# Pointers

指標就是力量

scc@teamt5.org

# Outline

TEAMT5
杜 浦 數 位 安 全

- The C World
  - 頭腦體操
  - Array
  - Compile time check
  - container_of
  - restrict qualifier
- The C++ World
  - Ownership
  - Virtuals and this pointer

# The C world

# 頭腦體操

TEAMT5
杜 浦 數 位 安 全

```
int **ptr;
int *(*ptr)[10];
int *(*ptr)(int *);
int *(*ptr[10])(int **);
int (*(*(*ptr(int *))[10]))(int *);
sizeof(int (******)[10][20][30][40]);
```

# 頭腦體操

TEAMT5
杜 浦 數 位 安 全

```
int **ptr;
int *(*ptr)[10];
int *(*ptr)(int *);
int *(*ptr[10])(int **);
int (*(*(*ptr(int *))[10]))(int *);
sizeof(int (******)[10][20][30][40]);
```

pointer to pointer

# 頭腦體操

TEAMT5
杜 浦 數 位 安 全

```
int **ptr;
int *(*ptr)[10];
int *(*ptr)(int *);
int *(*ptr[10])(int **);
int (*(*(*ptr(int *))[10]))(int *);
sizeof(int (******)[10][20][30][40]);
```

pointer to array 10 of pointer to int

# 頭腦體操

TEAMT5
杜 浦 數 位 安 全

```
int **ptr;
int *(*ptr)[10];
int *(*ptr)(int *);
int *(*ptr[10])(int **);
int (*(*(*ptr(int *))[10]))(int *);
sizeof(int (******)[10][20][30][40]);
```

function (pointer to int) returning pointer to int

# 頭腦體操

TEAMT5
杜 浦 數 位 安 全

```
int **ptr;
int *(*ptr)[10];
int *(*ptr)(int *);
int *(*ptr[10])(int **);
int (*(*(*ptr(int *))[10]))(int *);
sizeof(int (******)[10][20][30][40]);
```

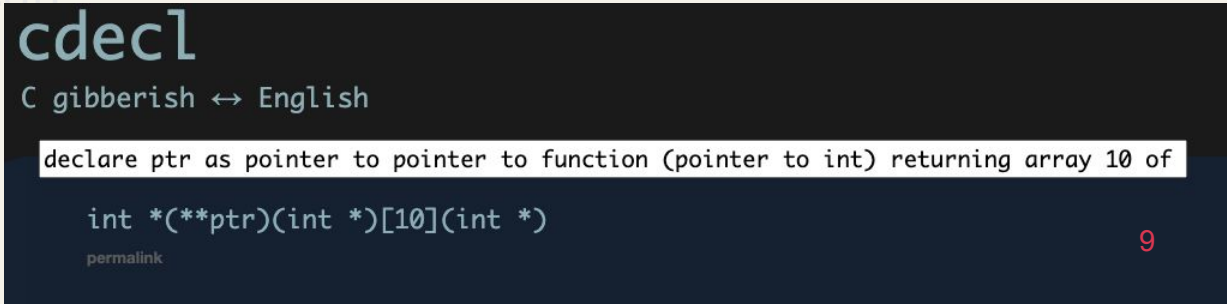
array 10 of pointers to function (pointer to pointer to int) returning a pointer to int

# 頭腦體操

TEAMT5
杜 浦 數 位 安 全

```
int **ptr;
int *(*ptr)[10];
int *(*ptr)(int *);
int *(*ptr[10])(int **);
int (*(*(*ptr(int *))[10]))(int *);
sizeof(int (******)[10][20][30][40]);
```

pointer to pointer to function (pointer to int) returning array 10 of function (pointer to int) returning pointer to int




https://cdecl.org/

# 頭腦體操

TEAMT5
杜 浦 數 位 安 全

```
int **ptr;
int *(*ptr)[10];
int *(*ptr)(int *);
int *(*ptr[10])(int **);
int (*(*(*ptr(int *))[10]))(int *);
sizeof(int (******)[10][20][30][40]);
```

pointer to pointer to pointer to pointer to pointer to pointer to array of 10 array of 20 array of 30 array of 40 int

# 頭腦體操

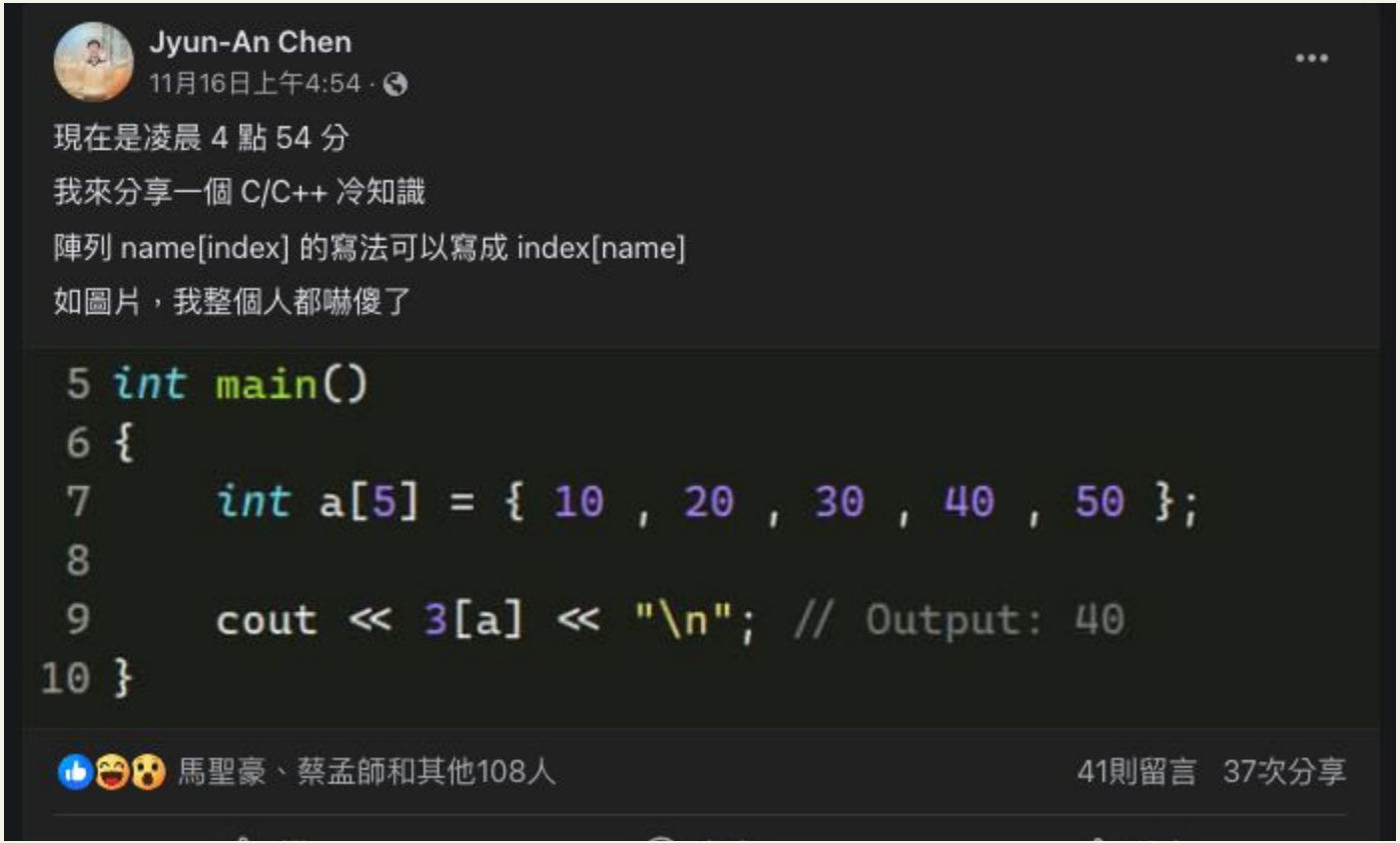
TEAMT5
杜浦數位安全

```
int **ptr;
int *(*ptr)[10];
int *(*ptr)(int *);
int *(*ptr[10])(int **);
int (*(*(*ptr(int *))[10]))(int *);
sizeof(int (******)[10][20][30][40]);
```

pointer to pointer to pointer to pointer to pointer to pointer to array of 10 array of 20 array of 30 array of 40 int

Anyway, it’s also a pointer.

# Array


Jyun-An Chen
11月16日上午4:54 ·
現在是凌晨 4 點 54 分
我來分享一個 C/C++ 冷知識
陣列 name[index] 的寫法可以寫成 index[name]
如圖片，我整個人都嚇傻了
5 int main()
6 {
7 int a[5] = { 10 , 20 , 30 , 40 , 50 };
8
9 cout << 3[a] << "\n"; // Output: 40
10 }
馬聖豪、蔡孟師和其他108人
41則留言 37次分享

# Array

TEAMT5 杜浦數位安全

### 6.5.2.1 Array subscripting

**Constraints**

1 One of the expressions shall have type "pointer to object *type*", the other expression shall have integer type, and the result has type "*type*".

**Semantics**

2 A postfix expression followed by an expression in square brackets `[]` is a subscripted designation of an element of an array object. The definition of the subscript operator `[]` is that `E1[E2]` is identical to `(*((E1)+(E2)))`. Because of the conversion rules that apply to the binary `+` operator, if `E1` is an array object (equivalently, a pointer to the initial element of an array object) and `E2` is an integer, `E1[E2]` designates the `E2`-th element of `E1` (counting from zero).

# Array

TEAMT5 杜浦數位安全

### 6.5.2.1 Array subscripting

**Constraints**

1 One of the expressions shall have type "pointer to object *type*", the other expression shall have integer type, and the result has type "*type*

**Semantics**

2 A postfix expression followed by an expression in square brackets [] is a subscripted designation of an element of an array object. The definition of the subscript operator `[]` is that `E1[E2]` is identical to `(*((E1)+(E2)))`. Because of the conversion rules that apply to the binary `+` operator, if `E1` is an array object (equivalently, a pointer to the initial element of an array object) and `E2` is an integer, `E1[E2]` designates the `E2`-th element of `E1` (counting from zero).

sizeof(E1) * E2

# Array

挑戰題：**++ptr[--ptr]**
Which one is sequenced before?

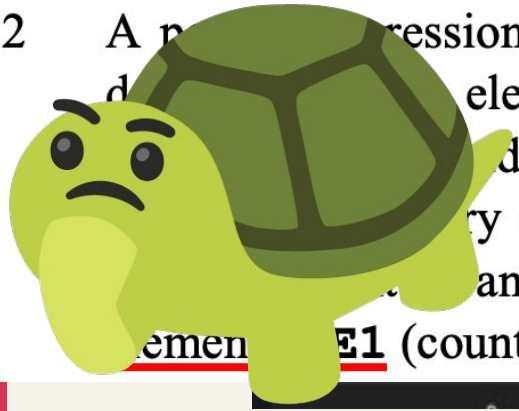
**6.5.2.1 Array**

**Constraints**

1 One of the exp … her expression shall have integer ty…

**Semantics**

2 A p…ression followed by an expression in square brackets … a subscripted d… element of an array object. The definition of the subsc… operator **[]** …dentical to **(*((E1)+(E2)))**. Because of the conversi… rules that …y **+** operator, if **E1** is an array object (equivalently, a pointer to the …an array object) and **E2** is an integer, **E1[E2]** designates the **E2**-th …emen… **E1** (counting from zero).



# Array

TEAMT5 杜浦數位安全

挑戰題：++ptr[--ptr]

**7.6.1.1 Subscripting** **[expr.sub]**

1 A postfix expression followed by an expression in square brackets is a postfix expression. One of the expressions shall be a glvalue of type "array of `T`" or a prvalue of type "pointer to `T`" and the other shall be a prvalue of unscoped enumeration or integral type. The result is of type "`T`". The type "`T`" shall be a completely-defined object type.[58] The expression `E1[E2]` is identical (by definition) to `*((E1)+(E2))`, except that in the case of an array operand, the result is an lvalue if that operand is an lvalue and an xvalue otherwise. The expression `E1` is sequenced before the expression `E2`.

UB

Well-defined after 17

# Indirect pointers

```
void
}
```

https://www.youtube.com/watch?v=o8NPllzkFhE

# Indirect pointers

TEAMT5
杜 浦 數 位 安 全

```
void remove_list_node(List *list, Node *target) {
    Node *prev = NULL;
    Node *current = list->head;
    // Walk the list
    while (current != target) {
        prev = current;
        current = current->next;
    }
// Remove the target by updating the head or the previous node.
    if (!prev)
        list->head = target->next;
    else
        prev->next = target->next;
}
```

# Indirect pointers

TEAMT5
杜浦數位安全

```
void remove_list_node(List *list, Node *target) {
    // The "indirect" pointer points to the *address*
    // of the thing we'll update.
    Node **indirect = &list->head;
    // Walk the list, looking for the thing that
    // points to the node we want to remove.
    while (*indirect != target)
        indirect = &(*indirect)->next;
    *indirect = target->next;
}
```

# callback and member function

TEAMT5 杜浦數位安全

```
int (*callback)(char *) = info_getter[ctz];
if (callback == NULL) {
    // It might impossible to reach here, but just in case
    pr_alert("Unsupported info: %d\n", ctz);
    return -EINTR;
}
int offset = callback(info);
```

```
// A dispatcher table for those callbacks
const static int (*info_getter[ALIGN(NUM)])(char *) = {
    f1,
    f2,
    f3,
    f4,
    f5,
    f6,
    NULL,
};
```

# callback and member function

TEAMT5 杜浦數位安全

```
const struct file_operations fops = {
    .owner = THIS_MODULE,
    .read = my_read,
    .write = my_write,
    .open = my_open,
    .release = my_release,
    .llseek = my_device_lseek,
};
```

```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
        ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
        //…
} __randomize_layout;
```

# NULL for compiler check?

TEAMT5 杜浦數位安全

```c
int foo(const int arr[static 1]) {}

int main(){
    foo(NULL);
    int a;
    foo(&a);
}
```

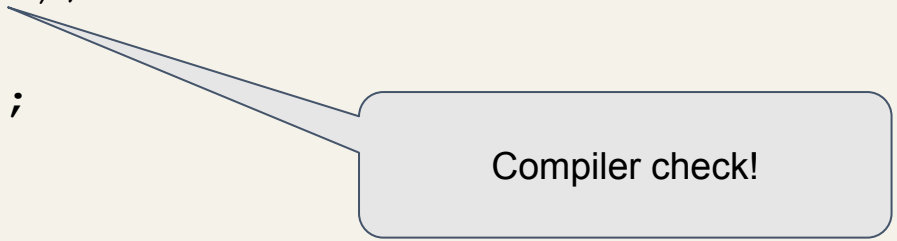
https://godbolt.org/z/196GTbcbE

# NULL for compiler check?

TEAMT5 杜浦數位安全

```
int foo(const int arr[static 1]) {}

int main(){
    foo(NULL);
    int a;
    foo(&a);
}
```




https://godbolt.org/z/196GTbcbE

# NULL for compiler check?

```c
int foo(const in

int main(){
    foo(NULL);
    int a;
    foo(&a);
}
```

C99: §6.7.5.3/7

A declaration of a parameter as "array of *type*" shall be adjusted to "qualified pointer to *type*", where the type qualifiers (if any) are those specified within the **[** and **]** of the array type derivation. If the keyword **static** also appears within the **[** and **]** of the array type derivation, then for each call to the function, the value of the corresponding actual argument shall provide access to the first element of an array with at least as many elements as specified by the size expression.

```
<source>: In function 'main':
<source>:8:5: warning: argument 1 null where non-null expected [-Wnonnull]
    8 |     foo(NULL);
      |     ^~~
<source>:3:5: note: in a call to function 'foo' declared 'nonnull'
    3 | int foo(const int arr[static 1]) {
      |     ^~~
```

https://godbolt.org/z/196GTbcbE

# The void pointer


This is the only one way to implement containers in CS101.

# The void pointer

TEAMT5
杜 浦 數 位 安 全

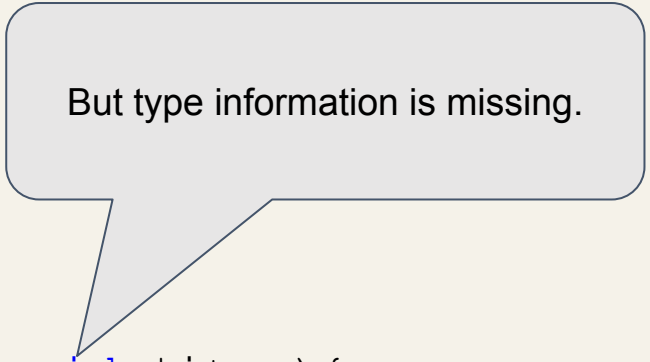
```
typedef struct vector {
    void *data;
    unsigned int size;
} vector_t;

void* push_back(vector_t* self, void *item){
    //...
}
```

# The void pointer

```
typedef struct vector {
    void *data;
    unsigned int size;
} vector_t;


void* push_back(vector_t* self, void *item){
    //...
}
```



# The void pointer

TEAMT5 杜浦數位安全

```
#define VECTOR_PUSH_BACK(v, item, type)                                    \
    do {                                                                   \
        if ((v).size >= (v).capacity) {                                    \
            (v).capacity *= 2;                                                 \
            void *temp = realloc((v).data, (v).capacity * (v).item_size); \
            if (temp == NULL) {                                            \
                fprintf(stderr, "Memory reallocation failed\n");               \
                exit(EXIT_FAILURE);                                        \
            }                                                              \
            (v).data = temp;                                               \
        }                                                                  \
        memcpy((char *)(v).data + ((v).size * (v).item_size), &item,
(v).item_size);\
        (v).size++;                                                        \
    } while (0)
```

https://godbolt.org/z/aY4EqTEhq

# The void pointer

```
#define VEC
    do {
        if (

        }
        memc
(v).item_si
        (v).
    } while
```

```
45  int main() {
46      vector_t myVector;
47      VECTOR_INIT(myVector, int); // Initialize vector for integers
48
49      int value1 = 10;
50      int value2 = 20;
51
52      VECTOR_PUSH_BACK(myVector, value1, int); // Push value1
53      VECTOR_PUSH_BACK(myVector, value2, int); // Push value2
54
55      // Access elements
56      int *arr = (int *)myVector.data;
57      for (size_t i = 0; i < myVector.size; ++i) {
58          printf("%d ", arr[i]);
59      }
60      printf("\n");
61
62      free(myVector.data); // Free allocated memory
63      return 0;
64  }
```

https://godbolt.

# The void pointer

```
#define VEC
    do {
        if (

        }
        memc
(v).item_si
        (v).
    } while
```

```
45  int main() {
46      vector_t myVector;
47      VECTOR_INIT(myVector, int); // Initialize vector for integers
48
49      int value1 = 10;
50      int value2 = 20;
51
52      VECTOR_PUSH_BACK(myVector, value1, int); // Push value1
53      VECTOR_PUSH_BACK(myVector, value2, int); // Push value2
54
55      // Access elements
56      int *arr = (int *)myVector.data;
57      for (size_t i = 0; i < myVector.size; ++i) {
58          printf("%d ", arr[i]);
59      }
60      printf("\n");
61
62      free(myVector.data); // Free allocated memory
63      return 0;
64  }
```

https://godbolt.

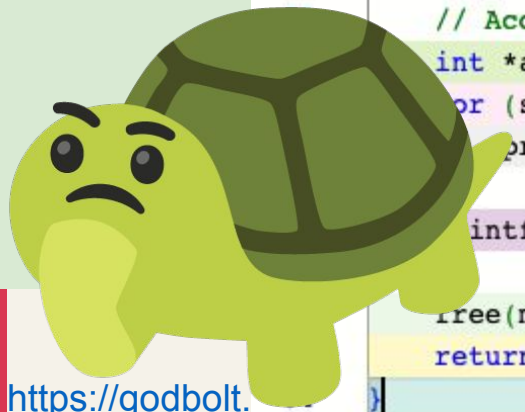
# The void pointer

TEAMT5 杜浦數位安全

```
#define VEC
    do {
        if (
```

```
45  int main() {
46      vector_t myVector;
47      VECTOR_INIT(myVector, int); // Initialize vector for integers
48
        ue1 = 10;
        e2 = 20;
        USH_BACK(myVector, value1, int); // Push value1
        USH_BACK(myVector, value2, int); // Push value2
54
        // Access elements
        int *arr = (int *)myVector.data;
        or (size_t i = 0; i < myVector.size; ++i) {
            printf("%d ", arr[i]);
        intf("\n");
        ree(myVector.data); // Free allocated memory
        return 0;
    }
```

How's about container in a container?



https://godbolt.

# The void pointer

```
#define VEC
    do {
        if (
```

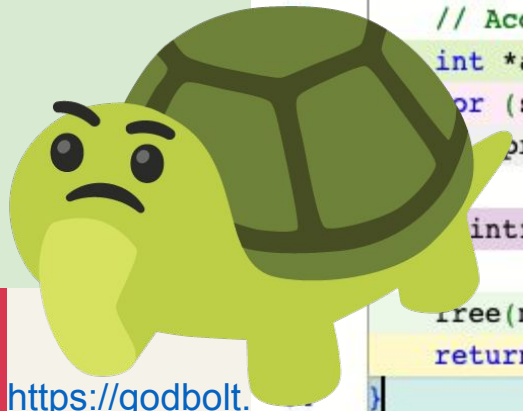
```
45  int main() {
46      vector_t myVector;
47      VECTOR_INIT(myVector, int); // Initialize vector for integers
48
        ue1 = 10;
        e2 = 20;

        USH_BACK(myVector, value1, int); // Push value1
        USH_BACK(myVector, value2, int); // Push value2
54
        // Acce
        int *ar
        or (si
        pri

        intf(

        ree(my ctor.data); // Free allocated memory
        return 0;
    }
```

How's about container in a container?



It's hard to implement: vector_t(vector_t(double), int)...

https://godbolt.

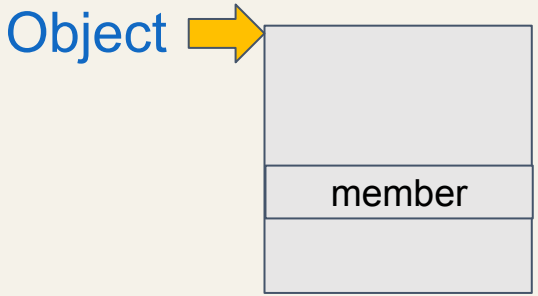
# container_of

TEAMT5
杜 浦 數 位 安 全

```
#define container_of(ptr, type, member)                                 \
    __extension__({                                                     \
        const __typeof__(((type *)0)->member) *(__pmember) = (ptr);     \
        (type *)((char *)__pmember - offsetof(type, member));           \
    })
```

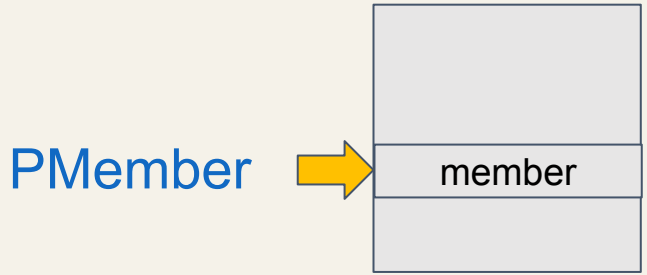
# container_of

```
#define container_of(ptr, Object, member)                                \
    __extension__({                                                      \
        const __typeof__(((Object *)0)->member) *(__pmember) = (ptr); \
        (type *)((char *)__pmember - offsetof(type, member));            \
    })
```



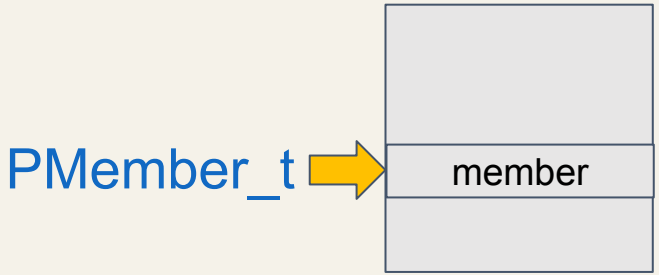
# container_of

```
#define container_of(ptr, type, member)                                 \
    __extension__({                                                     \
        const __typeof__(((type *)0)->member) *(__pmember) = (ptr);    \
        (type *)((char *)__pmember - offsetof(type, member));           \
    })
```



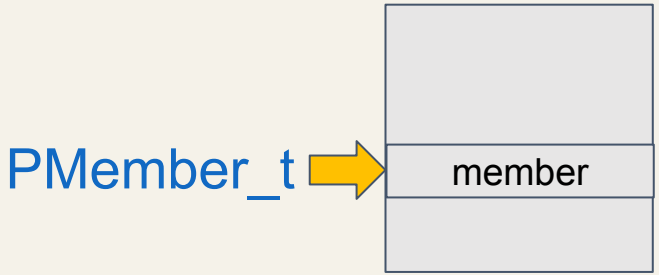
# container_of

```
#define container_of(ptr, type, member)                                 \
    __extension__({                                                     \
        const __typeof__(((type *)0)->member) *(__pmember) = (ptr);    \
        (type *)((char *)__pmember - offsetof(type, member));           \
    })
```



# container_of

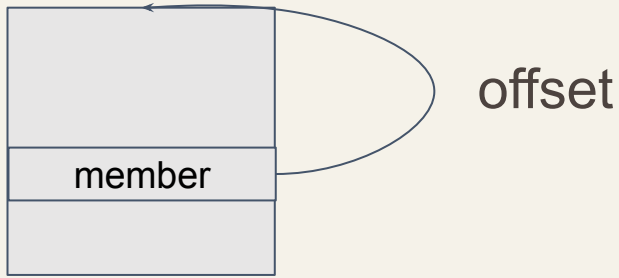
```
#define container_of(ptr, type, member)                                 \
    __extension__({                                                     \
        const __typeof__(((type *)0)->member) *(__pmember) = (ptr);    \
        (type *)((char *)__pmember - offsetof(type, member));           \
    })
```



# container_of

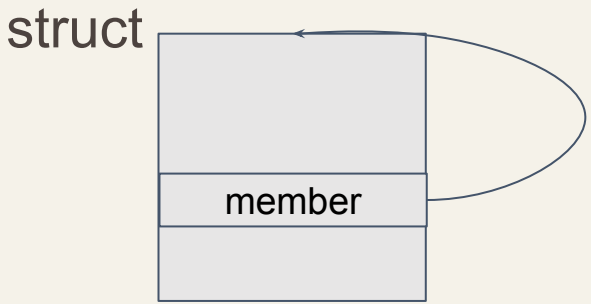
TEAMT5 杜浦數位安全

```
#define container_of(ptr, type, member)                                 \
    __extension__({                                                     \
        const __typeof__(((type *)0)->member) *(__pmember) = (ptr);     \
        (type *)((char *)__pmember - offsetof(type, member));           \
    })
```



# container_of

```
#define container_of(ptr, type, member)                                 \
    __extension__({                                                     \
        const __typeof__(((type *)0)->member) *(__pmember) = (ptr);    \
        (type *)((char *)__pmember - offsetof(type, member));           \
    })
```

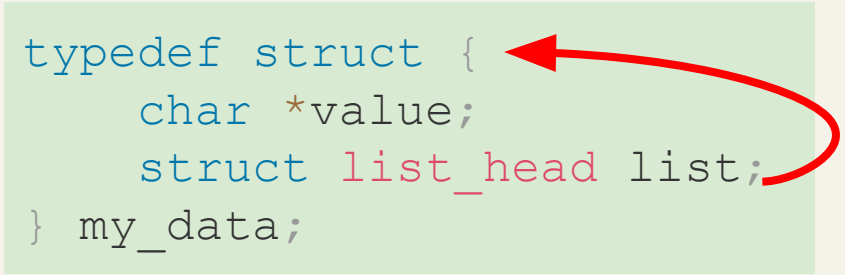


# container_of

TEAMT5 杜浦數位安全

```
#define container_of(ptr, type, member)                                 \
    __extension__({                                                     \
        const __typeof__(((type *)0)->member) *(__pmember) = (ptr);     \
        (type *)((char *)__pmember - offsetof(type, member));           \
    })
```

```
typedef struct {
    char *value;
    struct list_head list;
} my_data;
```



# container_of

```
#define container_of(ptr, type, member)                                 \
    __extension__({                                                     \
        const __typeof__(((type *)0)->member) *(__pmember) = (ptr);    \
        (type *)((char *)__pmember - offsetof(type, member));           \
    })
```

# container_of

```
    list_head_t *l = new_my_data(42);
    const int me = get_value(l);
    printf("%d\n", me);

    list_head_t l2 = *l;
    const int me2 = get_value(&l2);
    printf("%d\n", me2);
```

```
Program returned: 0
42
-1947187831
```

https://godbolt.org/z/K5PrK8nfK

# container_of

```
    list_head_t *l = new_my_data(42);
    const int me = get_value(l);
    printf("%d\n", me);

    list_head_t l2 = *l;
    const int me2 = get_value(&l2);
    printf("%d\n", me2);
```

Local variable.

```
Program returned: 0
   42
   -1947187831
```

https://godbolt.org/z/K5PrK8nfK

# container_of

```
    list_head_t *l = new_my_data(42);
    const int me = get_value(l);
    printf("%d\n", me);

    list_head_t l2 = *l;
    const int me2 = get_value(&l2);
    printf("%d\n", me2);
```

Get offset from stack.

```
Program returned: 0
   42
   -1947187831
```

https://godbolt.org/z/K5PrK8nfK

# Extra:

TEAMT5
杜 浦 數 位 安 全

Is container_of well-defined?

https://hackmd.io/@Mes/indirect_through_null_pointer

# restrict qualifier


- Not in C++
- For pointer optimizations
- since C99

# restrict qualifier

TEAMT5
杜 浦 數 位 安 全

```
void foo(int **a, int **b) {
    *b = NULL;
    *a = NULL;
}
```

# restrict qualifier


```
void foo(int **a, int **b) {
    *b = NULL;
    *a = NULL;
}
```

# restrict qualifier

What if a == b?

```
void foo(int **a, int **b) {
    *b = NULL;
    *a = NULL;
}
```

# restrict qualifier

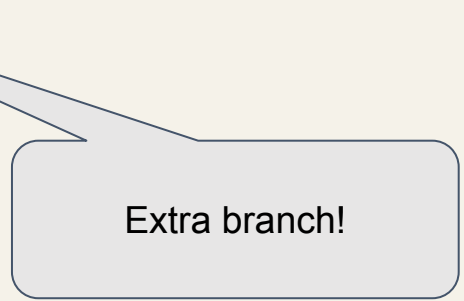
What if **a** == **b**?

```
void foo(int **a, int **b) {
    *b = NULL;
    *a = NULL;
}
```

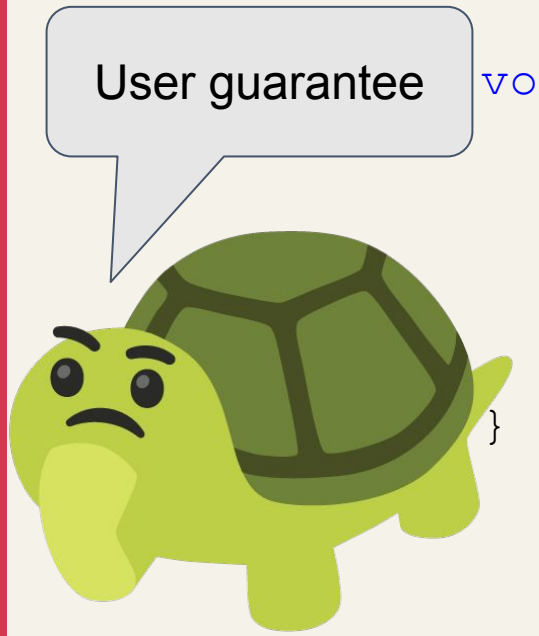
Memory Access Violation

# restrict qualifier

```
void foo(int **a, int **b) {
    *b = NULL;
    if (b != a)
        *a = NULL;
}
```



# restrict qualifier

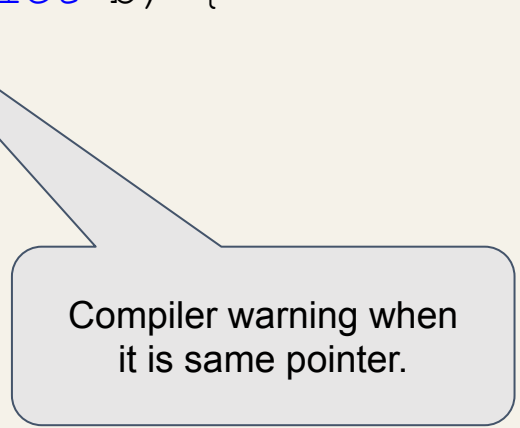



```
void foo(int **a, int **b) {
    *b = NULL;
    if (b != a)
        *a = NULL;
}
```

Extra branch!

# restrict qualifier

```
void foo(int **restrict a,
         int **restrict b) {
    *b = NULL;
    *a = NULL;
}
```



# The C++ world

# Ownership

TEAMT5
杜 浦 數 位 安 全

RAII

# Ownership

TEAMT5
杜 浦 數 位 安 全

**R**esource
**A**cquisition
**I**s
**I**nitialization

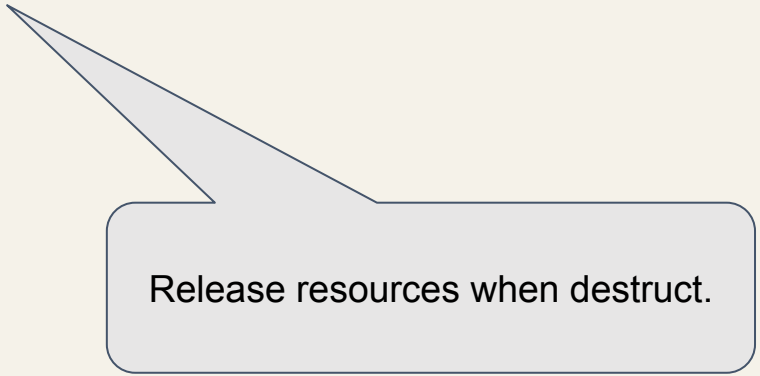
# Ownership

TEAMT5
杜 浦 數 位 安 全

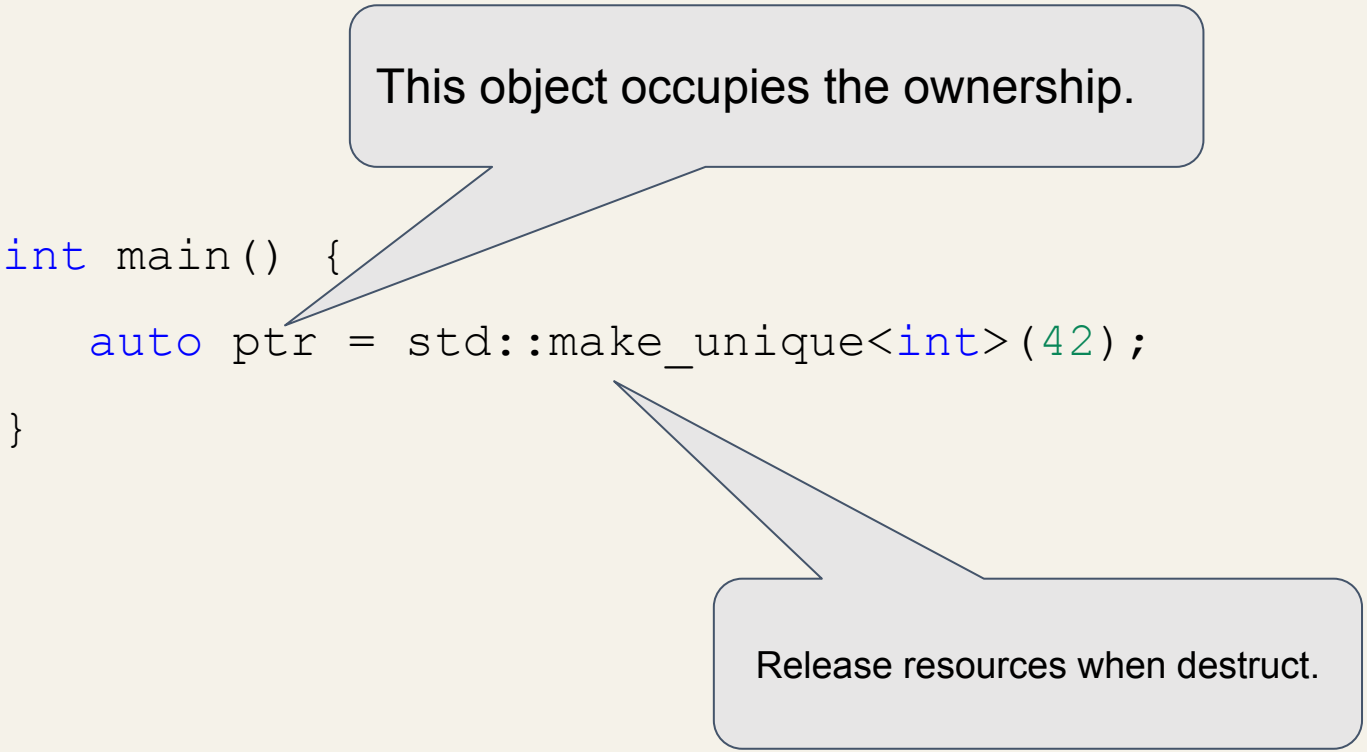
```
int main() {
    auto ptr = std::make_unique<int>(42);
}
```

# Ownership

```
int main() {
    auto ptr = std::make_unique<int>(42);
}
```



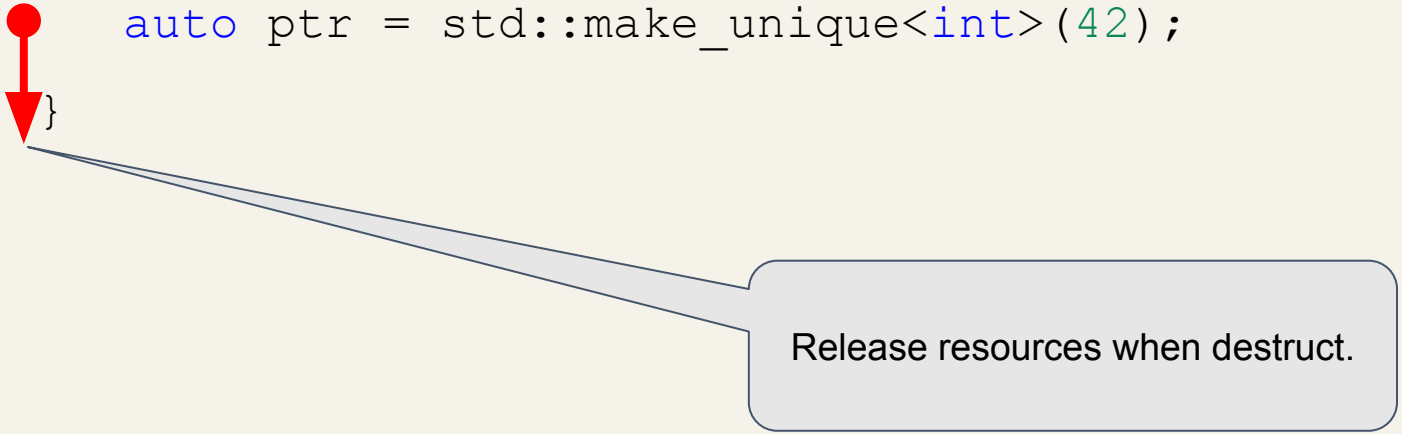
# Ownership


This object occupies the ownership.
int main() {
auto ptr = std::make_unique<int>(42);
}
Release resources when destruct.

# Object lifetime

TEAMT5 杜浦數位安全

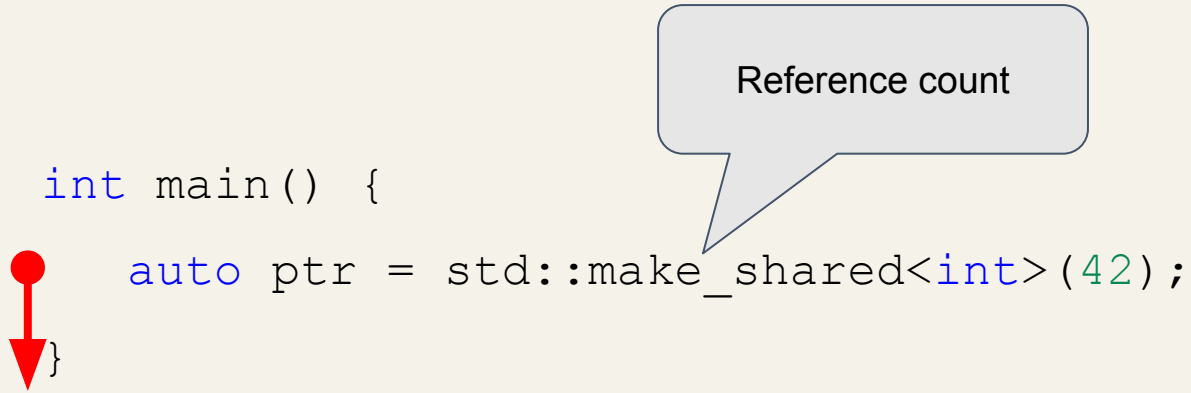
```
int main() {
    auto ptr = std::make_unique<int>(42);
}
```



# Object lifetime

```
int main() {
    auto ptr = std::make_shared<int>(42);
}
```

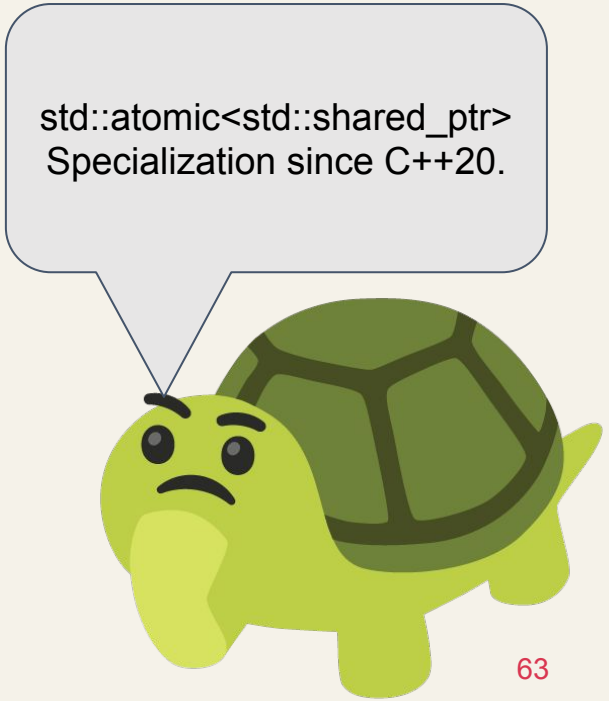
# Object lifetime



# Object lifetime

TEAMT5
杜 浦 數 位 安 全

```
int main() {
    auto ptr = std::make_shared<int>(42);
}
```



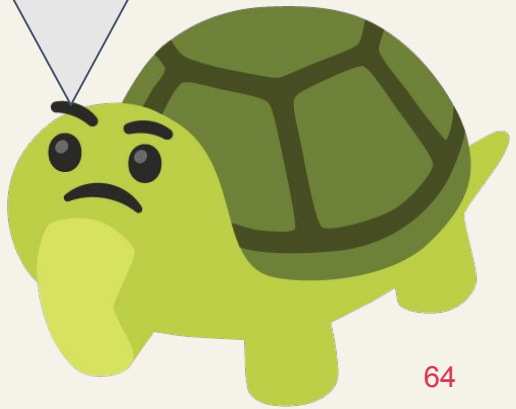
# Object lifetime

TEAMT5 杜浦數位安全

```
int main() {
    auto ptr = std::make_shared<int>(42);
}
```

std::atomic<std::shared_ptr>
Specialization since C++20.

It's atomic operation
for control block.
But not for user data.



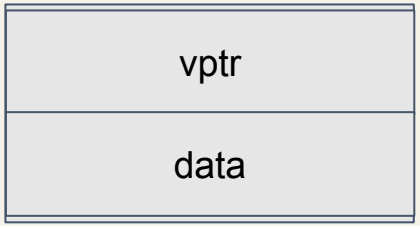
# Virtual functions

TEAMT5
杜 浦 數 位 安 全

```
struct Obj {
    void normal() {}
    virtual void foo() {}
    virtual ~Obj = default;
    int data;
};
```

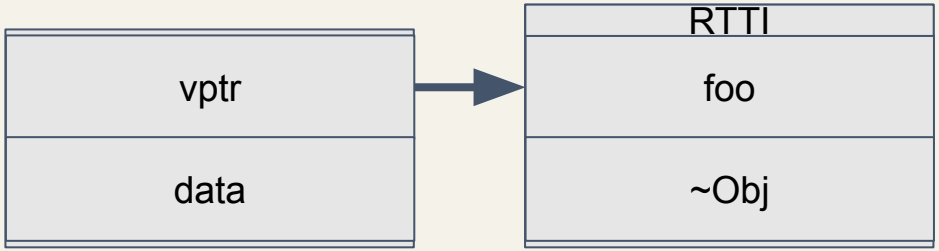
# Virtual functions

```
struct Obj {
    void normal() {}
    virtual void foo() {}
    virtual ~Obj = default;
    int data;
};
```



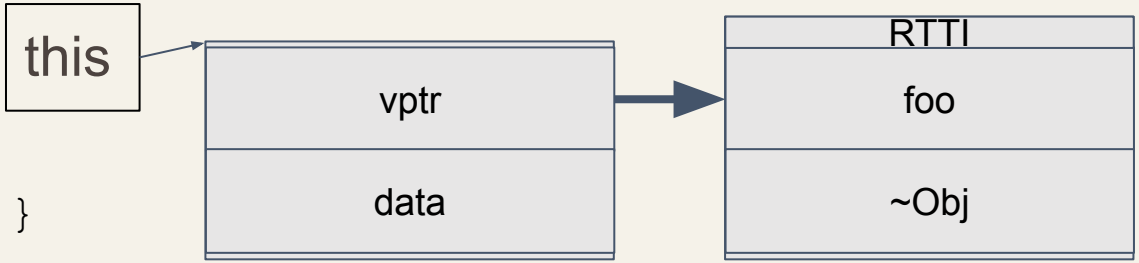
# Virtual functions

```
struct Obj {
    void normal() {}
    virtual void foo() {}
    virtual ~Obj = default;
    int data;
};
```



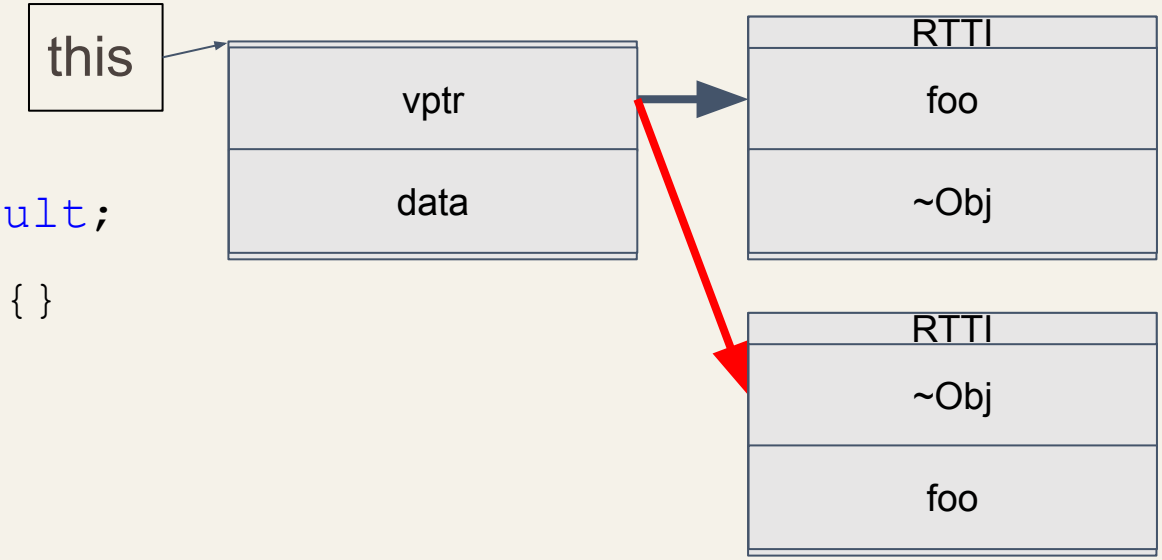
# Virtual functions

```
struct Obj {
    void normal() {}
    virtual void foo() {}
    virtual ~Obj = default;
    int data;
};
```



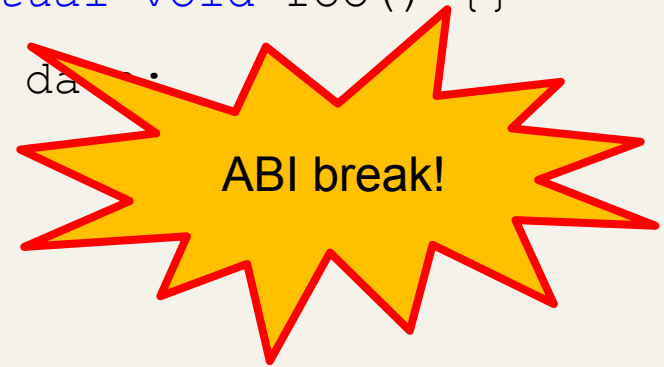
# Virtual functions

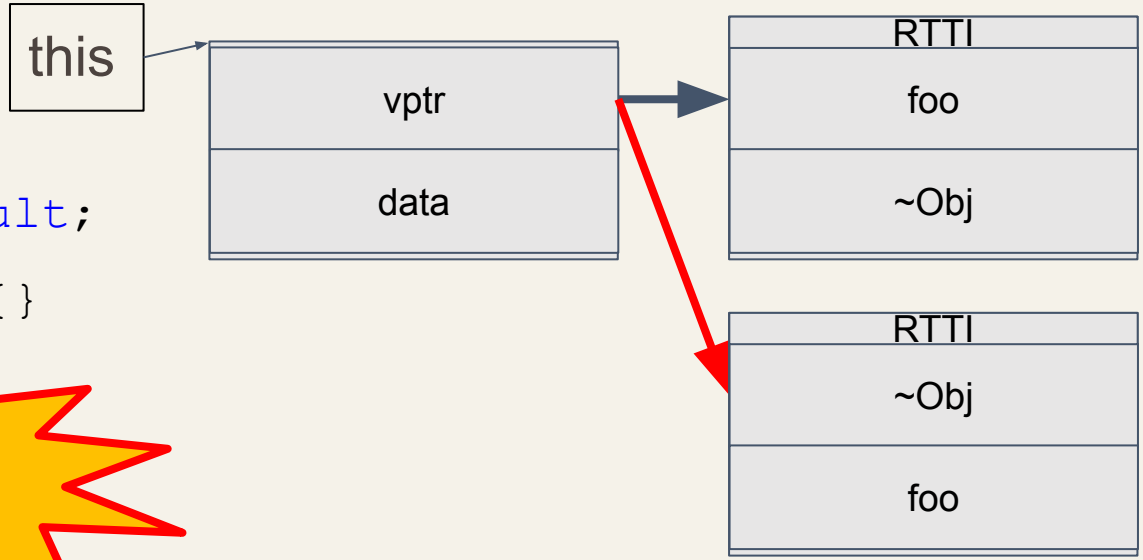
```
struct Obj {
    void normal() {}
    virtual ~Obj = default;
    virtual void foo() {}
    int data;
};
```



# Virtual functions

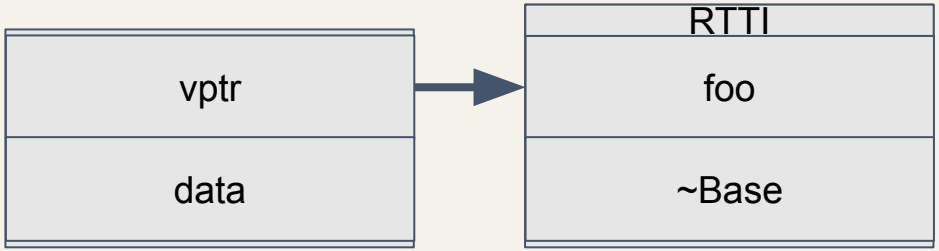
```
struct Obj {
    void normal() {}
    virtual ~Obj = default;
    virtual void foo() {}
    int da
};
```



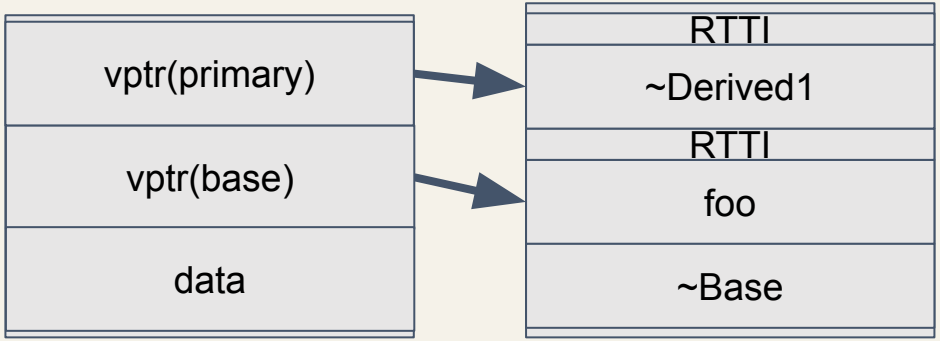
# Virtual functions

```
struct Base {
    virtual void foo() {}
    virtual ~Base() = default;
    int data;
};
```



# Virtual functions

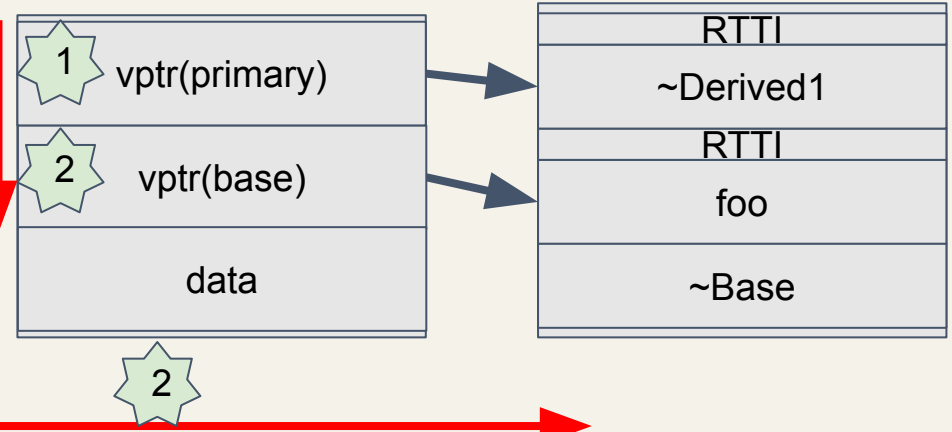
```
struct Base {
    virtual void foo() {}
    virtual ~Base() = default;
    int data;
};
struct Derived1 : virtual public Base { ~Derived1() = default; };
struct Derived2 : virtual public Base { ~Derived2() = default; };
```




https://aesophor.github.io/2023/05/19/C-Polymorphism-VTables-and-Game-Hacking/

# Virtual functions

```
struct Base {
    virtual void foo() {}
    virtual ~Base() = default;
    int data;
};
```

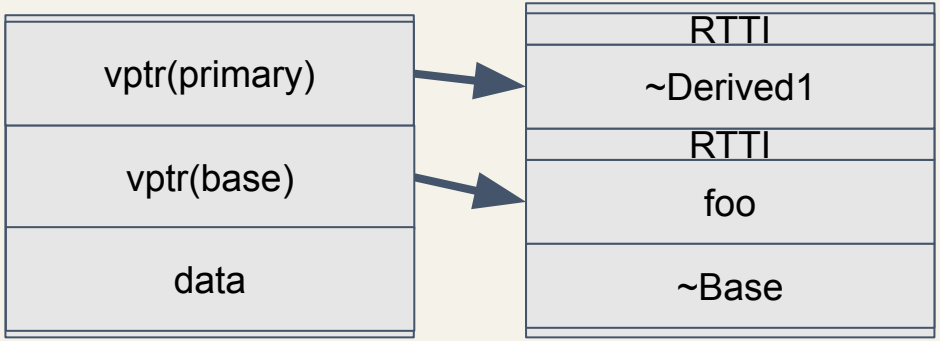


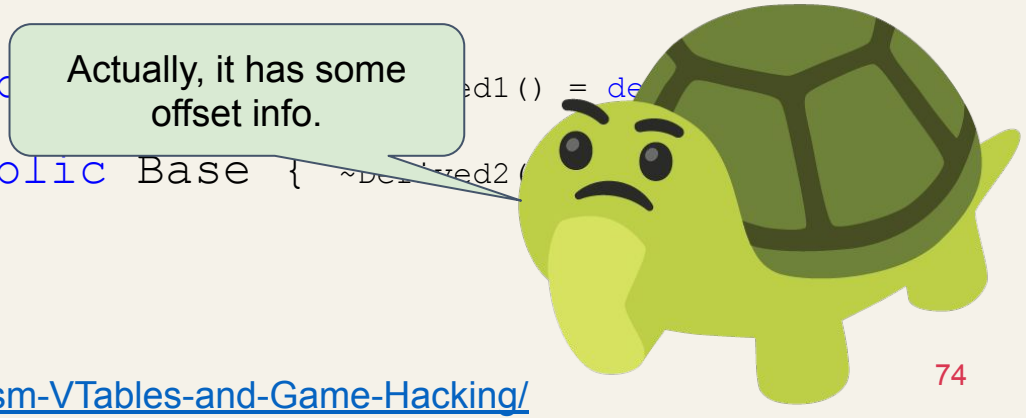

```
struct Derived1 : virtual public Base { ~Derived1() = default; };
struct Derived2 : virtual public Base { ~Derived2() = default; };
```

https://aesophor.github.io/2023/05/19/C-Polymorphism-VTables-and-Game-Hacking/

# Virtual functions

```
struct Base {
    virtual void foo() {}
    virtual ~Base() = default;
    int data;
};
struct Derived1 : virtual pub
struct Derived2 : virtual public Base {
```



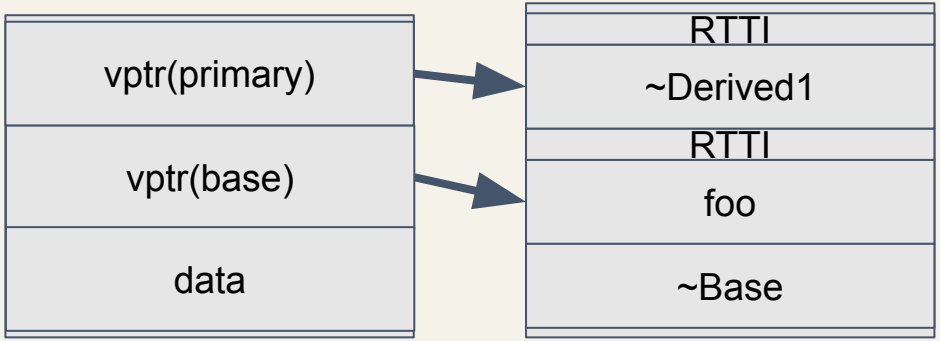




https://godbolt.org/z/375e9fPnW

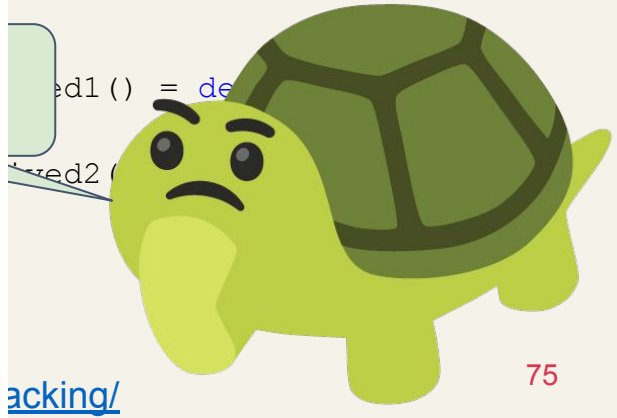
https://aesophor.github.io/2023/05/19/C-Polymorphism-VTables-and-Game-Hacking/

# Virtual functions

```
struct Base {
    virtual void foo() {}
    virtual ~Base() = default;
    int data;
```



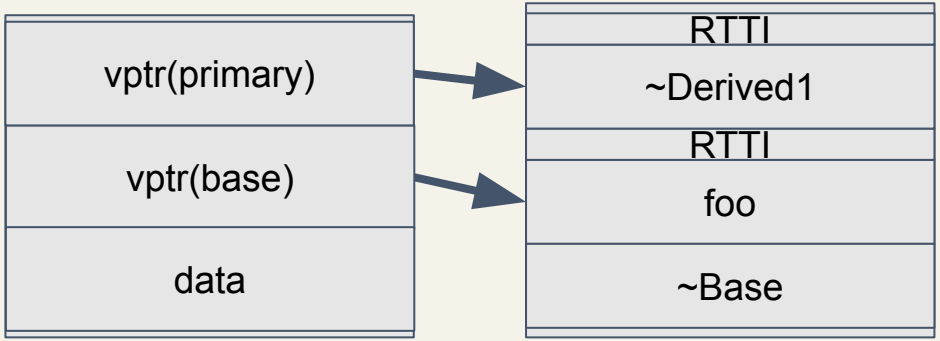

```
444 vtable for Derived1:
445         .quad   8
446         .quad   0
447         .quad   typeinfo for Derived1
448         .quad   Derived1::~Derived1() [complete object destructor]
449         .quad   Derived1::~Derived1() [deleting destructor]
450         .quad   -8
451         .quad   0
452         .quad   -8
453         .quad   typeinfo for Derived1
454         .quad   Base::foo()
455         .quad   virtual thunk to Derived1::~Derived1() [complete object destructor]
456         .quad   virtual thunk to Derived1::~Derived1() [deleting destructor]
```



acking/

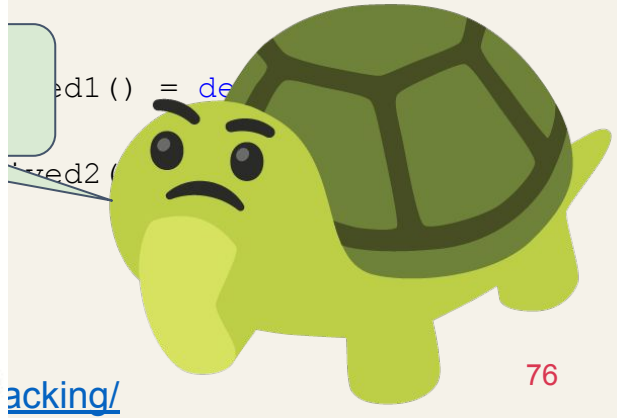
# Virtual functions

```
struct Base {
    virtual void foo() {}
    virtual ~Base() = default;
    int data;
```




```
444 vtable for Derived1:
445         .quad   8
446         .quad   0
447         .quad   typeinfo for Derived1
448         .quad   Derived1::~Derived1() [complete object destructor]
449         .quad   Derived1::~Derived1() [deleting destructor]
450         .quad   -8
451         .quad   0
452         .quad   -8
453         .quad   typeinfo for Derived1
454         .quad   Base::foo()
455         .quad   virtual thunk to Derived1::~Derived1() [complete object destructor]
456         .quad   virtual thunk to Derived1::~Derived1() [deleting destructor]
```

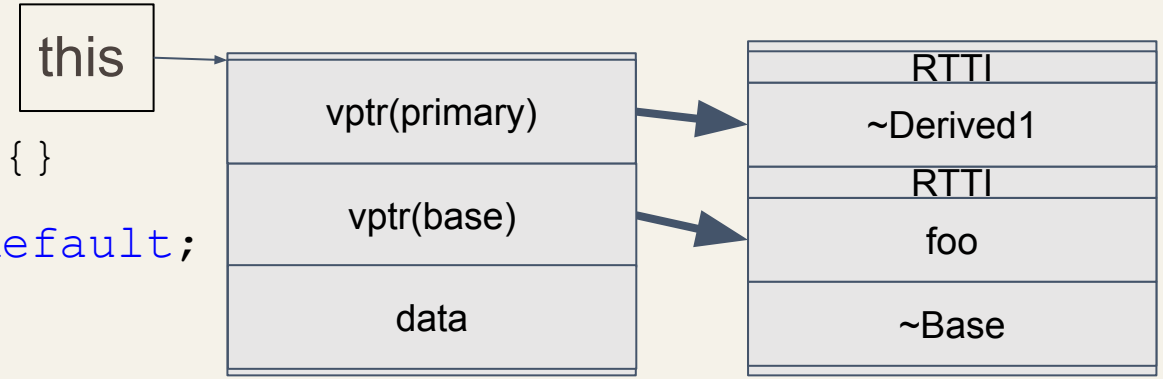
Offset to Base



acking/

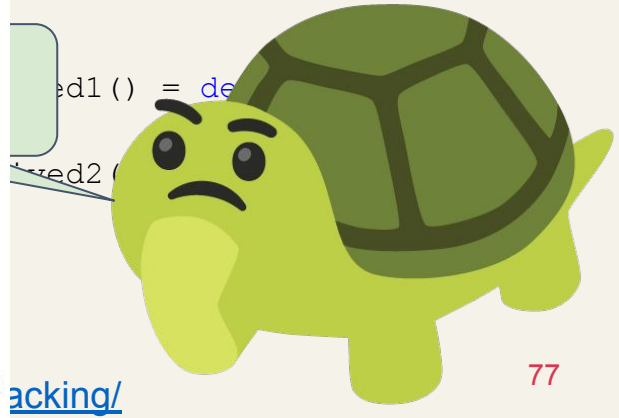
# Virtual functions

```
struct Base {
    virtual void foo() {}
    virtual ~Base() = default;
    int data;
```




```
444 vtable for Derived1:
445         .quad   8
446         .quad   0
447         .quad   typeinfo for Derived1
448         .quad   Derived1::~Derived1() [complete object destructor]
449         .quad   Derived1::~Derived1() [deleting destructor]
450         .quad   -8
451         .quad   0
452         .quad   -8
453         .quad   typeinfo for Derived1
454         .quad   Base::foo()
455         .quad   virtual thunk to Derived1::~Derived1() [complete object destructor]
456         .quad   virtual thunk to Derived1::~Derived1() [deleting destructor]
```

Offset to Base

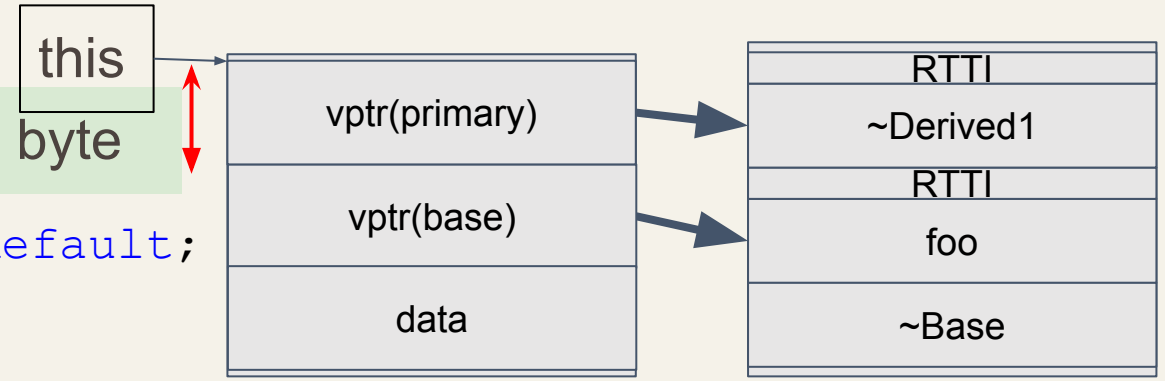


acking/

# Virtual functions

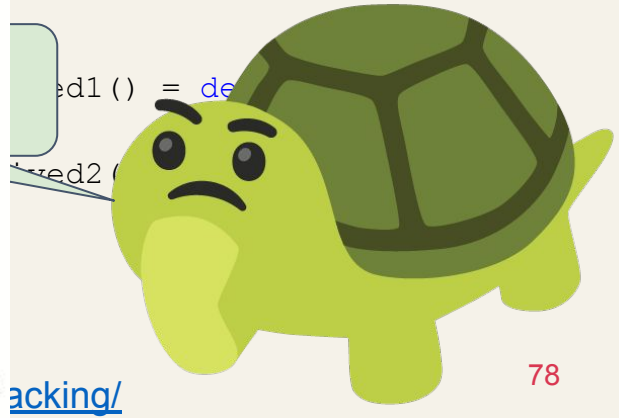
```
struct Base {
    virtual voi
    virtual ~Base() = default;
    int data;
```

64-bit, 8 byte




```
444 vtable for Derived1:
445         .quad   8
446         .quad   0
447         .quad   typeinfo for Derived1
448         .quad   Derived1::~Derived1() [complete object destructor]
449         .quad   Derived1::~Derived1() [deleting destructor]
450         .quad   -8
451         .quad   0
452         .quad   -8
453         .quad   typeinfo for Derived1
454         .quad   Base::foo()
455         .quad   virtual thunk to Derived1::~Derived1() [complete object destructor]
456         .quad   virtual thunk to Derived1::~Derived1() [deleting destructor]
```

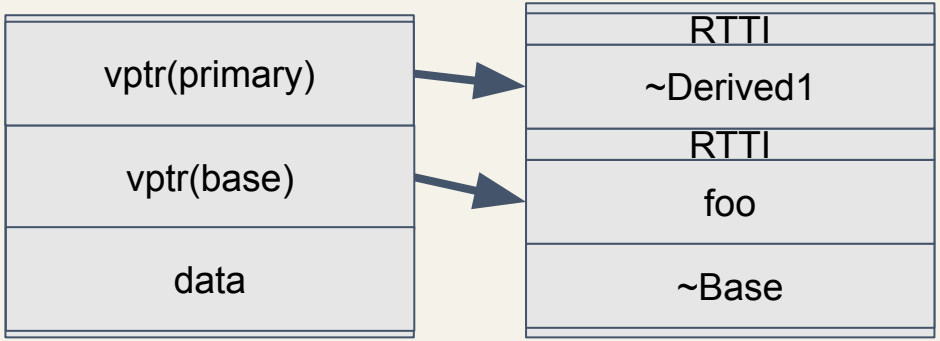
Offset to Base



acking/

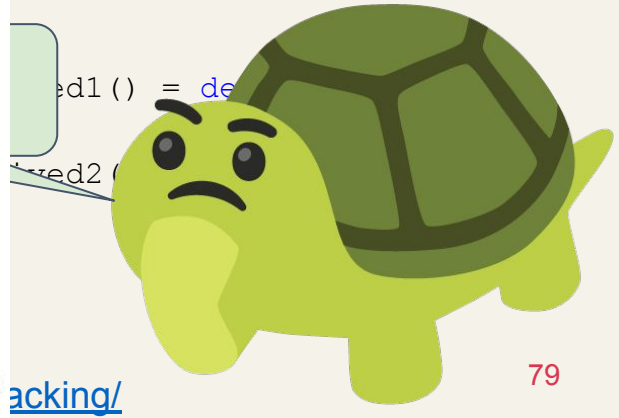
# Virtual functions

```
struct Base {
    virtual void foo() {}
    virtual ~Base() = default;
    int data;
```




```
444 vtable for Derived1:
445         .quad   8
446         .quad   0
447         .quad   typeinfo for Derived1
448         .quad   Derived1::~Derived1() [complete object destructor]
449         .quad   Derived1::~Derived1() [deleting destructor]
450         .quad   -8
451         .quad   0
452         .quad   -8
453         .quad   typeinfo for Derived1
454         .quad   Base::foo()
455         .quad   virtual thunk to Derived1::~Derived1() [complete object destructor]
456         .quad   virtual thunk to Derived1::~Derived1() [deleting destructor]
```

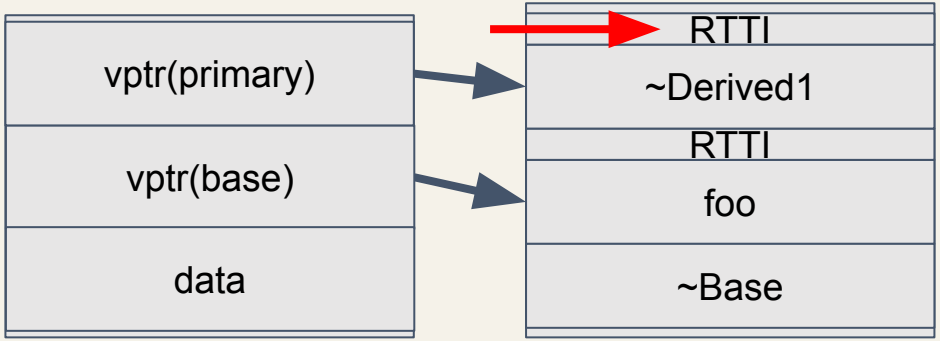
Offset to top



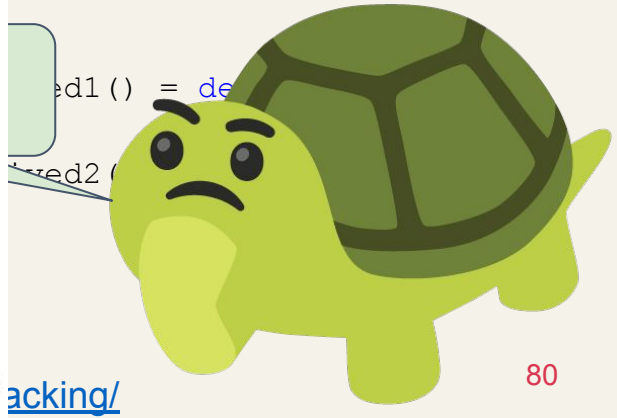
acking/

# Virtual functions

```
struct Base {
    virtual void foo() {}
    virtual ~Base() = default;
    int data;
```



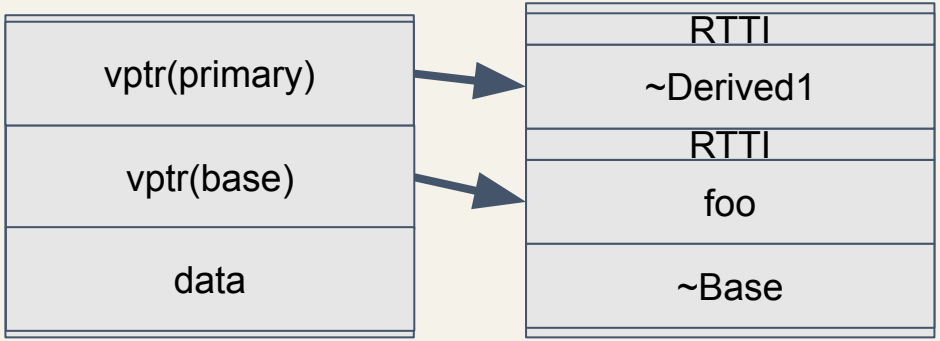

```
444 vtable for Derived1:
445         .quad   8
446         .quad   0
447         .quad   typeinfo for Derived1
448         .quad   Derived1::~Derived1() [complete object destructor]
449         .quad   Derived1::~Derived1() [deleting destructor]
450         .quad   -8
451         .quad   0
452         .quad   -8
453         .quad   typeinfo for Derived1
454         .quad   Base::foo()
455         .quad   virtual thunk to Derived1::~Derived1() [complete object destructor]
456         .quad   virtual thunk to Derived1::~Derived1() [deleting destructor]
```



acking/

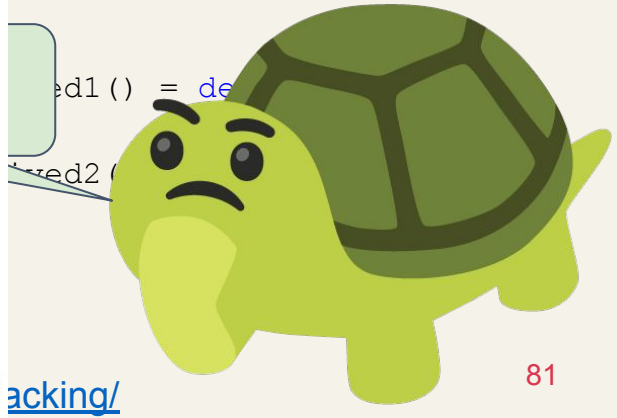
# Virtual functions

```
struct Base {
    virtual void foo() {}
    virtual ~Base() = default;
    int data;
```




```
444 vtable for Derived1:
445         .quad   8
446         .quad   0
447         .quad   typeinfo for Derived1
448         .quad   Derived1::~Derived1() [complete object destructor]
449         .quad   Derived1::~Derived1() [deleting destructor]
450         .quad   -8
451         .quad   0
452         .quad   -8
453         .quad   typeinfo for Derived1
454         .quad   Base::foo()
455         .quad   virtual thunk to Derived1::~Derived1() [complete object destructor]
456         .quad   virtual thunk to Derived1::~Derived1() [deleting destructor]
```

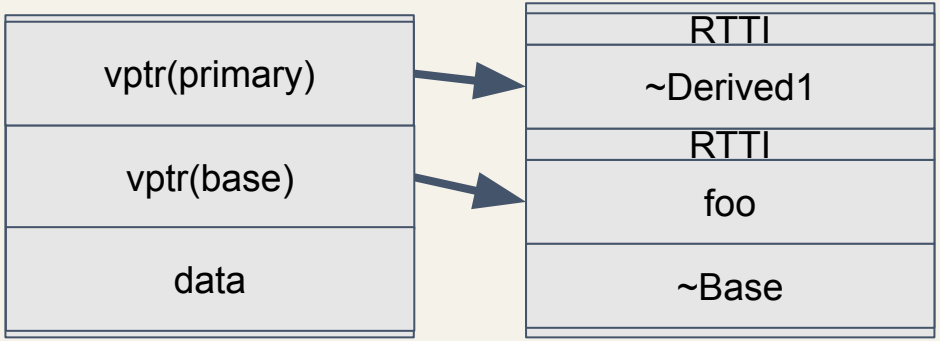
Real dtor



acking/

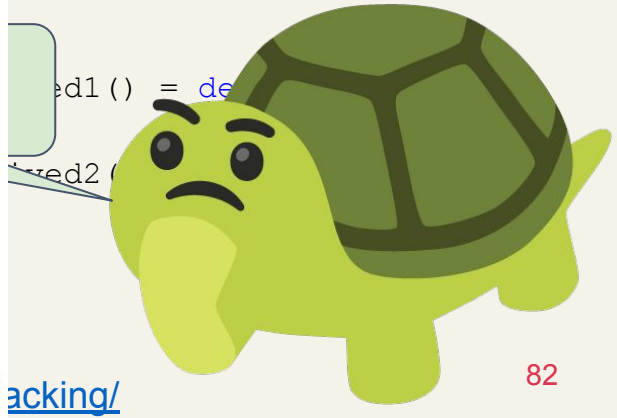
# Virtual functions

```
struct Base {
    virtual void foo() {}
    virtual ~Base() = default;
    int data;
```




```
444 vtable for Derived1:
445         .quad   8
446         .quad   0
447         .quad   typeinfo for Derived1
448         .quad   Derived1::~Derived1() [complete object destructor]
449         .quad   Derived1::~Derived1() [deleting destructor]
450         .quad   -8
451         .quad   0
452         .quad   -8
453         .quad   typeinfo for Derived1
454         .quad   Base::foo()
455         .quad   virtual thunk to Derived1::~Derived1() [complete object destructor]
456         .quad   virtual thunk to Derived1::~Derived1() [deleting destructor]
```

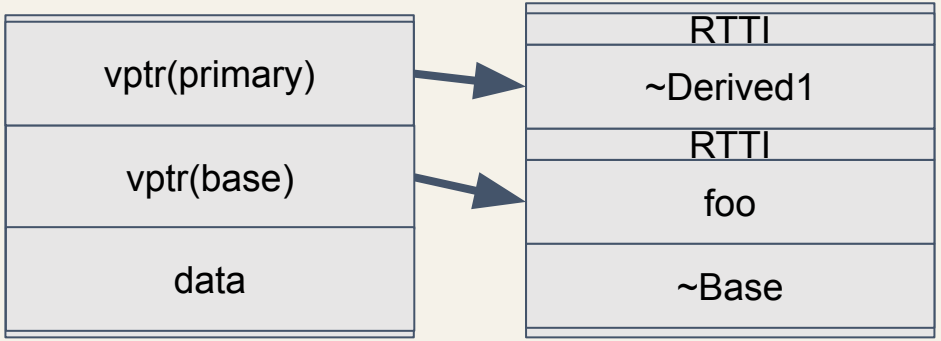
Dtor after dtor



acking/

# Virtual functions

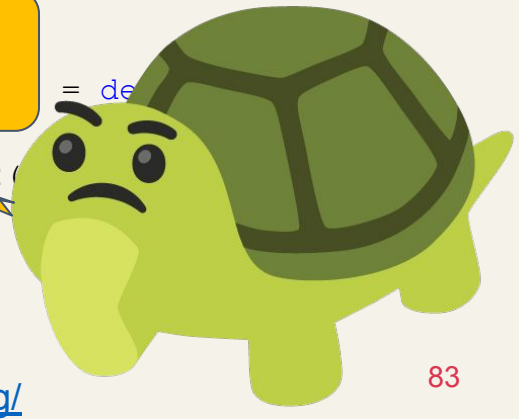
```
struct Base {
    virtual void foo() {}
    virtual ~Base() = default;
    int data;
```




```
444 vtable for Derived1:
445         .quad   8
446         .quad   0
447         .quad   typeinfo for Derived1
448         .quad   Derived1::~Derived1() [complete object destructor]
449         .quad   Derived1::~Derived1() [deleting destructor]
450         .quad   -8
451         .quad   0
452         .quad   -8
453         .quad   typeinfo for Derived1
454         .quad   Base::foo()
455         .quad   virtual thunk to Derived1::~Derived1() [complete object destructor]
456         .quad   virtual thunk to Derived1::~Derived1() [deleting destructor]
```

Yes, it's data raceable.

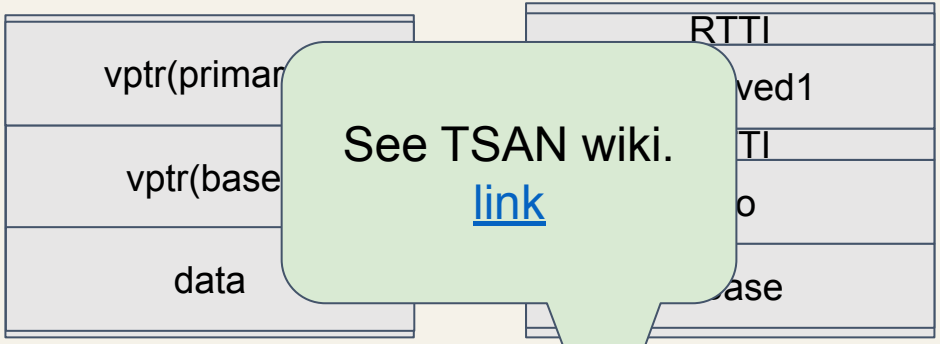
Dtor after dtor



acking/

# Virtual functions

```
struct Base {
    virtual void foo() {}
    virtual ~Base() = default;
    int data;
```

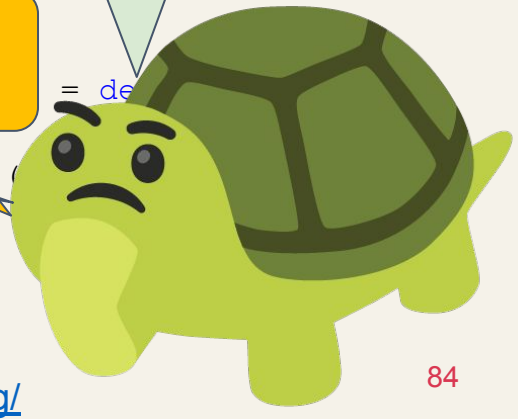



See TSAN wiki.
link

```
444 vtable for Derived1:
445         .quad   8
446         .quad   0
447         .quad   typeinfo for Derived1
448         .quad   Derived1::~Derived1() [complete object destructor]
449         .quad   Derived1::~Derived1() [deleting destructor]
450         .quad   -8
451         .quad   0
452         .quad   -8
453         .quad   typeinfo for Derived1
454         .quad   Base::foo()
455         .quad   virtual thunk to Derived1::~Derived1() [complete object destructor]
456         .quad   virtual thunk to Derived1::~Derived1() [deleting destructor]
```

Yes, it's data raceable.

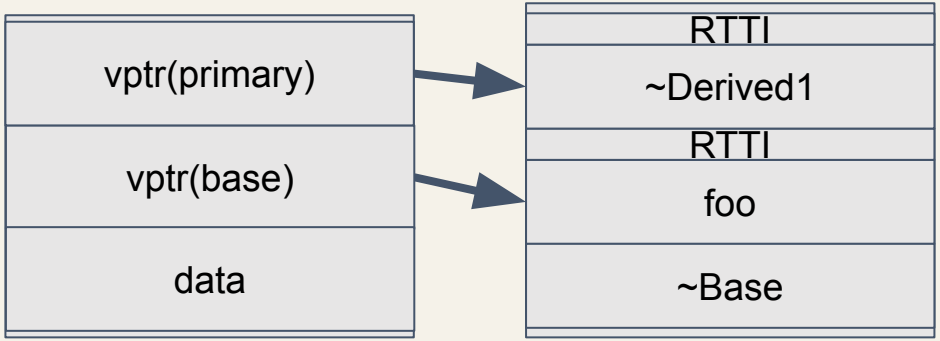
Dtor after dtor



acking/

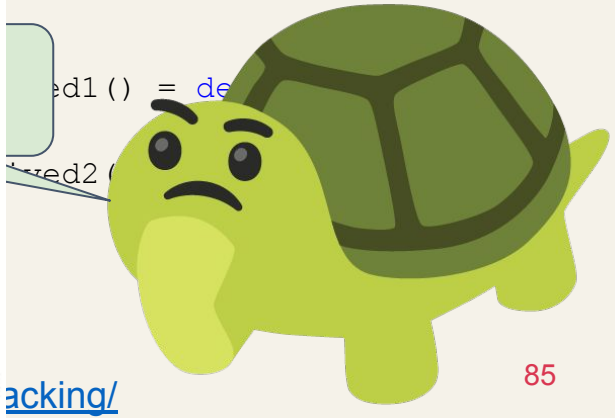
# Virtual functions

```
struct Base {
    virtual void foo() {}
    virtual ~Base() = default;
    int data;
```




```
444 vtable for Derived1:
445         .quad   8
446         .quad   0
447         .quad   typeinfo for Derived1
448         .quad   Derived1::~Derived1() [complete object destructor]
449         .quad   Derived1::~Derived1() [deleting destructor]
450         .quad   -8
451         .quad   0
452         .quad   -8
453         .quad   typeinfo for Derived1
454         .quad   Base::foo()
455         .quad   virtual thunk to Derived1::~Derived1() [complete object destructor]
456         .quad   virtual thunk to Derived1::~Derived1() [deleting destructor]
```

For dynamic_cast…



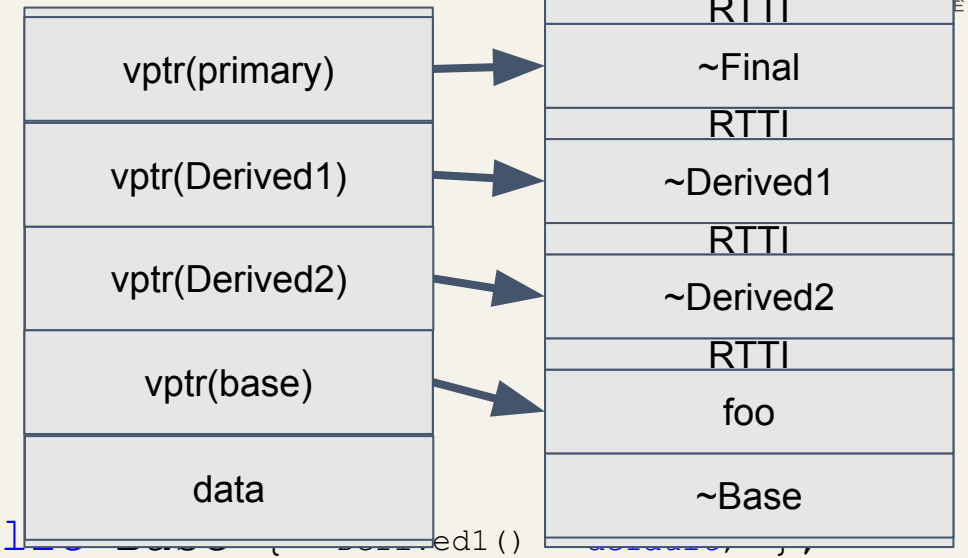
acking/

# Virtual functions

TEAMT5 杜浦數位安全

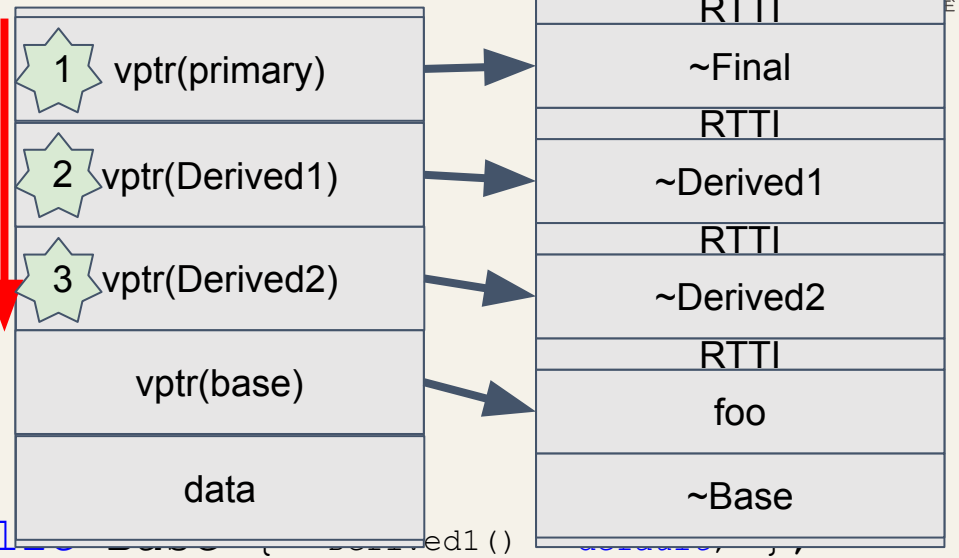
```
struct Base {
    virtual void foo() {}
    virtual ~Base() = default;
    int data;
};
struct Derived1 : virtual public Base { ~Derived1() = default; };
struct Derived2 : virtual public Base { ~Derived2() = default; };
struct Final    : public Derived1, public Derived2 {
~Final() = default; };
```

# Virtual functions




```
struct Base {
    virtual void foo() {}
    virtual ~Base() = default;
    int data;
};
struct Derived1 : virtual public Base { ~Derived1() = default; };
struct Derived2 : virtual public Base { ~Derived2() = default; };
struct Final    : public Derived1, public Derived2 {
~Final() = default; };
```

# Virtual functions



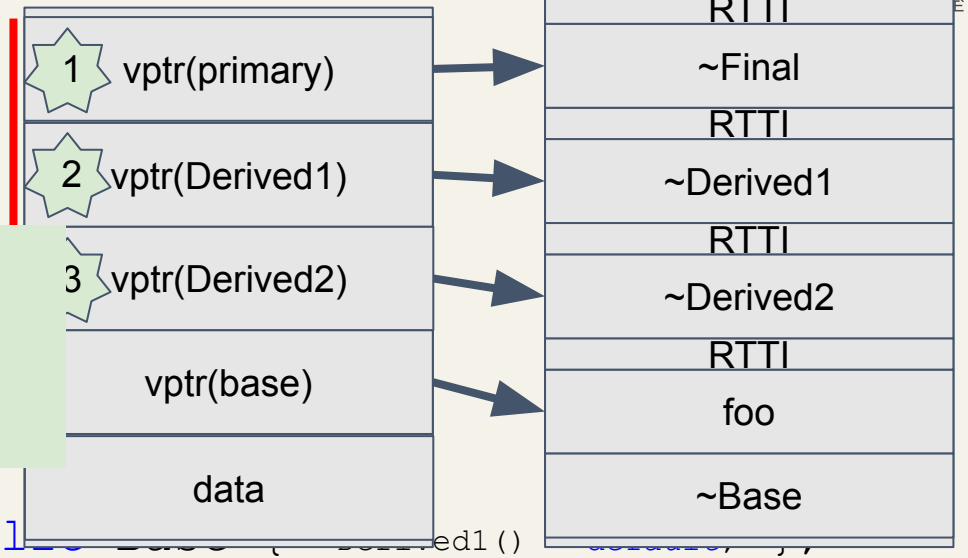

```
struct Base {
    virtual void foo() {}
    virtual ~Base() = default;
    int data;
};
struct Derived1 : virtual public Base { ~Derived1() = default; };
struct Derived2 : virtual public Base { ~Derived2() = default; };
struct Final    : public Derived1, public Derived2 {
~Final() = default; };
```

1 2 3

# Virtual functions

```
struct Base {
    virtual void foo() {}
    virtual ~
    int data;
};
struct Derived1 : virtual public Base { ~Derived1() = default; };
struct Derived2 : virtual public Base { ~Derived2() = default; };
struct Final    : public Derived1, public Derived2 {
~Final() = default; };
```

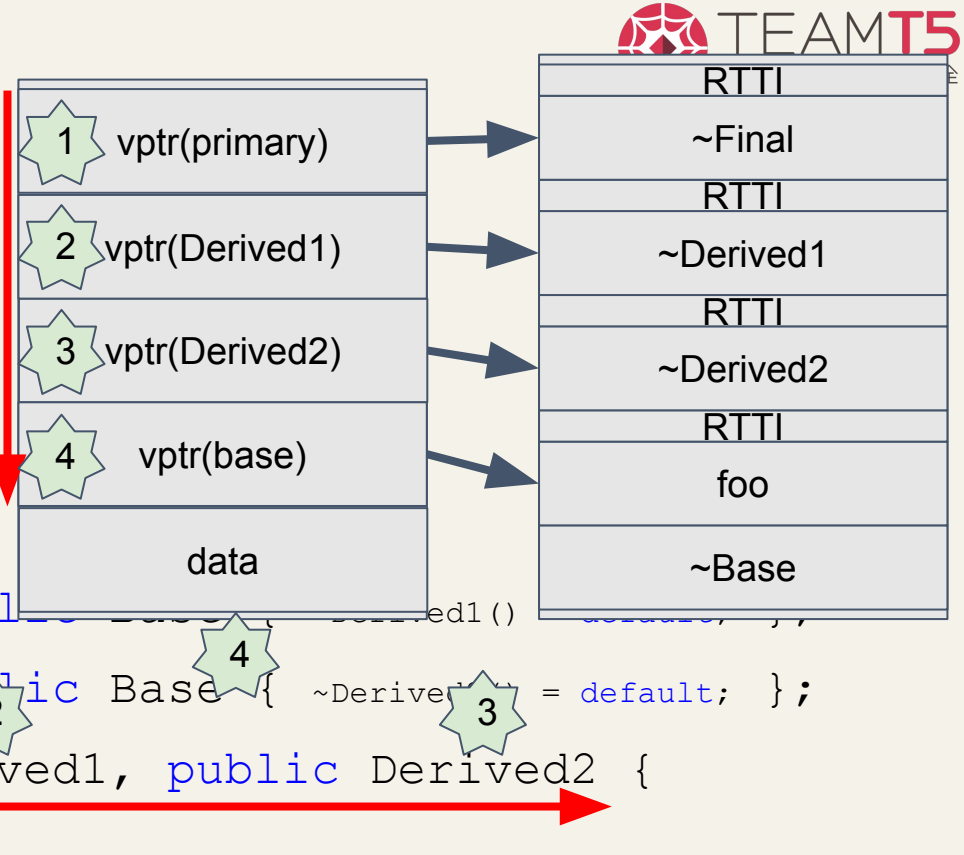
Virtual inheritance would share same type of data.



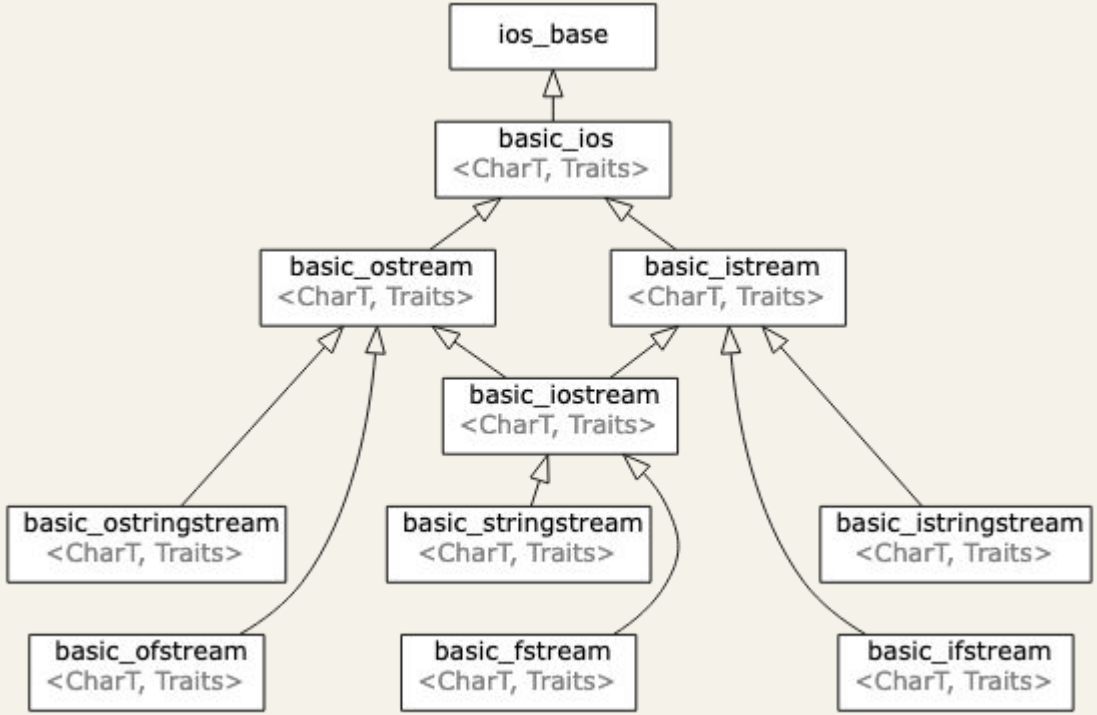

1 2 3

# Virtual functions

```
struct Base {
    virtual void foo() {}
    virtual ~Base() = default;
    int data;
};
struct Derived1 : virtual public Base { ~Derived1() = default; };
struct Derived2 : virtual public Base { ~Derived2() = default; };
struct Final    : public Derived1, public Derived2 {
~Final() = default; };
```



# Diamond of death




https://en.cppreference.com/w/cpp/io

# Virtual functions

```
struct Obj {
    void normal() {}
    virtual void foo() {}
    virtual ~Obj = default;
    int data;
};
```


https://devblogs.microsoft.com/oldnewthing/20200618-00/?p=103874
https://gcc.gnu.org/wiki/VerboseDiagnostics#delete-non-virtual-dtor

# Virtual functions

UB!

```
struct Obj {
    void normal() {}
    virtual void foo() {}
    virtual ~Obj = default;
    int data;
};
```

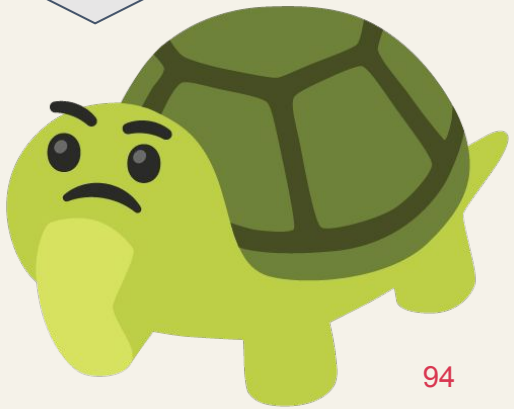
What if no dtor?

https://devblogs.microsoft.com/oldnewthing/20200618-00/?p=103874
https://gcc.gnu.org/wiki/VerboseDiagnostics#delete-non-virtual-dtor

# Virtual functions


```
struct Obj {
    void normal() {}
    virtual void foo() {}
    virtual ~Obj = default;
    int data;
};
```

What if no dtor?



[§ 5.3.5 p5](#) If the object being deleted has incomplete class type at the point of deletion and the complete class has a non-trivial destructor or a deallocation function, the behavior is undefined.

https://devblogs.microsoft.com/oldnewthing/20200618-00/?p=103874

https://gcc.gnu.org/wiki/VerboseDiagnostics#delete-non-virtual-dtor

# Virtual functions

```
struct Obj {
    void normal() {}
    virtual void foo() {}
    ~Obj = default;
    int data;
};
```


https://devblogs.microsoft.com/oldnewthing/20200618-00/?p=103874
https://gcc.gnu.org/wiki/VerboseDiagnostics#delete-non-virtual-dtor

# Virtual functions


```
struct Obj {
    void normal() {}
    virtual void foo() {}
    ~Obj = default;
    int data;
};
```


§ 5.3.5 p5 If the object being deleted has incomplete class type at the point of deletion and the complete class has a non-trivial destructor or a deallocation function, the behavior is undefined. https://godbolt.org/z/Gq6P5TWK3

https://devblogs.microsoft.com/oldnewthing/20200618-00/?p=103874

https://gcc.gnu.org/wiki/VerboseDiagnostics#delete-non-virtual-dtor

# Calling based virtual

TEAMT5 杜浦數位安全

```
struct Base {
    void goo() {puts(__PRETTY_FUNCTION__);}
    virtual void foo() {puts(__PRETTY_FUNCTION__);}
    void use_thread() {
        std::thread t(&Base::foo, this);
        t.join();
    }
    virtual ~Base() = default;
};

struct Derived : public Base {
    void goo() {puts(__PRETTY_FUNCTION__);}
    virtual void foo() override {puts(__PRETTY_FUNCTION__);}
    virtual ~Derived() = default;
};
```

# Calling based virtual

TEAMT5 杜浦數位安全

```
struct Base {
    void goo() {puts(__PRETTY_FUNCTION__);}
    virtual void foo() {puts(__PRETTY_FUNCTION__);}
    void use_thread() {
        std::thread t(&Base::foo, this);
        t.join();
    }
    virtual ~Base() = default;
};

struct Derived : public Base {
    void goo() {puts(__PRETTY_FUNCTION__);}
    virtual void foo() override {puts(__PRETTY_FUNCTION__);}
    virtual ~Derived() = default;
};
```

```
int main() {
    Derived d;
    d.use_thread();
}
```

# Calling based virtual

TEAMT5 杜浦數位安全

```
struct Base {
    void goo() {puts(__PRETTY_FUNCTION__);}
    virtual void foo() {puts(__PRETTY_FUNCTION__);}
    void use_thread() {
        std::thread t(&Base::foo, this);
        t.join();
    }
    virtual ~Base() = default;
};

struct Derived : public Base {
    void goo() {puts(__PRETTY_FUNCTION__);}
    virtual void foo() override {puts(__PRETTY_FUNCTION__);}
    virtual ~Derived() = default;
};
```

```
int main() {
    Derived d;
    d.use_thread();
}
```

# Calling based virtual

TEAMT5 杜浦數位安全

```
struct Base {
    void goo() {puts(__PRETTY_FUNCTION__);}
    virtual void foo() {puts(__PRETTY_FUNCTION__);}
    void use_thread() {
        std::thread t(&Base::foo, this);
        t.join();
    }
    virtual ~Base() = default;
};

struct Derived : public Base {
    void goo() {puts(__PRETTY_FUNCTION__);}
    virtual void foo() override {puts(__PRETTY_FUNCTION__);}
    virtual ~Derived() = default;
};
```

```
int main() {
    Derived d;
    d.use_thread();
}
```

# Calling based virtual

TEAMT5 杜浦數位安全

```
struct Base {
    void goo() {puts(__PRETTY_FUNCTION__);}
    virtual void foo() {puts(__PRETTY_FUNCTION__);}
    void use_thread() {
        std::thread t(&Base::foo, this);
        t.join();
    }
    virtual ~Base() = default;
};

struct Derived : public Base {
    void goo() {puts(__PRETTY_FUNCTION__);}
    virtual void foo() override {puts(__PRETTY_FUNCTION__);}
    virtual ~Derived() = default;
};
```

```
int main() {
    Derived d;
    d.use_thread();
}
```

# Calling based virtual

```
struct Base {
    void goo() {puts(__PRETTY_FUNCTION__);}
    virtual void foo() {puts(__PRETTY_FUNCTION__);}
    void use_thread() {
        std::thread t(&Base::foo, this);
        t.join();
    }
    virtual ~Base() = default;
};

struct Derived : public Base {
    void goo() {puts(__PRETTY_FUNCTION__)
    virtual void foo() override {puts(__PRETTY_FUNCTION__);}
    virtual ~Derived() = default;
};
```

```
int main() {
    Derived d;
    d.use_thread();
```

```
Program returned: 0
    virtual void Derived::foo()
```

# Calling based virtual

```
struct Base {
    void goo() {puts(__PRETTY_FUNCTION
    virtual void foo() {puts(__PRETTY_
    void use_thread() {
        std::thread t(&Base::foo, this
        t.join();
    }
    virtual ~Base() = default;
};

struct Derived : public Base {
    void goo() {puts(__PRETTY_FUNCTION__)
    virtual void foo() override {puts(__PRETTY_F
    virtual ~Derived() = default;
};
```

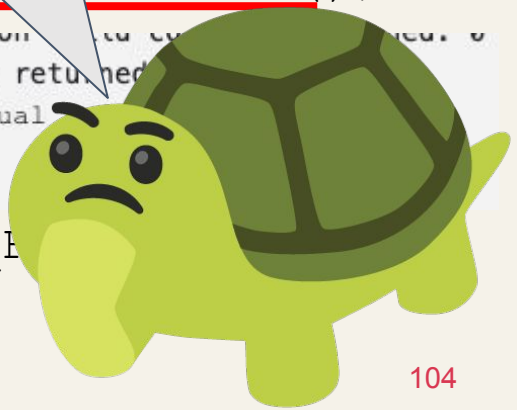
§11.7.3 p10 [Note 3: The interpretation of the call of a virtual function depends on the type of the object for which it is called (the dynamic type), whereas the interpretation of a call of a non-virtual member function depends only on the type of the pointer or reference denoting that object (the static type) ([expr.call]). — end note]

# Calling based virtual

```
struct Base {
    void goo() {puts(__PRETTY_FUNCTION
    virtual void foo() {puts(__PRETTY_
    void use_thread() {
        std::thread t(&Base::foo, this
        t.join();
    }
    virtual ~Base() = default;
};

struct Derived : public Base {
    void goo() {puts(__PRETTY_FUNCTION__)
    virtual void foo() override {puts(__PRETTY_F
    virtual ~Derived() = default;
};
```

§11.7.3 p10 [Note 3: The interpretation of the call of a virtual function depends on the type of the object for which it is called (the dynamic type), whereas the interpretation of a call of a non-virtual member function depends only on the type of the pointer or reference denoting that object (the static type) ([expr.call]). — end note]



# Disallow heap allocation

```
struct O {
void *operator new(size_t)
    = delete;
void operator delete(void*)
    = delete;
};

int main() {
   auto op =
    std::make_unique<O>();
}
```

```
std::make_unique(_Args&& ...) [with _Tp = O; _Args = {}; __detail::__unique_ptr_t<_Tp> = __detail::__unique_ptr_t<O>]':
<source>:11:34:   required from here
   11 |     auto op = std::make_unique<O>();
      |               ~~~~~~~~~~~~~~~~~~^~
/opt/compiler-explorer/gcc-trunk-20231201/include/c++/14.0.0/bits/unique_ptr.h:1076:30: error: use of deleted function 'static void* O::operator
new(size_t)'
 1076 |     { return unique_ptr<_Tp>(new _Tp(std::forward<_Args>(__args)...)); }
      |                              ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
<source>:6:11: note: declared here
    6 |     void *operator new(size_t) = delete;
      |           ^~~~~~~~
/opt/compiler-explorer/gcc-trunk-20231201/include/c++/14.0.0/bits/unique_ptr.h: In instantiation of 'void std::default_delete<_Tp>::operator()
(_Tp*) const [with _Tp = O]':
/opt/compiler-explorer/gcc-trunk-20231201/include/c++/14.0.0/bits/unique_ptr.h:398:17:   required from 'std::unique_ptr<_Tp, _Dp>::~unique_ptr()
[with _Tp = O; _Dp = std::default_delete<O>]'
<source>:6:11: note:   398 |           get_deleter()(std::move(__ptr));
<source>:6:11: note:       |           ~~~~~~~~~~~~^~~~~~~~~~~~~~~~~~~
<source>:11:34:   required from here
<source>:6:11: note:    11 |     auto op = std::make_unique<O>();
<source>:6:11: note:       |               ~~~~~~~~~~~~~~~~~~^~
/opt/compiler-explorer/gcc-trunk-20231201/include/c++/14.0.0/bits/unique_ptr.h:93:9: error: use of deleted function 'static void O::operator
delete(void*)'
   93 |         delete __ptr;
      |         ^~~~~~~~~~~~
<source>:7:10: note: declared here
    7 |     void operator delete(void *)= delete;
      |          ^~~~~~~~
Compiler returned: 1
```

https://godbolt.org/z/7K1TK85dY

# Disallow heap allocation

```
struct O {
void *operator new(size_t)
    = delete;
void operator delete(void*)
    = delete;
};

int main() {
   auto op =
    std::make_unique<O>();
}
```

```
std::make_unique(_Args&& ...) [with _Tp = O; _Args = {}; __detail::__unique_ptr_t<_Tp> = __detail::__unique_ptr_t<O>]':
<source>:11:34:   required from here
   11 |     auto op = std::make_unique<O>();
      |               ~~~~~~~~~~~~~~~~~~^~
/opt/compiler-explorer/gcc-trunk-20231201/include/c++/14.0.0/bits/unique_ptr.h:1076:30: error: use of deleted function 'static void* O::operator
new(size_t)'
 1076 |     { return unique_ptr<_Tp>(new _Tp(std::forward<_Args>(__args)...)); }
      |                              ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
<source>:6:11: note: declared here
    6 |     void *operator new(size_t) = delete;
      |           ^~~~~~~~
/opt/compiler-explorer/gcc-trunk-20231201/include/c++/14.0.0/bits/unique_ptr.h: In instantiation of 'void std::default_delete<_Tp>::operator()
(_Tp*) const [with _Tp = O]':
/opt/compiler-explorer/gcc-trunk-20231201/include/c++/14.0.0/bits/unique_ptr.h:398:17:   required from 'std::unique_ptr<_Tp, _Dp>::~unique_ptr()
[with _Tp = O; _Dp = std::default_delete<O>]'
<source>:6:11: note:   398 |           get_deleter()(std::move(__ptr));
<source>:6:11: note:       |           ~~~~~~~~~~~~^~~~~~~~~~~~~~~~~~~
<source>:11:34:   required from here
<source>:6:11: note:    11 |     auto op = std::make_unique<O>();
<source>:6:11: note:       |               ~~~~~~~~~~~~~~~~~~^~
/opt/compiler-explorer/gcc-trunk-20231201/include/c++/14.0.0/bits/unique_ptr.h:93:9: error: use of deleted function 'static void O::operator
delete(void*)'
   93 |         delete __ptr;
      |         ^~~~~~~~~~~~
<source>:7:10: note: declared here
    7 |     void operator delete(void *)= delete;
      |          ^~~~~~~~
Compiler returned: 1
```

Overload the new/delete operator.

https://godbolt.org/z/7K1TK85dY

# Disallow heap allocation

```
struct O {
void *operator new(size_t)
    = delete;
void operator delete(void*)
    = delete;
};

int main() {
   auto op =
    std::make_unique<O>();
}
```

```
std::make_unique(_Args&& ...) [with _Tp = O; _Args = {}; __detail::__unique_ptr_t<_Tp> = __detail::__unique_ptr_t<O>]':
<source>:11:34:   required from here
   11 |     auto op = std::make_unique<O>();
      |               ~~~~~~~~~~~~~~~~~~^~
/opt/compiler-explorer/gcc-trunk-20231201/include/c++/14.0.0/bits/unique_ptr.h:1076:30: error: use of deleted function 'static void* O::operator
new(size_t)'
 1076 |     { return unique_ptr<_Tp>(new _Tp(std::forward<_Args>(__args)...)); }
      |                              ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
<source>:6:11: note: declared here
    6 |     void *operator new(size_t) = delete;
      |           ^~~~~~~~
/opt/compiler-explorer/gcc-trunk-20231201/include/c++/14.0.0/bits/unique_ptr.h: In instantiation of 'void std::default_delete<_Tp>::operator()
(_Tp*) const [with _Tp = O]':
/opt/compiler-explorer/gcc-trunk-20231201/include/c++/14.0.0/bits/unique_ptr.h:398:17:   required from 'std::unique_ptr<_Tp, _Dp>::~unique_ptr()
[with _Tp = O; _Dp = std::default_delete<O>]'
<source>:6:11: note:   398 |           get_deleter()(std::move(__ptr));
<source>:6:11: note:       |           ~~~~~~~~~~~~^~~~~~~~~~~~~~~~~~~
<source>:11:34:   required from here
<source>:6:11: note:    11 |     auto op = std::make_unique<O>();
<source>:6:11: note:       |               ~~~~~~~~~~~~~~~~~~^~
/opt/compiler-explorer/gcc-trunk-20231201/include/c++/14.0.0/bits/unique_ptr.h:93:9: error: use of deleted function 'static void O::operator
delete(void*)'
   93 |         delete __ptr;
      |         ^~~~~~~~~~~~
<source>:7:10: note: declared here
    7 |     void operator delete(void *)= delete;
      |          ^~~~~~~~
Compiler returned: 1
```

Use malloc to bypass.

```
#define define_obj_on_heap(type) ({          \
   void* ptr = std::malloc(sizeof(type));\
   reinterpret_cast<type*>(ptr);           \
})
```

https://godbolt.org/z/qb8Y7EE3e

# Disallow stack allocation

```
struct O {
    ~O() = delete;
};

int main() {
    O o;
}
```

```
<source>: In function 'int main()':
<source>:6:7: error: use of deleted function 'O::~O()'
    6 |     O o;
      |       ^
<source>:2:1: note: declared here
    2 | ~O() = delete;
      | ^
Compiler returned: 1
```

# Disallow stack allocation

```
struct O {
    ~O() = delete;
};

int main() {
    O o;
}
```

Dtor guaranteed to be invoked.

```
<source>: In function 'int main()':
<source>:6:7: error: use of deleted function 'O::~O()'
    6 |     O o;
      |       ^
<source>:2:1: note: declared here
    2 | ~O() = delete;
      | ^
Compiler returned: 1
```

# Disallow stack allocation

```
struct O {
    ~O() = delete;
};

int main() {
    O o;
}
```

```
<source>: In function 'int main()':
<source>:6:7: error: use of deleted function 'O::~O()'
    6 |     O o;
      |       ^
<source>:2:1: note: declared here
    2 | ~O() = delete;
      | ^
Compiler returned: 1
```

```
#define define_obj_on_stack(type) ({    \
    char obj_intl__[sizeof(type)];       \
    reinterpret_cast<type*>(obj_intl__);\
    })

int main() {
    auto op = define_obj_on_stack(O);
}
```

Use pointer to bypass.

https://godbolt.org/z/YKErcdo44

Conceptually yes,
but actually no.

# Extra: sequence before

TEAMT5 杜浦數位安全

```
struct O {};

O &f() {
    O o;
    puts(__PRETTY_FUNCTION__);
    return o;
}

O &g() {
    O o;
    puts(__PRETTY_FUNCTION__);
    return o;
}
```

```
int main() {
    f() = g();
    f().operator=(g());
}
```

gcc

```
Program returned:
O& g()
O& f()
O& g()
O& f()
```

clang

```
Program returned: 0
O &g()
O &f()
O &f()
O &g()
```

https://godbolt.org/z/5Ee5bPhef

# Extra: sequence before

TEAMT5 杜浦數位安全

```
struct O {};

O &f() {
    O o;
    puts(__PRETTY_FUNCTION__);
    return o;
}

O &g() {
    O o;
    puts(__PRETTY_FUNCTION__);
    return o;
}
```

```
int main() {
    f() = g();
    f().operator=(g());
}
```

§8.18 p1[expr.ass] The right operand is sequenced before the left operand.

§16.3.1.2 p2 [over.match.oper] …function-call notation…the operands are sequenced in the order prescribed for the built-in operator.

gcc

```
Program returned:
O& g()
O& f()
O& g()
O& f()
```

clang

```
Program returned: 0
O &g()
O &f()
O &f()
O &g()
```

https://godbolt.org/z/5Ee5bPhef

# Extra: sequence before

TEAMT5 杜浦數位安全

```
struct O {};

O &f() {
    O o;
    puts(__PRETTY_FUNCTION__);
    return o;
}

O &g() {
    O o;
    puts(__PRETTY_FUNCTION__);
    return o;
}
```

```
int main() {
    f() = g();
    f().operator=(g());
}
```

§8.18 p1[expr.ass] The right operand is sequenced before the left operand.

§16.3.1.2 p2 [over.match.oper] …function-call notation…the operands are sequenced in the order prescribed for the built-in operator.

gcc

```
Program returned:
O& g()
O& f()
O& g()
O& f()
```

clang

```
Program returned: 0
O &g()
O &f()
O &f()
O &g()
```

https://godbolt.org/z/5Ee5bPhef

# Extra: sequence before

TEAMT5 杜浦數位安全

```
struct O {};

O &f() {
    O o;
    puts(__PRETTY_FUNCTION__);
    return o;
}

O &g() {
    O o;
    puts(__PRETTY_FUNCTION__);
    return o;
}
```

```
int main() {
    f() = g();
    f().operator=(g());
}
```

§8.18 p1[expr.ass] The right operand is sequenced before the left operand.

§16.3.1.2 p2 [over.match.oper] …function-call notation…the operands are sequenced in the order prescribed for the built-in operator.

gcc

```
Program returned:
O& g()
O& f()
O& g()
O& f()
```

clang

```
Program returned: 0
O &g()
O &f()
O &f()
O &g()
```

https://godbolt.org/z/5Ee5bPhef

# Extra: sequence before

```
struct O {};

O &f() {
    O o;
    puts(__PRETTY_FUNCTION__);
    return o;
}

O &g() {
    O o;
    puts(__PRETTY_FUNCTION__);
    return o;
}
```

```
int main() {
    f() = g();
    f().operator=(g());
}
```

So, clang's one is a bug.

gcc

```
Program returned:
O& g()
O& f()
O& g()
O& f()
```

clang

```
Program returned: 0
O &g()
O &f()
O &f()
O &g()
```

https://godbolt.org/z/5Ee5bPhef

# Thank you!

scc@teamt5.org