Including List Element is Faster, But Allows Just One List

In other words,
- **create a list element** structure
- **consisting of a `next` field** (only), and
- **include** a **list element** structure
- **as first element of** the **"thing"** structure.

For list code, a "thing" is just a list node.

List nodes can be turned into "things" (the address is the same)!

But **can only have one list** for any "thing."

Example: Cyclic, Doubly-Linked List with a Sentinel

Let's build an **example container**:
- **cyclic, doubly-linked list with a sentinel**
- **using list element inclusion** (rather than extra pointers).

In particular,
- a **`double_list_t`** includes pointers to **`next`** and **`prev`** **`double_list_t`**'s, and
- first field of "thing" is a **`double_list_t`**.

A List Node is Simple: Just Two Pointers

```
typedef struct double_list_t
              double_list_t;
struct double_list_t {
    double_list_t* prev;
    double_list_t* next;
};
```

How do we initialize a list?

List Variables Can be Statically or Dynamically Initialized

List variables can be **initialized statically**:

```
// this variable is the sentinel
static double_list_t my_list =
    {&my_list, &my_list};
```

But **dynamically-allocated lists** must be **initialized at runtime**.