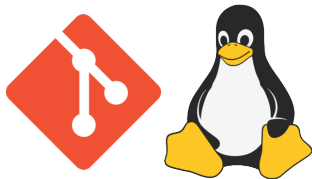


# Object Oriented Programming in C: A Case Study



Git and Kernel Linux

Matheus Tavares <matheus.bernardino@usp.br>

Renato Lui Geh <renatolg@ime.usp.br>

Institute of Mathematics and Statistics - University of São Paulo



# Introduction

# OO Techniques in non-OO Languages

**Motivation:** Combine good features of a language with good but absent techniques from other languages.

*No programming technique solves all problems.*

*No programming language produces only correct results.*

— Schreiner, Axel T (1993). *Object-Oriented Programming With ANSI-C*

# Content

We'll see the following concepts implemented in C:

- ▶ Private attributes
- ▶ Private methods
- ▶ “Public” Inheritance
- ▶ “Private” Inheritance
- ▶ Multiple inheritance
- ▶ Abstract Classes and Polymorphism
- ▶ Metaprogramming
- ▶ Design Pattern: Iterator

In particular, we'll talk about implementation examples of these concepts in the **Git** and **Kernel Linux IIO** codebases.

# Kernel Linux IIO

# Linux IIO

- ▶ `iio_dev`: Industrial Input/Output Device
- ▶ `ad7780_state`: Analog to Digital Sigma-Delta Converter Device



- ▶ **struct** `ad7780_state` specialization of **struct** `iio_dev`.
- ▶ In other words, `ad7780_state` is a **subclass** of `iio_dev`.

# Inheritance

## Inheritance by composition

Let  $S$  be the superclass, and  $C$  a subclass of  $S$ . Assume  $n$  and  $m$ ,  $S$  and  $C$ 's memory size in bytes. We create an object of  $C$  in the following way:

1. Allocate a block  $B$  of size  $n + m + a$  (in bytes);
2. Save first  $n$  bytes  $[0, n]$  for  $S$ ;
3. Save last  $m$  bytes  $[n + a, n + a + m]$  for  $C$ ;
4. Return a pointer to block  $B$  of type  $*S$ .

$C$  is now a “private” object of  $S$ .

# Inside the Kernel

## Definitions

$S := \text{iio\_dev}$

$C := \text{ad7780\_state}$

$n := \text{sizeof}(\text{struct iio\_dev})$

$m := \text{sizeof}(\text{struct ad7780\_state})$

Function `devm_iio_device_alloc` allocs block  $B$  and returns a pointer `struct iio_dev*` to the beginning of the block.

- ▶ `drivers/iio/ad7780.c:ad7780_state`
- ▶ `include/linux/iio/iio.h:iio_dev`



# Access to the subclass

How to access  $c$  from an  $s$  pointer given the address of a block  $B$ ?

```
#define ALIGN(x, a) ALIGN_MASK(x, (typeof(x))(a)-1)
#define ALIGN_MASK(x, mask) (((x) + (mask)) & ~(mask))
#define ALIGN_BYTES CACHE_BYTES_IN_POWER_OF_2

static inline void *priv(const struct S *s) {
    /* Returns a pointer to c "hidden" in s. */
    return (char*) s + ALIGN(sizeof(struct S), ALIGN_BYTES);
}
```

- ▶ include/linux/kernel.h:ALIGN
- ▶ include/linux/iio/iio.h:iio\_priv
- ▶ include/uapi/linux/kernel.h:\_\_ALIGN\_KERNEL\_MASK

# Understanding ALIGN

ALIGN is a function parameterized by the size of  $S$  and some power of two.

Recall from CS101...

Access to memory is faster when address is a power of two.

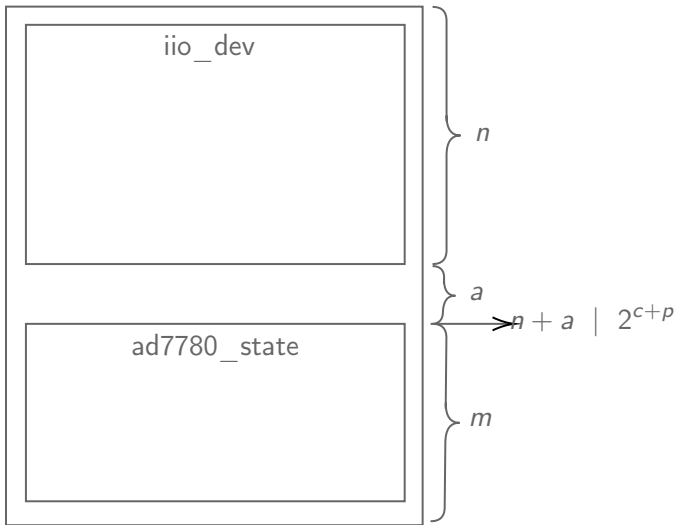
We want to access an address of the alloc'ed memory somewhere near  $s + \text{sizeof}(S)$ , and it must be a power of two.

(see next slide)

**Claim**

$\text{ALIGN}(x, a)$  is the smallest multiple of  $a$  greater than  $x$ .

```
ad7780_state* = priv(iio_dev*) = iio_dev* + ALIGN(...)
```



**Figure:** “Private” inheritance. Given `iio_dev*`, how do we find `ad7780_state*` in a fast, memory efficient way?

## Lemma 1

Let  $n, m \in \mathbb{Z}_k$ , st  $m = 2^c - 1$  for some  $c \in \mathbb{N}$ ,  $c < k$ . Then

$$(n + m) \& \sim(m) \mid 2^{c+p}, \text{ for some } p \in \mathbb{N}.$$

### Proof.

First note that  $m = 2^c - 1 = (0 \dots 01 \dots 1)_2$ , so  $(\sim m) = (1 \dots 10 \dots 0)_2$ . Therefore  $(n + m) \& \sim(m)$  are the most significant bits of  $n + 2^c - 1$  starting from the  $c$ -th bit. But that's exactly taking  $n + m$  and "subtracting" all the least significant bits starting from  $c - 1$ . More formally,

$$(n + m) \& \sim(m) = (n + m) - ((n + m) \& (2^c - 1)).$$

The right-hand term on the right side of the equality can be rewritten as

$$((n + m) \& (2^c - 1)) \equiv (n + m) \pmod{2^c}.$$

In other words, taking the  $c - 1$  least significant bits is equivalent to taking the remainder of the division by  $2^c$ . To give some intuition, note that the  $2^i$  bits for all  $i > c$  are all multiples of  $2^c$ , and therefore equivalent to zero. From that

$$\begin{aligned}(n + m) \& \sim (m) &= (n + m) - ((n + m) \pmod{2^c}) \\ &= (n + 2^c - 1) - ((n + 2^c - 1) \pmod{2^c}) \\ &= (n + 2^c - 1) - ((n - 1) \pmod{2^c}).\end{aligned}$$

If  $n < 2^c$ , then  $(n + 2^c - 1) - n + 1 = 2^c \mid 2^c$  and therefore the hypothesis is trivially true. The same can be said when  $n = 2^c$ . Now, assuming  $n > 2^c$ , let's analyze the two possible cases for  $n$ .

**Case 1:**  $2^c \mid n$

$$\begin{aligned}(n + m) \& \sim (m) &= (n + 2^c - 1) - ((n - 1) \bmod 2^c) \\ &= (n + 2^c - 1) + 1 \\ &= (n + 2^c) \mid 2^{c+p}. \quad (\text{by assumption})\end{aligned}$$

**Case 2:**  $2^c \nmid n$

$$\begin{aligned}(n + m) \& \sim (m) &= (n + 2^c - 1) - ((n - 1) \bmod 2^c) \\ &= (n - r + r + 2^c - 1) - ((r - 1) \bmod 2^c),\end{aligned}$$

where  $r = n \bmod 2^c$ , that is, the “remainder” of  $n/2^c$ .

We get two subcases: when  $0 \equiv r - 1 \pmod{2^c}$ , and thus

$$\begin{aligned}(n + m) \& \sim (m) &= (n + 2^c - 1) - ((r - 1) \pmod{2^c}) \\ &= (n - r + r + 2^c - 1) \\ &= ((n - r) + (r - 1) + 2^c) \mid 2^{c+p},\end{aligned}$$

Note that  $n - r = 2^c$ , since  $r = n \pmod{2^c}$ ; and  $r - 1 = 0$ , as  $r \equiv 1 \pmod{2^c}$  and  $r < 2^c$  by definition. This means  $(n - r) + (r - 1) = 2^c$ , and thus  $(2^c + 2^c) = 2^{c+1} \mid 2^{c+p}$ .

Finally, we analyze the last case:  $0 \not\equiv r - 1 \pmod{2^c}$ . In this subcase, since  $r < 2^c$ ,  $(r - 1 \pmod{2^c}) = r - 1$  so:

$$\begin{aligned}(n + m) \& \sim (m) &= (n + 2^c - 1) - ((r - 1) \pmod{2^c}) \\ &= (n - r + r + 2^c - 1) - r + 1 \\ &= ((n - r) + 2^c) \mid 2^{c+p}.\end{aligned}$$

And therefore  $(n + m) \& \sim (m) \mid 2^{c+p}$ , for some integer  $p$ . □



## Lemma 2

Let  $n, m \in \mathbb{Z}_k$ , st  $m = 2^c - 1$  for some  $c \in \mathbb{N}$ ,  $c < k$ . Then

$$t = (n + m) \& \sim (m)$$

Is the smallest multiple of  $2^c$  greater than  $n$ .

### Proof.

We will show by exhaustion that  $t$  is in fact the minimum candidate multiple of  $2^c$  with respect to  $n$ . Recall the cases shown in Lemma 1's proof. When  $n < 2^c$ ,

$$\begin{aligned}(n + m) \& \sim (m) &= (n + 2^c - 1) - ((n - 1) \bmod 2^c) \\ &= n + 2^c - 1 - n + 1 \\ &= 2^c = t\end{aligned}$$

and therefore  $t$  is the smallest multiple of  $2^c$  greater than  $n$ .

For  $n = 2^c$ ,

$$\begin{aligned}(n + m) \& \sim (m) &= (n + 2^c - 1) - ((n - 1) \bmod 2^c) \\ &= 2^c + 2^c - 1 + 1 \\ &= 2^c + 2^c = t,\end{aligned}$$

again  $t$  is the “minimum” multiple. Recall the two cases when  $n > 2^c$ .

**Case 1:**  $2^c \mid n$

$$\begin{aligned}(n + m) \& \sim (m) &= (n + 2^c - 1) - ((n - 1) \bmod 2^c) \\ &= (n + 2^c - 1) + 1 \\ &= n + 2^c = t\end{aligned}$$

If  $n$  is a multiple of  $2^c$ , then the next multiple greater than  $n$  is  $n + 2^c$ .

## Case 2: $2^c \nmid n$

When  $0 \equiv r - 1 \pmod{2^c}$ ,  $r = 1$ , since  $r < 2^c$  by definition.

$$\begin{aligned}(n + m) \& \sim (m) &= (n + 2^c - 1) - ((r - 1) \pmod{2^c}) \\ &= n - 1 + 2^c = t\end{aligned}$$

But if  $r = 1$ , then  $n - 1$  is a multiple of  $2^c$ , and therefore the smallest multiple greater than  $n$ , which is exactly  $n - 1 + 2^c$ .

For the last subcase, take  $0 \not\equiv r - 1 \pmod{2^c}$ . Then

$$\begin{aligned}(n + m) \& \sim (m) &= (n + 2^c - 1) - ((r - 1) \pmod{2^c}) \\ &= (n - r + r + 2^c - 1) - r + 1 \\ &= n - r + 2^c. = t\end{aligned}$$

Again,  $n - r$  is a multiple of  $2^c$  by definition, and therefore the next candidate is  $n - r + 2^c$ . □

## Claim

$\text{ALIGN}(x, a)$  is the smallest multiple of  $a$  greater than  $x$ .

Our claim, but fancier:

## Theorem 3

*The function  $\text{ALIGN}(\text{sizeof}(\text{struct } S), \text{ALIGN\_BYTES})$  returns the smallest address of memory multiple of  $\text{ALIGN\_BYTES}$  greater than  $\text{sizeof}(\text{struct } S)$ .*

**Proof.**

Follows directly from Lemma 1 and Lemma 2.



# ALIGN in the wild

```
static int ad7780_probe(struct spi_device *spi) {
    struct ad7780_state *st;
    struct iio_dev *indio_dev;

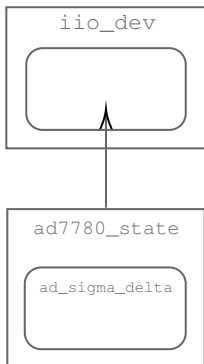
    indio_dev = devm_iio_device_alloc(&spi->dev, sizeof(*st));
    if (!indio_dev) return -ENOMEM;

    st = iio_priv(indio_dev);
    st->gain = 1;
    ...
}
```

► drivers/iio/adc/ad7780.c:ad7780\_probe

# Multiple inheritance

- ▶ `ad7780_state` child of `iio_dev` (“**private**” inheritance);
- ▶ `ad7780_state` child of `ad_sigma_delta` (“**public**” inheritance).



Both use inheritance by composition, but in different ways.

# “Public” vs “private” inheritance

## Private inheritance

As seen on `iio_dev` and `ad7780_state`.

- ▶ Subclass attributes are private;
- ▶ Runtime inheritance;
- ▶ Subclass could be of any type (`ad7780_state`, `ad7793_state`, `mcp3422`, etc).

We shall now see “public” inheritance.

## Public inheritance

To be seen on `ad_sigma_delta` and `ad7780_state`.

- ▶ Attributes of superclass and subclass are public;
- ▶ Compile-time inheritance;

`ad_sigma_delta`: Analog-Digital Sigma-Delta Converter (ADSD)

`ad7780_state`: ADSD Converter for AD7170/1 and AD7780/1

```
struct ad7780_state {
    const struct ad7780_chip_info *chip_info;
    struct regulator *reg;
    struct gpio_desc *powerdown_gpio;
    ...
    unsigned int int_vref_mv;

    struct ad_sigma_delta sd;
}
```

In private inheritance, the **superclass** (`iio_dev`) contains the **subclass** (`ad7780_state`).

In public inheritance, the **subclass** (`ad7780_state`) contains the **superclass** (`ad_sigma_delta`).



# Access to the subclass

How to access object *c* of subclass *C* when object *s* of type *S* is *inside C*?

```
#define container_of(ptr, type, member) \
    (type*)((void*)(ptr) - ((size_t)&((type*)0)->member))

static struct ad7780_state *ad_sigma_delta_to_ad7780(
    struct ad_sigma_delta *sd) {
    return container_of(sd, struct ad7780_state, sd);
}
```

- ▶ drivers/iio/adc/ad7780.c:ad\_sigma\_delta\_to\_ad7780
- ▶ include/linux/kernel.h:container\_of
- ▶ include/linux/stddef.h:offsetof

# Understanding container\_of

We want to access the “outer” pointer, that is, find the pointer `struct ad7780_state*` that contains `struct ad_sigma_delta*`.

```
#define container_of(ptr, type, member) \  
    (type*)((void*)(ptr) - ((size_t)&((type*)0)->member))  
  
static struct ad7780_state *ad_sigma_delta_to_ad7780(  
    struct ad_sigma_delta *sd) {  
    return container_of(sd, struct ad7780_state, sd);  
}
```

## Trick

`&((type*)0)->member`: returns the address of member as if `type*` were allocated to the zero-address. In other words, the size (in bytes) of type *up to* variable member. (see next slide)

```
container_of(p, C, p)=(C*)((void*)p-((size_t)&((C*)0)->p))
```

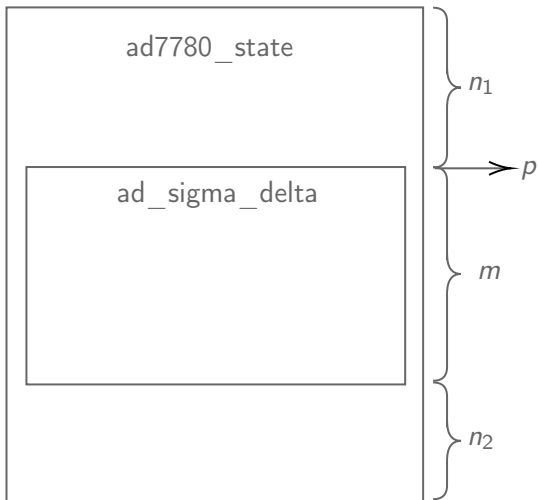
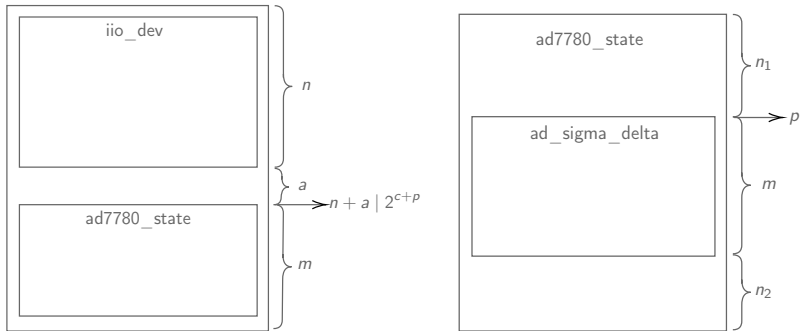


Figure: “Public” inheritance. Given `ad_sigma_delta*`, how do we find `ad7780_state*`?

# Summary



**Figure:** Different approaches for different uses. Which one is better? Depends on what you need.

Git

# The dir-iterator object

Usage example (simplified):

```
struct dir_iterator *iter = dir_iterator_begin(path);

while (dir_iterator_advance(iter) == ITER_OK) {
    if (want_to_stop_iteration()) {
        dir_iterator_abort(iter);
        break;
    }

    // Access information about the current path:
    if (S_ISDIR(iter->st.st_mode))
        printf("%s is a directory\n", iter->relative_path);
}
```

# API: public attributes and methods

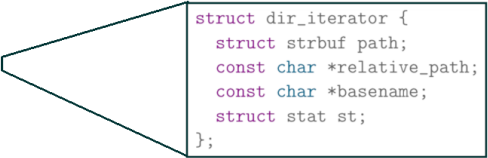
```
// The current iteration state, with path,  
// basename and etc.  
struct dir_iterator {  
    struct strbuf path;  
    const char *relative_path;  
    const char *basename;  
    struct stat st;  
};  
  
struct dir_iterator *dir_iterator_begin(const char *path);  
int dir_iterator_abort(struct dir_iterator *iterator);  
int dir_iterator_advance(struct dir_iterator *iterator);
```

▶ dir-iterator.h

# Full declaration and constructor

```
struct dir_iterator_int {
    struct dir_iterator base;

    size_t levels_nr;
    size_t levels_alloc;
    struct dir_iterator_level *levels;
};
```



```
struct dir_iterator {
    struct strbuf path;
    const char *relative_path;
    const char *basename;
    struct stat st;
};
```

```
struct dir_iterator *dir_iterator_begin(const char *path) {
    struct dir_iterator_int *iter = xmalloc(1, sizeof(*iter));
    struct dir_iterator *dir_iterator = &iter->base;
    ... /* Initialize fields. */
    return dir_iterator;
}
```

► dir-iterator.c

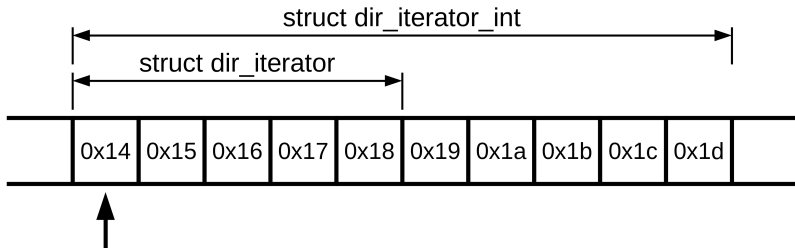


# How to access private attributes?

```
int dir_iterator_advance(struct dir_iterator *dir_iterator)
{
    struct dir_iterator_int *iter =
        (struct dir_iterator_int *)dir_iterator;
    // Use iter as needed
    ...
}
```

▶ dir-iterator.c

# Private attributes: how it works



- ▶ Use this technique with caution:
  - ▶ memcpy and others:  
`sizeof(struct dir_iterator) != sizeof(struct dir_iterator_int)`
  - ▶ arrays and initializations out of `dir_iterator_begin()`

# What about private methods?

```
int dir_iterator_advance(struct dir_iterator *dir_iterator)
{
    struct dir_iterator_int *iter =
        (struct dir_iterator_int *)dir_iterator;

    if (/* ... */ && push_level(iter))
        ...
}
```

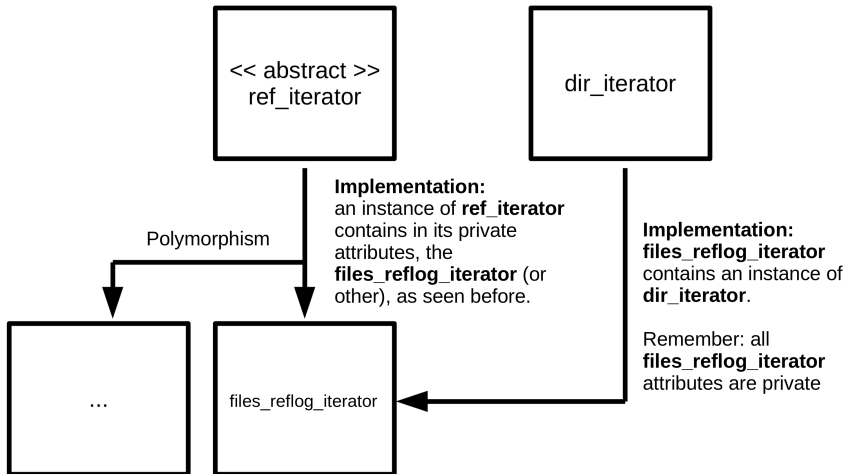
```
static int push_level(struct dir_iterator_int *iter)
{
    // Use iter as needed
}
```

▶ dir-iterator.c

# Abstract classes, inheritance and polymorphism

- ▶ `refs/refs-internal.h`
- ▶ `refs/iterator.c`
- ▶ `refs/files-backend.c`

# Big picture



# The abstract class ref\_iterator

```
struct ref_iterator {  
    struct ref_iterator_vtable *vtable;  
  
    unsigned int ordered : 1;  
    const char *refname;  
    const struct object_id *oid;  
    unsigned int flags;  
};
```

► refs/refs-internal.h

# ref\_iterator: abstract methods

```
int ref_iterator_advance(struct ref_iterator *ref_iterator)
{
    return ref_iterator->vtable->advance(ref_iterator);
}
```

```
int ref_iterator_abort(struct ref_iterator *ref_iterator)
{
    return ref_iterator->vtable->abort(ref_iterator);
}
```

► refs/iterator.c

# The sub-class reflog\_iterator

```
static struct ref_iterator *reflog_iterator_begin(struct ref_store *
                                                ref_store,
                                                const char *gitdir)
{
    struct files_reflog_iterator *iter = xcalloc(1, sizeof(*iter));
    struct ref_iterator *ref_iterator = &iter->base;
    struct strbuf sb = STRBUF_INIT;

    base_ref_iterator_init(ref_iterator,
                          &files_reflog_iterator_vtable, 0);
    strbuf_addf(&sb, "%s/logs", gitdir);
    iter->dir_iterator = dir_iterator_begin(sb.buf);
    iter->ref_store = ref_store;
    strbuf_release(&sb);

    return ref_iterator;
}
```

► refs/files-backend.c



# Multiple inheritance

```
static int files_reflog_iterator_advance(struct ref_iterator *ref_iterator)
{
    struct files_reflog_iterator *iter =
        (struct files_reflog_iterator *)ref_iterator;
    struct dir_iterator *diter = iter->dir_iterator;
    int ok;

    while ((ok = dir_iterator_advance(diter)) == ITER_OK) {
        int flags;
        ...
    }
}
```

► refs/files-backend.c

“Metaprogramming” in Git

# Metaprogramming

```
#include "cache.h"
```

```
...
```

```
define_commit_slab(blame_suspects, struct blame_origin *);  
static struct blame_suspects blame_suspects;
```

▶ blame.c

# Metaprogramming

```
#define define_commit_slab(slabname, elemtype)  \  
    declare_commit_slab(slabname, elemtype); \  
    implement_static_commit_slab(slabname, elemtype)
```

▶ commit-slab.h

# Metaprogramming

```
#define declare_commit_slab(slabname, elemtype) \  
\  
struct slabname { \  
    unsigned slab_size; \  
    unsigned stride; \  
    unsigned slab_count; \  
    elemtype **slab; \  
}
```

► commit-slab-decl.h

# Metaprogramming

```
#define implement_static_commit_slab(slabname, elemtype) \  
    implement_commit_slab(slabname, elemtype, MAYBE_UNUSED static)  
  
#define implement_commit_slab(slabname, elemtype, scope)      \  
                                                                \  
scope void init_ ##slabname## _with_stride(struct slabname *s, \  
                                             unsigned stride)    \  
{                                                                \  
    unsigned int elem_size;                                     \  
    if (!stride)                                              \  
        stride = 1;                                           \  
        ...                                                    \  
}                                                                \  
...
```

▶ commit-slab-impl.h

Questions?

# References I



Git.

URL: <https://git.kernel.org/pub/scm/git/git.git>.



Industrial input/output linux kernel subsystem.

URL: <https://git.kernel.org/pub/scm/linux/kernel/git/jic23/iio.git/>.



Axel Schreiner.

*Object-Oriented Programming with ANSI-C.*

Axel Schreiner / Lulu, first edition edition, 2011.

URL: <https://www.cs.rit.edu/~ats/books/ooc.pdf>.