`sizeof (expr)` Evaluates to Number of Bytes Needed

```
struct book_t book;
```

When you need
- the size of a structure,
- use the **`sizeof ()`** operator
- with a variable or an expression.

For example,

```
sizeof (book)
```

evaluates to the **number of bytes occupied by the variable `book`** (a `struct book_t`).

Pitfall: Using `sizeof` with a Type

You will see code using a type with **`sizeof`**.

For example,
- **`sizeof (struct book_t)`** in place of
- **`sizeof (book)`**.

This code will work correctly…

…until someone changes the type of `book`.

Just hope that they remember to change the type used with **`sizeof`**, too.

Pitfall: "Calculating" Sizes

You **may want to calculate a size** yourself.

Avoid doing so if possible:
- **sizes change** from ISA to ISA,
- and sometimes from OS to OS,
- or even from compiler to compiler.

Compilers must guarantee aligned accesses
- and thus **sometimes insert padding**
- between fields or at the end of a **`struct`**.

Most ISAs Impose Alignment Requirements

What is an alignment requirement?

Most **ISAs** (with byte-addressable memory) **require that**
- **loads and stores of N bytes**
- **use addresses that are multiples of N**.

For example,
- trying to load a 32-bit value (4B)
- from address 0x20000001 (= 1 mod 4)
- causes a program to crash.