEN];
```

All uppercase is  
reserved for the  
preprocessor

better?

```
enum { max_len = 1024 };
char buffer[max_len];
```



best?

```
typedef struct x912_buffer
{
    char bytes[1024];
} buffer;
```



- useful for designators and switch labels

```
enum   { january, february, march,  
    ...  
    october, november, december  
};
```

```
const int days_in_month[] =  
{  
    [january] = 31,  
    [february] = 28,  
    ...  
    [november] = 30,  
    [december] = 31  
};
```



```
switch (...)  
{  
    case january: ...  
    case february: ...  
    ...  
    case november: ...  
    case december: ...  
}
```



- **enums cannot be forward declared**
  - ◆ or have cyclic dependencies
  - ◆ so never any need for a tag name

```
typedef enum suit
{
    clubs, diamonds, hearts, spades
} suit;
```

```
enum suit trumps;
suit lead;
```



```
typedef enum  
{
    clubs, diamonds, hearts, spades
} suit;
```

```
enum suit trumps;
suit lead;
```



**weak enum**

- **enums are very weakly typed**
  - ◆ **an enum's enumerators are of type integer, not of the enum type itself!**

```
typedef enum
{
    clubs, diamonds, hearts, spades
} suit;
```



```
typedef enum
{
    spring, summer, autumn, winter
} season;
```



```
void weak(suit trumps, season now)
{
    trumps = now;
}
```



- an enum can also be wrapped in a struct

```
typedef struct suit
{
    enum
    {
        clubs, diamonds, hearts, spades
    } value;
} suit;
```



```
typedef struct season
{
    enum
    {
        spring, summer, autumn, winter
    } value;
} season;
```



- **struct wrapper provides strong typing**
  - ◆ expressions that shouldn't compile don't
- **enum enumerators still available**
  - ◆ important for compile time constructs, eg switch
- **wrapper type is a struct**
  - ◆ forward declaration now possible

```
void strong(suit trumps, season now)
{
    trumps = now;
    switch (now.value)
    {
        case spring: ...
        case summer: ...
        case autumn: ...
        case winter: ...
    }
}
```



- a struct wrapper does not introduce any extra runtime overhead

suit1.c

```
typedef struct suit
{
    enum { clubs, diamonds, hearts, spades } value;
} suit;

void f(int n)
{
    suit trumps = { n % 2 ? clubs : diamonds };
    if (trumps.value == clubs)
        ...
}
```

- a struct wrapper does not introduce any extra runtime overhead

suit2.c

```
typedef enum suit
{
    clubs, diamonds, hearts, spades
} suit;

void f(int n)
{
    suit trumps = n % 2 ? clubs : diamonds;
    if (trumps == clubs)
        ...
}
```

```
>gcc -O2 -S suit1.c
>gcc -O2 -S suit2.c
>diff suit1.s suit2.s
→ no difference
```

- what might this look like if refactored to a wrapped enum?

```
#define PKG_ID          0
#define PKG_VERSION    3
#define PKG_CKSUM      4
#define PKG_FILESIZE   8
#define PKG_FILE_CKSUM 12
...
```

exercise



```
typedef struct package_offset
{
    enum
    {
        package_offset_id = 0,
        package_offset_version = 3,
        package_offset_checksum = 4,
        package_offset_filesize = 8,
        package_offset_file_checksum = 12,
    } value;
} package_offset;
```