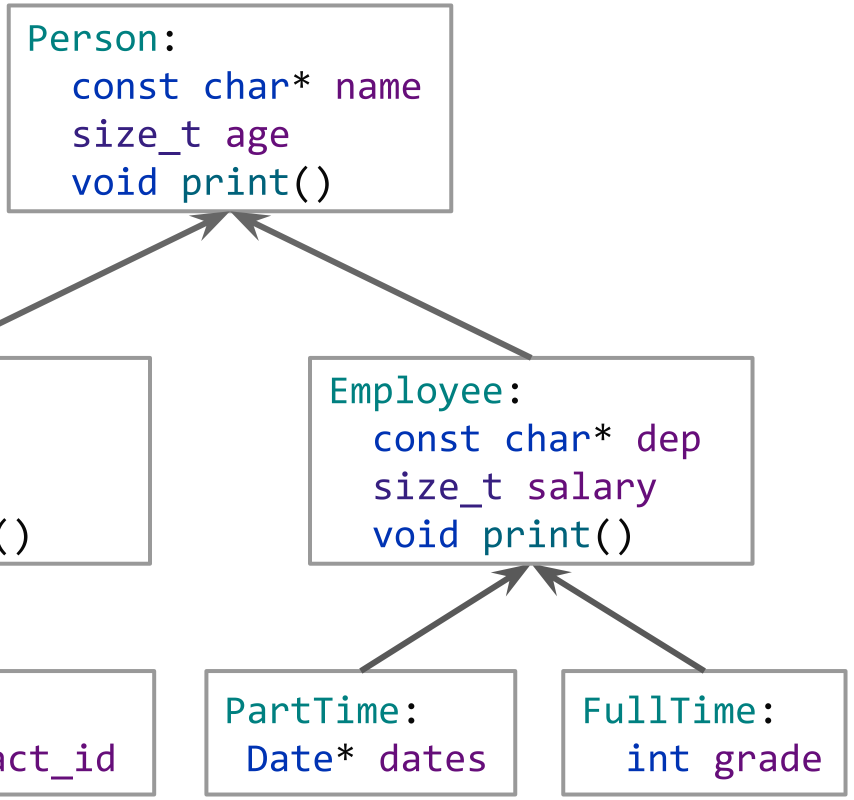
System Programming with C++

VMT

How virtual functions actually work?

Subtyping polymorphism

And all of them have print method (at least one)



Subtyping polymorphism

```
void print_info(Person& k) {
    k.print();
}
```

k can be a reference to some Person instance, but can be also a reference to instance of any Derived class

What should we call here?

Person::print(), Student::print() or Employee::print()?

Subtyping polymorphism

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* name, size_t age):
                     name(name), age(age) {}

    void print() const {
        std::cout << "Person " << name
                  << "; age = " << age
                  << std::endl;
    }
};
```

```
class Student: public Person {
    ...
public:
    ...

    void print() const {
        std::cout << "Student " << name
                  << " from group " << group
                  << std::endl;
    }
};
```

```
class Employee: public Person {
    ...
public:
    ...

    void print() const {
        std::cout << "Employee " << name
                  << " from dep " << dep
                  << std::endl;
    }
};
```

Subtyping polymorphism

```
void print_info(Person& k) {
    k.print();
}
```

k can be a reference to some Person instance, but can be also a reference to instance of any Derived class

```
What should we call here?

Person::print(), Student::print() or Employee::print()?

In such case it is obvious, it should be Person::print
```

Subtyping polymorphism

```
void print_info(Person& k) {
    k.print();
}
```

k can be a reference to some Person instance, but can be also a reference to instance of any Derived class

```
Person p("Bob", 30);
Student s("Alice", 18, 22126, 1);
Employee e("John", 25, "MMF", 5000);
```

```
✓ print_info(p); // Person Bob; age = 30
✓ print_info(s); // Person Alice; age = 18
✓ print_info(e); // Person John; age = 25
```

By default we will call the method print from type that is actually (statically) specified in the code.

Subtyping polymorphism

```cpp
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* name, size_t age):
                         name(name), age(age) {}

    virtual void print() const {
        std::cout << "Person " << name
                  << "; age = " << age
                  << std::endl;
    }
};
```

Virtual modifier changes this behavior: the closest method to the real type of the instance will be called.

```cpp
class Student: public Person {
    ...
public:
    ...

    void print() const {
        std::cout << "Student " << name
                  << " from group " << group
                  << std::endl;
    }
};
```

```cpp
class Employee: public Person {
    ...
public:
    ...

    void print() const {
        std::cout << "Employee " << name
                  << " from dep " << dep
                  << std::endl;
    }
};
```

Subtyping polymorphism

```
void print_info(Person& k) {
    k.print();
}
```

k can be a reference to some Person instance, but can be also a reference to instance of any Derived class

What should we call here?

Person::print(), Student::print() or Employee::print()?

This time this is not so obvious!

Subtyping polymorphism

```
void print_info(Person& k) {
    k.print();
}
```

k can be a reference to some Person instance, but can be also a reference to instance of any Derived class

What should we call here?

Person::print(), Student::print() or Employee::print()?

This time it is not so obvious! We just can't know it during compilation of method print_info!

Subtyping polymorphism

In C++ values can have static and dynamic type.

```
void print_info(Person& k) {
    k.print();
}


Person p("Bob", 30);
Student s("Alice", 18, 22126, 1);
Employee e("John", 25, "MMF", 5000);

✓ print_info(p); // Person Bob; age = 30
✓ print_info(s); // Student Alice from group 22126
✓ print_info(e); // Employee John from dep MMF
```

But if print is virtual method: the closest print to real derived class (that was passed here) will be called.

Subtyping polymorphism

```
void print_info(Person& k) {
    k.print();
}
```

k can be a reference to some Person instance, but can be also a reference to instance of any Derived class

```
What should we call here?

Person::print(), Student::print() or Employee::print()?

Looks like we need to somehow check reference to which
object do we have in runtime.
```

Let's reverse engineer it!

```
# nmap -v -sS -O 10.2.2.2

Starting nmap V. 2.54BETA25
Insufficient responses for TCP sequencing (3), OS detection
accurate
Interesting ports on 10.2.2.2:
(The 1539 ports scanned but not shown below are in state: c
Port       State       Service
22/tcp     open        ssh

No exact OS matches for host

Nmap run completed -- 1 IP address (1 host up) scanneds
# sshnuke 10.2.2.2 -rootpw="Z1ON0101"
Connecting to 10.2.2.2:ssh ... successful.
Attempting to exploit SSHv1 CRC32 ... successful.
Reseting root password to "Z1ON0101".
System open: Access Level <9>
# ssh 10.2.2.2 -l root
root@10.2.2.2's password:
```

Mystery virtual

```
class Person {
protected:
    const char* name;
    size_t age;
    friend void print_info(Person& k)
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    void print() const { ... }
};
```

```
void print_info(Person& k) {
    printf("%s", k.name);
}
```

https://godbolt.org/z/Kv3MW8E73

Mystery virtual

```
class Person {
protected:
    const char* name;
    size_t age;
    friend void print_info(Person& k)
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    void print() const { ... }
};
```

```
void print_info(Person& k) {
    printf("%s", k.name);
}
```

↓

```
.LC0:
        .string "%s"
print(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        nop
        leave
        ret
```

Mystery virtual

https://godbolt.org/z/Kv3MW8E73

```
class Person {
protected:
    const char* name;
    size_t age;
    friend void print_info(Person& k)
public:
    Person(const char* n, size_t a):
        name(n), age(a) {}
    void print() const { ... }
};
```

```
void print_info(Person& k) {
    printf("%s", k.name);
}
```

↓

```
.LC0:
        .string "%s"
print(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        nop
        leave
        ret
```

dereference of k to get access to Person.name ⟶

https://godbolt.org/z/9rTv5Knze

Mystery virtual

```
class Person {
protected:
    const char* name;
    size_t age;
    friend void print_info(Person& k)
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    virtual void print() const { ... }
};
```

```
void print_info(Person& k) {
    printf("%s", k.name);
}
```

$\downarrow$

```
.LC0:
        .string "%s"
print(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax+8]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        nop
        leave
        ret
```

dereference of k to get access to Person.name $\longrightarrow$

Mystery virtual

https://godbolt.org/z/9rTv5Knze

```cpp
class Person {
protected:
    const char* name;
    size_t age;
    friend void print_info(Person& k)
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    virtual void print() const { ... }
};
```

```cpp
void print_info(Person& k) {
    printf("%s", k.name);
}
```

```asm
.LC0:
        .string "%s"
print(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax+8]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        nop
        leave
        ret
```

dereference of k to get access to Person.name $\longrightarrow$

Fact #1: when we add a virtual method, fields offsets are changed

Mystery virtual

https://godbolt.org/z/9rTv5Knze

```
class Person {
protected:
    const char* name;
    size_t age;
    friend void print_info(Person& k)
public:
    Person(const char* n, size_t a):
        name(n), age(a) {}
    virtual void print() const { ... }
};
```

```
void print_info(Person& k) {
    printf("%s", k.name);
}
```

```
.LC0:
        .string "%s"
print(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax+8]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        nop
        leave
        ret
```

dereference of k to get access to Person.name $\longrightarrow$ `mov rax, QWORD PTR [rax+8]`

Fact #1: when we add a virtual method, fields offsets are changed… just like we have some additional field in the very beginning of an object

Mystery virtual

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    void print() const { ... }
};
```

```
void print_info(Person& k) {
    k.print();
}
```

Mystery virtual

https://godbolt.org/z/hcYs199vd

```cpp
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    void print() const { ... }
};
```

```cpp
void print_info(Person& k) {
    k.print();
}
```

$\downarrow$

```asm
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    Person::print() const
        nop
        leave
        ret
```

Mystery virtual

https://godbolt.org/z/hcYs199vd

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    void print() const { ... }
};
```

```
void print_info(Person& k) {
    k.print();
}
```

Passing k as "this" argument to print

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    Person::print() const
        nop
        leave
        ret
```

Mystery virtual

https://godbolt.org/z/hcYs199vd

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    void print() const { ... }
};
```

```
void print_info(Person& k) {
    k.print();
}
```

↓

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    Person::print() const
        nop
        leave
        ret
```

Passing k as "this" argument to print →

Direct call of Person::print →

Mystery virtual

https://godbolt.org/z/Tnesvxzez

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    virtual void print() const { ... }
};
```

```
void print_info(Person& k) {
    k.print();
}
```

↓

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

Mystery virtual

https://godbolt.org/z/Tnesvxzez

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    virtual void print() const { ... }
};
```

```
void print_info(Person& k) {
    k.print();
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

Passing k as "this" argument to print

Indirect call of something in rdx reg

Mystery virtual

https://godbolt.org/z/Tnesvxzez

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    virtual void print() const { ... }
};
```

```
void print_info(Person& k) {
    k.print();
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

What the hell is in rdx? → `mov rdx, QWORD PTR [rax]`

Passing k as "this" argument to print → `mov rax, QWORD PTR [rbp-8]`, `mov rdi, rax`

Indirect call of something in rdx reg → `call rdx`

Mystery virtual

https://godbolt.org/z/Tnesvxzez

```cpp
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    virtual void print() const { ... }
};
```

```cpp
void print_info(Person& k) {
    k.print();
}
```

```asm
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

rax contains an address of an object

What the hell is in rdx?

Passing k as "this" argument to print

Indirect call of something in rdx reg

https://godbolt.org/z/Tnesvxzez

Mystery virtual

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    virtual void print() const { ... }
};
```

```
void print_info(Person& k) {
    k.print();
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

rax contains an address of an object → `mov rax, QWORD PTR [rbp-8]`

rax contains value of the first field → `mov rax, QWORD PTR [rax]`

What the hell is in rdx? → `mov rdx, QWORD PTR [rax]`

Passing k as "this" argument to print → `mov rax, QWORD PTR [rbp-8]`, `mov rdi, rax`

Indirect call of something in rdx reg → `call rdx`

Mystery virtual

https://godbolt.org/z/Tnesvxzez

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    virtual void print() const { ... }
};
```

```
void print_info(Person& k) {
    k.print();
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

rax contains an address of an object → `mov rax, QWORD PTR [rbp-8]`

rax contains value of first field (an address) → `mov rax, QWORD PTR [rax]`

What the hell is in rdx? → `mov rdx, QWORD PTR [rax]`

Passing k as "this" argument to print → `mov rax, QWORD PTR [rbp-8]`, `mov rdi, rax`

Indirect call of something in rdx reg → `call rdx`

Mystery virtual

https://godbolt.org/z/Tnesvxzez

```cpp
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
        name(n), age(a) {}
    virtual void print() const { ... }
};
```

```cpp
void print_info(Person& k) {
    k.print();
}
```

```asm
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

rax contains an address of an object

rax contains value of first field (an address)

we dereference it and store result into rdx

Passing k as "this" argument to print

Indirect call of something in rdx reg

https://godbolt.org/z/Tnesvxzez

Mystery virtual

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    virtual void print() const { ... }
};
```

```
void print_info(Person& k) {
    k.print();
}
```

↓

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

Fact #2: first field is somehow used to get an address of function to call indirectly.

Mystery virtual

https://godbolt.org/z/YxqTf35zz

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    virtual void print() const { ... }
    virtual void test() const { ... }
};
```

```
void print_info(Person& k) {
    k.test();
}
```

↓

```
print_info(Person&):
       push    rbp
       mov     rbp, rsp
       sub     rsp, 16
       mov     QWORD PTR [rbp-8], rdi
       mov     rax, QWORD PTR [rbp-8]
       mov     rax, QWORD PTR [rax]
       add     rax, 8
       mov     rdx, QWORD PTR [rax]
       mov     rax, QWORD PTR [rbp-8]
       mov     rdi, rax
       call    rdx
       nop
       leave
       ret
```

Mystery virtual

https://godbolt.org/z/YxqTf35zz

```cpp
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    virtual void print() const { ... }
    virtual void test() const { ... }
};
```

```cpp
void print_info(Person& k) {
    k.test();
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        add     rax, 8
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

the only difference is this increment of rax $\longrightarrow$

Mystery virtual

https://godbolt.org/z/YxqTf35zz

```cpp
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    virtual void print() const { ... }
    virtual void test() const { ... }
};
```

```cpp
void print_info(Person& k) {
    k.test();
}
```

$\downarrow$

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        add     rax, 8
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

the only difference is this increment of rax $\longrightarrow$ `add rax, 8`

so, we dereference the first field with some offset $\longrightarrow$ `mov rdx, QWORD PTR [rax]`

Mystery virtual

https://godbolt.org/z/YxqTf35zz

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    virtual void print() const { ... }
    virtual void test() const { ... }
};
```

```
void print_info(Person& k) {
    k.test();
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        add     rax, 8
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

the only difference is this increment of rax →

so, we dereference the first field with some offset →

Fact #3: looks like we work with the first field just like it is an array

Mystery virtual

https://godbolt.org/z/YxqTf35zz

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
        name(n), age(a) {}
    virtual void print() const { ... }
    virtual void test() const { ... }
};
```

```
void print_info(Person& k) {
    k.test();
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        add     rax, 8
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

the only difference is this increment of rax $\rightarrow$ add rax, 8

so, we dereference the first field with some offset $\rightarrow$ mov rdx, QWORD PTR [rax]

Fact #3: looks like we work with the first field just like it is an array (with addresses of functions!)

Reverse engineering results

Fact #1: when we add a virtual method to the class, additional field is added (with offset zero) to objects of such class

```
$ nmap -v -sS -O 10.2.2.2

Starting nmap V. 2.54BETA25
Insufficient responses for TCP sequencing (3),
accurate
Interesting ports on 10.2.2.2:
(The 1539 ports scanned but not shown below ar
Port        State       Service
22/tcp      open        ssh

No exact OS matches for host

Nmap run completed -- 1 IP address (1 host up)
$ sshnuke 10.2.2.2 -rootpw="Z1ON0101"
Connecting to 10.2.2.2:ssh ... successful.
Attempting to exploit SSHv1 CRC32 ... successf
Reseting root password to "Z1ON0101".
System open: Access Level (9)
$ ssh 10.2.2.2 -l root
root@10.2.2.2's password:
```

Reverse engineering results

Fact #1: when we add a virtual method to the class, additional field is added (with offset zero) to objects of such class

Fact #2: this field is somehow used to get an address of function for indirect call

```
# nmap -v -sS -O 10.2.2.2

Starting nmap V. 2.54BETA25
Insufficient responses for TCP sequencing (3),
accurate
Interesting ports on 10.2.2.2:
(The 1539 ports scanned but not shown below ar
Port       State       Service
22/tcp     open        ssh

No exact OS matches for host

Nmap run completed -- 1 IP address (1 host up
# sshnuke 10.2.2.2 -rootpw="Z1ON0101"
Connecting to 10.2.2.2:ssh ... successful.
Attempting to exploit SSHv1 CRC32 ... success
Reseting root password to "Z1ON0101".
System open: Access Level <9>
# ssh 10.2.2.2 -l root
root@10.2.2.2's password:
```

Reverse engineering results

Fact #1: when we add a virtual method to the class, additional field is added (with offset zero) to objects of such class

Fact #2: this field is somehow used to get an address of function for indirect call

Fact #3: looks like this first field contains an address of array and for different virtual calls we get different elements from this array (which should be pointers to functions)

VMT (Virtual Method Table)

VMT (Virtual Method Table)

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    virtual void print() const { ... }
    virtual void test() const { ... }
};
```

VMT (Virtual Method Table)

For each class with virtual methods (both own and inherited) compiler generates special table

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    virtual void print() const { ... }
    virtual void test() const { ... }
};
```

Person::print	Person::test

VMT (Virtual Method Table)

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    virtual void print() const { ... }
    virtual void test() const { ... }
};
```

For each class with virtual methods (both own and inherited) compiler generates special table

Person::print	Person::test

It contains addresses of virtual functions implementations (the most specific to this class)

VMT (Virtual Method Table)

For each class with virtual methods (both own and inherited) compiler generates special table

```
class Student: public Person {
protected:
    size_t group;
public:
    Student(const char* n, size_t a, size_t g):
          Person(n, a), group(g) {}

    void print() const {
       std::cout << "Student " << name
                 << " from group " << group
                 << std::endl;
    }
};
```

Student::print	Person::test

It contains addresses of virtual functions implementations (the most specific to this class)

VMT (Virtual Method Table)

```
class Employee: public Person {
protected:
    size_t salary;
public:
    Employee(const char* n, size_t a, size_t s):
          Person(n, a), salary(s) {}

    void test() const { … }
};
```

For each class with virtual methods (both own and inherited) compiler generates special table

Person::print	Employee::test

It contains addresses of virtual functions implementations (the most specific to this class)

VMT (Virtual Method Table)

For each class with virtual methods (both own and inherited) compiler generates special table

```
class Employee: public Person {
protected:
    size_t salary;
public:
    Employee(const char* n, size_t a, size_t s):
          Person(n, a), salary(s) {}

    void test() const { … }
};
```

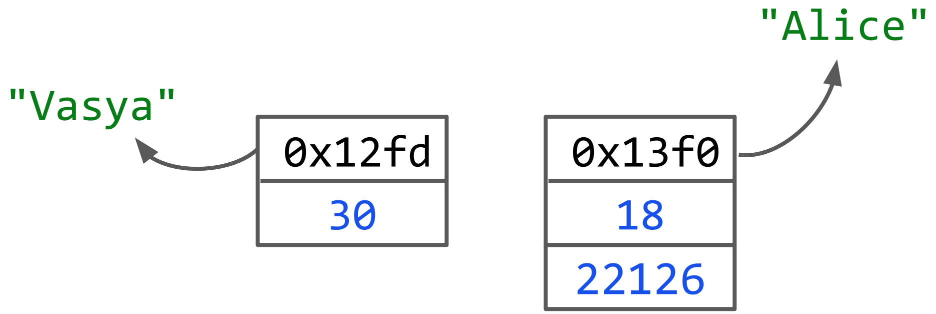
Person::print	Employee::test

It contains addresses of virtual functions implementations (the most specific to this class)

Each object of such classes contains pointer to VMT.

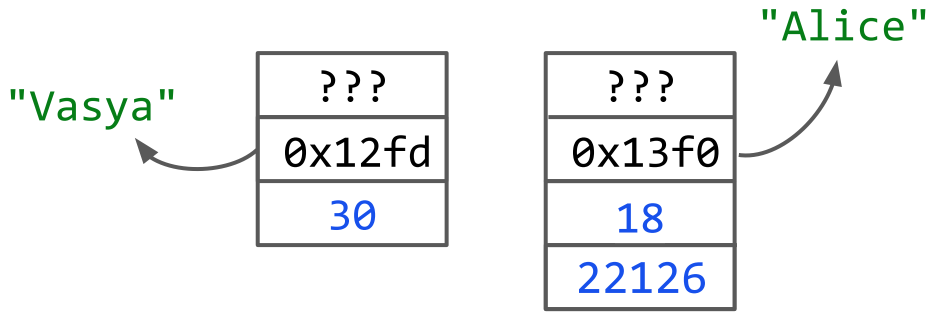
VMT (Virtual Method Table)

```
Person* p = new Person("Vasya", 30);
Person* s = new Student("Alice", 18, 22126);
```



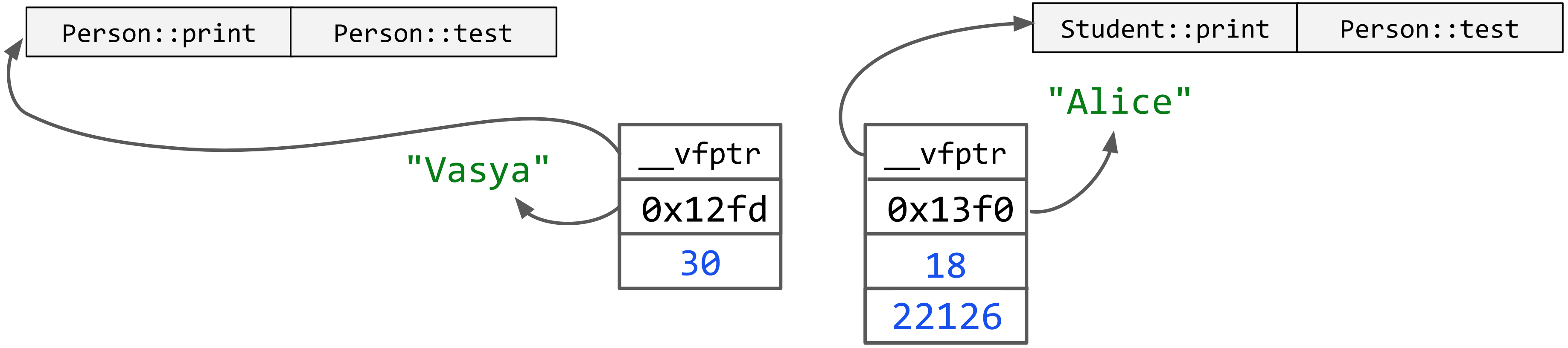
VMT (Virtual Method Table)

```
Person* p = new Person("Vasya", 30);
Person* s = new Student("Alice", 18, 22126);
```



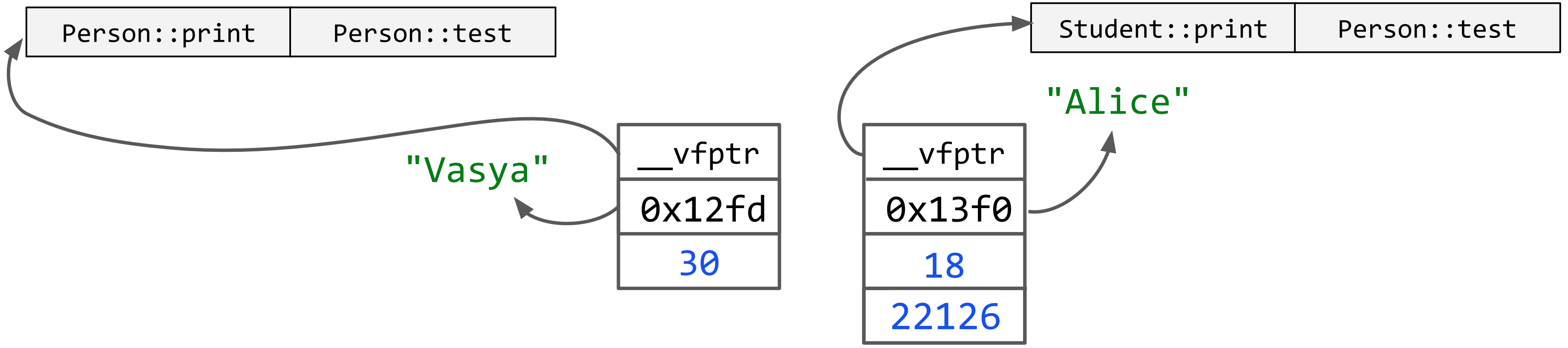
VMT (Virtual Method Table)

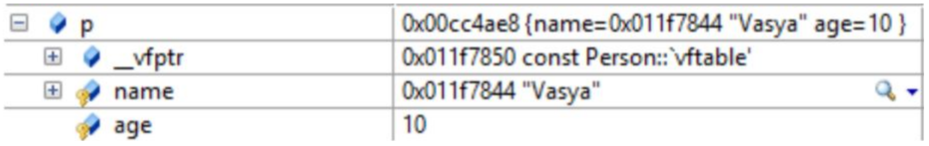
```
Person* p = new Person("Vasya", 30);
Person* s = new Student("Alice", 18, 22126);
```



VMT (Virtual Method Table)

```
Person* p = new Person("Vasya", 30);
Person* s = new Student("Alice", 18, 22126);
```





p	0x00cc4ae8 {name=0x011f7844 "Vasya" age=10 }
__vfptr	0x011f7850 const Person::`vftable'
name	0x011f7844 "Vasya"
age	10

Debuggers in IDE can show
it or try to hide it.

VMT (Virtual Method Table)

https://godbolt.org/z/Tnesvxzez

```cpp
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    virtual void print() const { ... }
};
```

```cpp
void print_info(Person& k) {
    k.print();
}
```

```asm
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

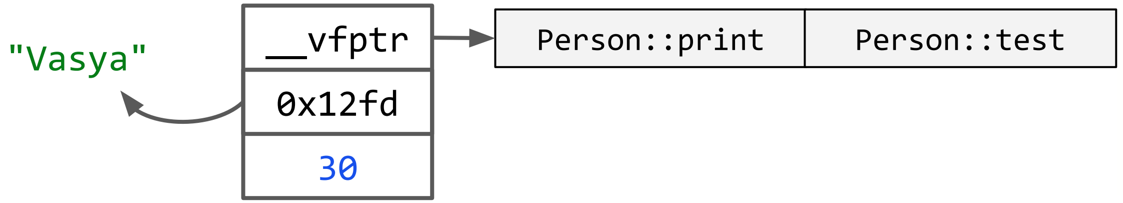
https://godbolt.org/z/Tnesvxzez

VMT (Virtual Method Table)

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
        name(n), age(a) {}
    virtual void print() const { ... }
};

Person p = Person("Vasya", 30);
print_info(p);
```

```
void print_info(Person& k) {
    k.print();
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```



VMT (Virtual Method Table)

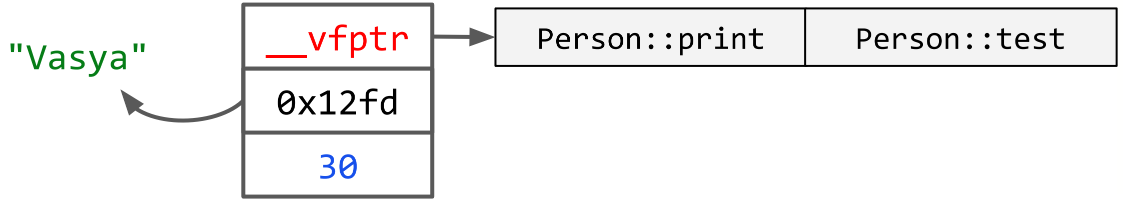
https://godbolt.org/z/Tnesvxzez

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    virtual void print() const { ... }
};

Person p = Person("Vasya", 30);
print_info(p);
```

```
void print_info(Person& k) {
    k.print();
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

takes __vfptr



VMT (Virtual Method Table)

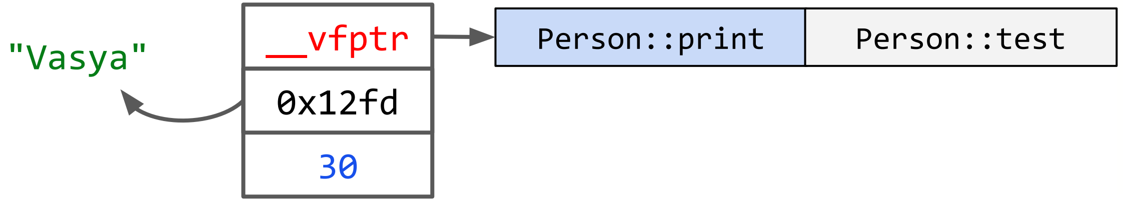
https://godbolt.org/z/Tnesvxzez

```cpp
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    virtual void print() const { ... }
};

Person p = Person("Vasya", 30);
print_info(p);
```

```cpp
void print_info(Person& k) {
    k.print();
}
```

```asm
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

takes __vfptr[0]



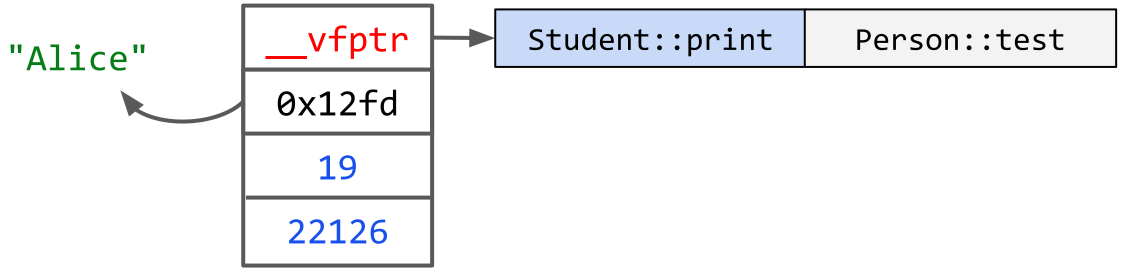
VMT (Virtual Method Table)

https://godbolt.org/z/Tnesvxzez

```
class Student: public Person {
protected:
    size_t group;
public:
    Student(const char* n, size_t a, size_t g):
          Person(n, a), group(g) {}

    void print() const { … }
};

Student s = Student("Alice", 19, 22126);
print_info(s);
```

```
void print_info(Person& k) {
    k.print();
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```



VMT (Virtual Method Table)

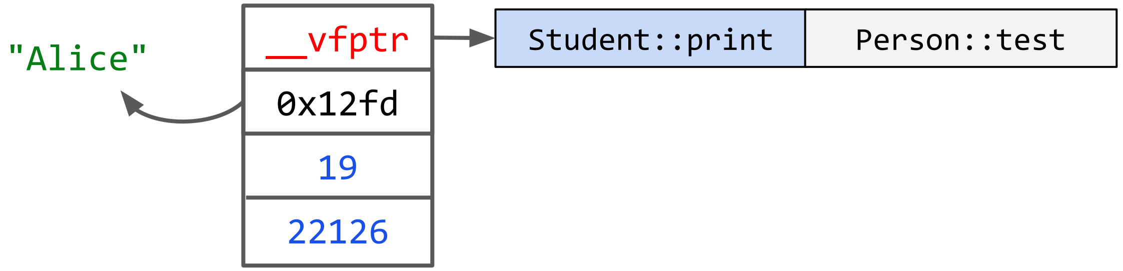
https://godbolt.org/z/Tnesvxzez

```
class Student: public Person {
protected:
    size_t group;
public:
    Student(const char* n, size_t a, size_t g):
          Person(n, a), group(g) {}

    void print() const { … }
};

Student s = Student("Alice", 19, 22126);
print_info(s);
```

```
void print_info(Person& k) {
    k.print();
}
```

The same code,
different behaviour

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```



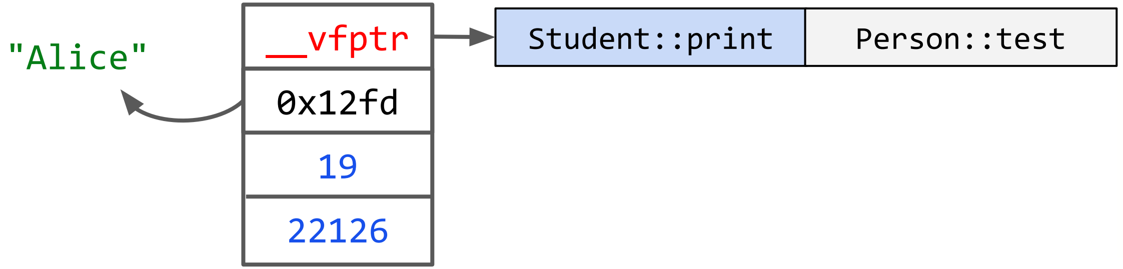
https://godbolt.org/z/Tnesvxzez

VMT (Virtual Method Table)

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    virtual void print() const { ... }
};

Student s = Student("Alice", 19, 22126);
print_info(s);
```

```
void print_info(Person& k) {
    k.test();
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```



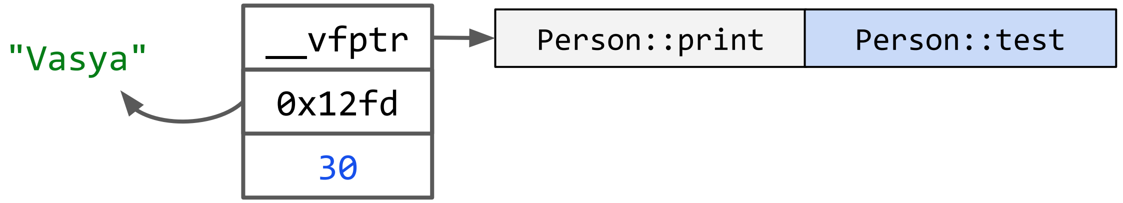
VMT (Virtual Method Table)

https://godbolt.org/z/YxqTf35zz

```cpp
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    virtual void print() const { ... }
    virtual void test() const { ... }
};

Person p = Person("Vasya", 30);
print_info(p);
```

```cpp
void print_info(Person& k) {
    k.test();
}
```

```asm
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        add     rax, 8
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```



VMT so *far*

`Terminology:`

VMT so far

Terminology: if it is known which method to call in compile time, it is called static or early binding.

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}
    void print() const { ... }
};
```

```
void print_info(Person& k) {
    k.print();
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    Person::print() const
```

VMT so far

Terminology: if method to call is chosen in runtime, it is called dynamic or late binding.

```cpp
void print_info(Person& k) {
    k.print();
}
```

$\longrightarrow$

```asm
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

VMT so *far*

Terminology: if method to call is chosen in runtime, it is called dynamic or late binding.

VMT is only one (but of course classical) of many possible approaches how to implement late binding.

VMT so far

Terminology: if method to call is chosen in runtime, it is called dynamic or late binding.

VMT is only one (but of course classical) of many possible approaches how to implement late binding.

In C++ it is always possible to say whether late or early binding will be used in the concrete code (but you/compiler should analyze the type hierarchy for that).

VMT so far

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    Person::print() const
```

```
void print_info(Person& k) {
    k.print();
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

What should be chosen?

VMT so far

```
void print_info(Person& k) {
    k.print();
}
```

If print was never virtual

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    Person::print() const
```

If print was virtual in Person (or its base class!)

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

VMT so *far*

`Why not all methods are virtual by default (like in Java)?`

VMT so *far*

Why not all methods are virtual by default (like in Java)?

Because late binding is expensive!!

VMT so far

```
void print_info(Person& k) {
    k.print();
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    Person::print() const
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

2 additional dereferences!

VMT so far

```
void print_info(Person& k) {
    k.print();
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    Person::print() const
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

2 additional dereferences!

Also, maybe some additional work with indexes (but it is just nothing in comparison with dereferences).

VMT so *far*

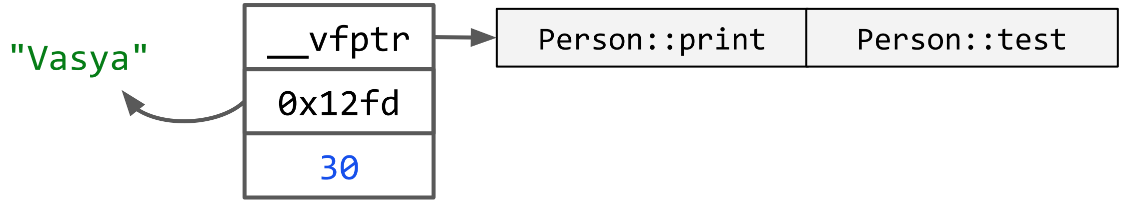
Why not all methods are virtual by default (like in Java)?

1. It is expensive in terms of performance.
2. What else?

VMT so far

Why not all methods are virtual by default (like in Java)?

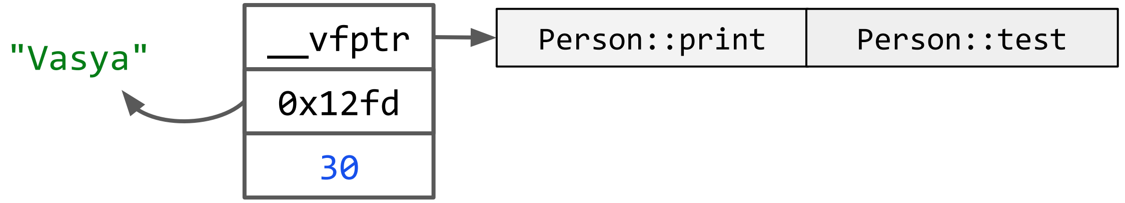
1. It is expensive in terms of performance.
2. Objects become fatty.



VMT so far

Why not all methods are virtual by default (like in Java)?

1. It is expensive in terms of performance.
2. Objects become fatty => it is expensive in terms of memory.



VMT so far

Why not all methods are virtual by default (like in Java)?

1. It is expensive in terms of performance.
2. Objects become fatty => it is expensive in terms of memory.

C++ philosophy: don't pay for features you don't need.

VMT so *far*

Why not all methods are virtual by default (like in Java)?

1. It is expensive in terms of performance.

2. Objects become fatty => it is expensive in terms of memory.

C++ philosophy: don’t pay for features you don’t need.
So, no virtual methods by default.

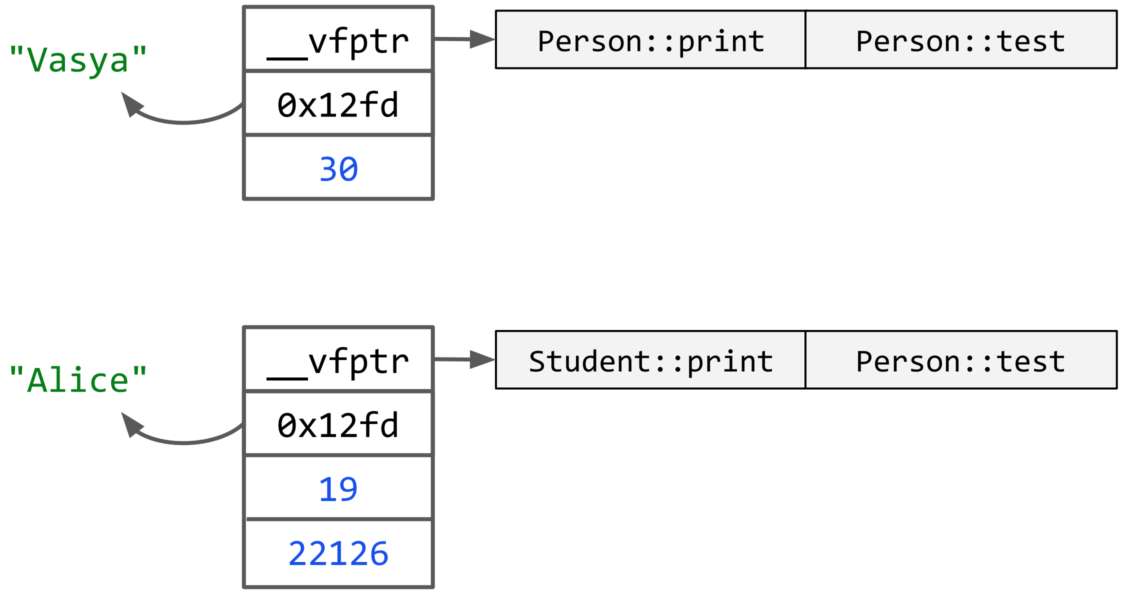
VMT: more questions

VMT: more questions

Question: how and when this field __vfptr is initialized?

VMT: more questions

`Question: how and when this field __vfptr is initialized?`



VMT: more questions

Question: how and when this field __vfptr is initialized?

Answer: in the constructor of course!

```cpp
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}

    virtual void print() const { ... }
    virtual void test() const { ... }
};
```

https://godbolt.org/z/PEhv4a8en

```cpp
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}

    virtual void print() const { ... }
    virtual void test() const { ... }
};
```

```
Person::Person(char const*, unsigned long)
[base object constructor]:

        mov     QWORD PTR [rdi],
                OFFSET FLAT:vtable for Person+16
        mov     QWORD PTR [rdi+8], rsi
        mov     QWORD PTR [rdi+16], rdx
        ret
```

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}

    virtual void print() const { ... }
    virtual void test() const { ... }
};
```

$\longrightarrow$

https://godbolt.org/z/PEhv4a8en

```
Person::Person(char const*, unsigned long)
[base object constructor]:

        mov     QWORD PTR [rdi],
                OFFSET FLAT:vtable for Person+16
        mov     QWORD PTR [rdi+8], rsi
        mov     QWORD PTR [rdi+16], rdx
        ret
```

static data

```
vtable for Person:
        .quad   0
        .quad   typeinfo for Person
        .quad   Person::print() const
        .quad   Person::test() const
```

https://godbolt.org/z/PEhv4a8en

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}

    virtual void print() const { ... }
    virtual void test() const { ... }
};
```

```
Person::Person(char const*, unsigned long)
[base object constructor]:

        mov     QWORD PTR [rdi],
                OFFSET FLAT:vtable for Person+16
        mov     QWORD PTR [rdi+8], rsi
        mov     QWORD PTR [rdi+16], rdx
        ret
```

static data

smth interesting we will discuss later

```
vtable for Person:
        .quad   0
        .quad   typeinfo for Person
        .quad   Person::print() const
        .quad   Person::test() const
```

https://godbolt.org/z/PEhv4a8en

```cpp
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person(const char* n, size_t a):
          name(n), age(a) {}

    virtual void print() const { ... }
    virtual void test() const { ... }
};
```

```asm
Person::Person(char const*, unsigned long)
[base object constructor]:

        mov     QWORD PTR [rdi],
                OFFSET FLAT:vtable for Person+16
        mov     QWORD PTR [rdi+8], rsi
        mov     QWORD PTR [rdi+16], rdx
        ret
```

static data

```asm
vtable for Person:
        .quad   0
        .quad   typeinfo for Person
        .quad   Person::print() const
        .quad   Person::test() const
```

smth interesting we will discuss later → `.quad 0`, `.quad typeinfo for Person`

addresses of implementations of virtual methods → `.quad Person::print() const`, `.quad Person::test() const`

```
class Student: public Person {
protected:
    size_t group;
public:
    Student(const char* n, size_t a, size_t g):
          Person(n, a), group(g) {}

    void print() const {
       std::cout << "Student " << name
                 << " from group " << group
                 << std::endl;
    }
};
```

```cpp
class Student: public Person {
protected:
    size_t group;
public:
    Student(const char* n, size_t a, size_t g):
          Person(n, a), group(g) {}

    void print() const {
       std::cout << "Student " << name
                 << " from group " << group
                 << std::endl;
    }
};
```

```asm
Student::Student(char const*,
                 unsigned long,
                 unsigned long)
[base object constructor]:
    push    rbp
    mov     rbp, rcx
    push    rbx
    mov     rbx, rdi
    sub     rsp, 8
    call    Person::Person(char const*,
                           unsigned long)
            [base object constructor]
    mov     QWORD PTR [rbx],
            OFFSET FLAT:vtable for Student+16
    mov     QWORD PTR [rbx+24], rbp
    add     rsp, 8
    pop     rbx
    pop     rbp
    ret
```

https://godbolt.org/z/13x7db3Wf

```cpp
class Student: public Person {
protected:
    size_t group;
public:
    Student(const char* n, size_t a, size_t g):
          Person(n, a), group(g) {}

    void print() const {
       std::cout << "Student " << name
                 << " from group " << group
                 << std::endl;
    }
};
```

```
Student::Student(char const*,
                 unsigned long,
                 unsigned long)
[base object constructor]:
    push    rbp
    mov     rbp, rcx
    push    rbx
    mov     rbx, rdi
    sub     rsp, 8
    call    Person::Person(char const*,
                           unsigned long)
            [base object constructor]
    mov     QWORD PTR [rbx],
            OFFSET FLAT:vtable for Student+16
    mov     QWORD PTR [rbx+24], rbp
    add     rsp, 8
    pop     rbx
    pop     rbp
    ret
```

initialization of
__vfptr for Student

https://godbolt.org/z/13x7db3Wf

```cpp
class Student: public Person {
protected:
    size_t group;
public:
    Student(const char* n, size_t a, size_t g):
          Person(n, a), group(g) {}

    void print() const {
       std::cout << "Student " << name
                 << " from group " << group
                 << std::endl;
    }
};
```

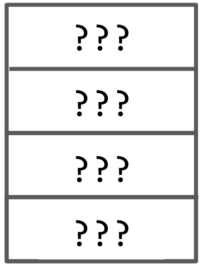
```asm
Student::Student(char const*,
                 unsigned long,
                 unsigned long)
[base object constructor]:
    push    rbp
    mov     rbp, rcx
    push    rbx
    mov     rbx, rdi
    sub     rsp, 8
    call    Person::Person(char const*,
                          unsigned long)
            [base object constructor]
    mov     QWORD PTR [rbx],
            OFFSET FLAT:vtable for Student+16
    mov     QWORD PTR [rbx+24], rbp
    add     rsp, 8
    pop     rbx
    pop     rbp
    ret
```

But we've already set it in Person constructor!

https://godbolt.org/z/13x7db3Wf

```
class Student: public Person {
protected:
    size_t group;
public:
    Student(const char* n, size_t a, size_t g):
          Person(n, a), group(g) {}

    void print() const {
       std::cout << "Student " << name
                 << " from group " << group
                 << std::endl;
    }
};

Student s = Student("Alice", 19, 22126);
```

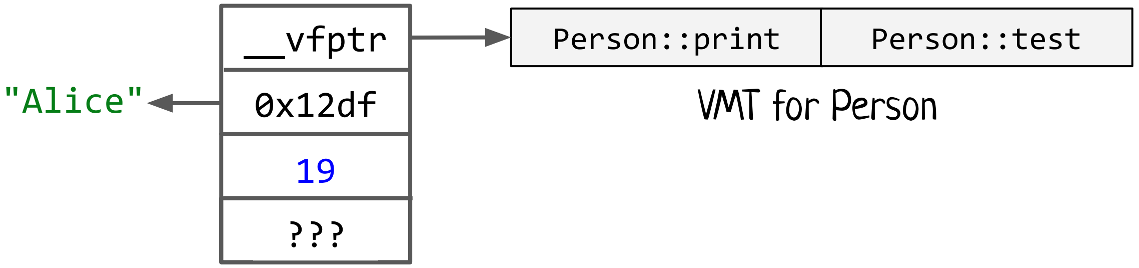
```
Student::Student(char const*,
                 unsigned long,
                 unsigned long)
[base object constructor]:
    push    rbp
    mov     rbp, rcx
    push    rbx
    mov     rbx, rdi
    sub     rsp, 8
    call    Person::Person(char const*,
                     unsigned long)
            [base object constructor]
    mov     QWORD PTR [rbx],
            OFFSET FLAT:vtable for Student+16
    mov     QWORD PTR [rbx+24], rbp
    add     rsp, 8
    pop     rbx
    pop     rbp
    ret
```



https://godbolt.org/z/13x7db3Wf

```
class Student: public Person {
protected:
    size_t group;
public:
    Student(const char* n, size_t a, size_t g):
          Person(n, a), group(g) {}

    void print() const {
       std::cout << "Student " << name
                 << " from group " << group
                 << std::endl;
    }
};

Student s = Student("Alice", 19, 22126);
```

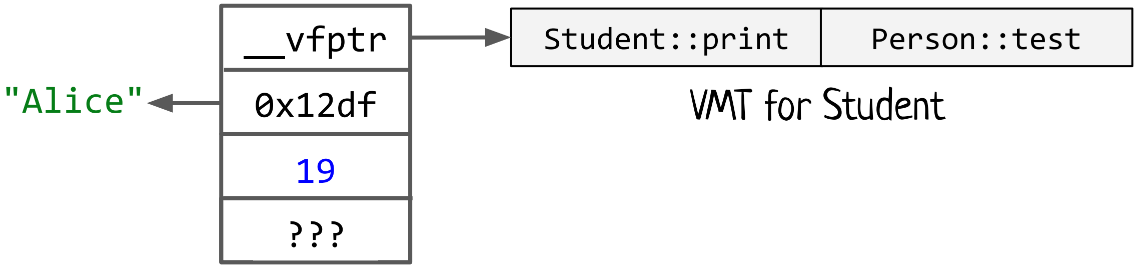
```
Student::Student(char const*,
                 unsigned long,
                 unsigned long)
[base object constructor]:
    push    rbp
    mov     rbp, rcx
    push    rbx
    mov     rbx, rdi
    sub     rsp, 8
    call    Person::Person(char const*,
                     unsigned long)
            [base object constructor]
    mov     QWORD PTR [rbx],
            OFFSET FLAT:vtable for Student+16
    mov     QWORD PTR [rbx+24], rbp
    add     rsp, 8
    pop     rbx
    pop     rbp
    ret
```



https://godbolt.org/z/13x7db3Wf

```cpp
class Student: public Person {
protected:
    size_t group;
public:
    Student(const char* n, size_t a, size_t g):
          Person(n, a), group(g) {}

    void print() const {
       std::cout << "Student " << name
                 << " from group " << group
                 << std::endl;
    }
};

Student s = Student("Alice", 19, 22126);
```

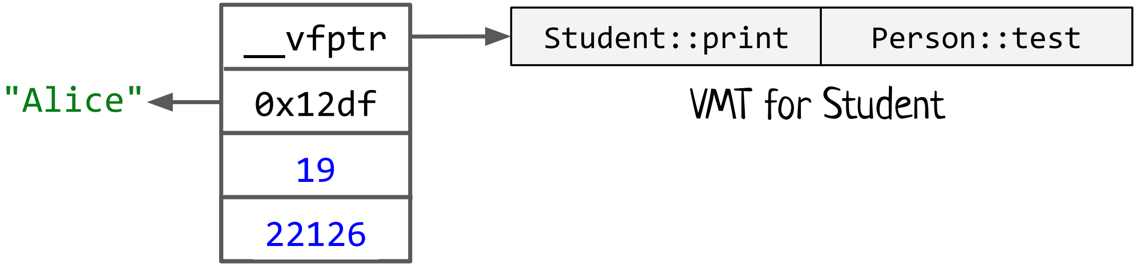
```
Student::Student(char const*,
                 unsigned long,
                 unsigned long)
[base object constructor]:
    push    rbp
    mov     rbp, rcx
    push    rbx
    mov     rbx, rdi
    sub     rsp, 8
    call    Person::Person(char const*,
                           unsigned long)
            [base object constructor]
    mov     QWORD PTR [rbx],
            OFFSET FLAT:vtable for Student+16
    mov     QWORD PTR [rbx+24], rbp
    add     rsp, 8
    pop     rbx
    pop     rbp
    ret
```



https://godbolt.org/z/13x7db3Wf

```
class Student: public Person {
protected:
    size_t group;
public:
    Student(const char* n, size_t a, size_t g):
          Person(n, a), group(g) {}

    void print() const {
       std::cout << "Student " << name
                 << " from group " << group
                 << std::endl;
    }
};

Student s = Student("Alice", 19, 22126);
```

```
Student::Student(char const*,
                 unsigned long,
                 unsigned long)
[base object constructor]:
    push    rbp
    mov     rbp, rcx
    push    rbx
    mov     rbx, rdi
    sub     rsp, 8
    call    Person::Person(char const*,
                      unsigned long)
            [base object constructor]
    mov     QWORD PTR [rbx],
            OFFSET FLAT:vtable for Student+16
    mov     QWORD PTR [rbx+24], rbp
    add     rsp, 8
    pop     rbx
    pop     rbp
    ret
```



https://godbolt.org/z/13x7db3Wf

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person() { std::cout << "Base"; }
    virtual void print() const { … }

};

class Student: public Person {
protected:
    size_t group;
public:
    Student() {}

    void print() const { … }
};

Student s = Student();
```

https://godbolt.org/z/TPn1e8jvz

```cpp
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person() { std::cout << "Base"; }
    virtual void print() const { … }

};

class Student: public Person {
protected:
    size_t group;
public:
    Student() {}

    void print() const { … }
};

Student s = Student();
```

$\rightarrow$

```
Student::Student()
[base object constructor]:
        push    rbx
        mov     rbx, rdi
        call    Person::Person()
                [base object constructor]
        mov     QWORD PTR [rbx],
                OFFSET FLAT:vtable for Student+16
        pop     rbx
        ret
```

https://godbolt.org/z/TPn1e8jvz

```cpp
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person() { std::cout << "Base"; }
    virtual void print() const { … }

};

class Student: public Person {
protected:
    size_t group;
public:
    Student() {}

    void print() const { … }
};

Student s = Student();
```

Base constructor is called even without your direct order.

```
Student::Student()
[base object constructor]:
        push    rbx
        mov     rbx, rdi
        call    Person::Person()
                [base object constructor]
        mov     QWORD PTR [rbx],
                OFFSET FLAT:vtable for Student+16
        pop     rbx
        ret
```

https://godbolt.org/z/TPn1e8jvz

```cpp
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person() { std::cout << "Base"; }
    virtual void print() const { … }

};

class Student: public Person {
protected:
    size_t group;
public:
    Student() {}

    void print() const { … }
};

Student s = Student();
```

Base constructor is called even without your direct order.

So, __vfptr is overridden in each constructor (if not optimized out).

```
Student::Student()
[base object constructor]:
        push    rbx
        mov     rbx, rdi
        call    Person::Person()
                [base object constructor]
        mov     QWORD PTR [rbx],
                OFFSET FLAT:vtable for Student+16
        pop     rbx
        ret
```

https://godbolt.org/z/TPn1e8jvz

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person() { this->print(); }
    virtual void print() const { … }

};

class Student: public Person {
protected:
    size_t group;
public:
    Student() {}

    void print() const { … }
};

Student s = Student();
```

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person() { this->print(); }
    virtual void print() const { … }

};

class Student: public Person {
protected:
    size_t group;
public:
    Student() {}

    void print() const { … }
};

Student s = Student();
```

Which method will be called here?
Person::print or Student::print?

Currently constructing Student

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person() { this->print(); }
    virtual void print() const { … }

};

class Student: public Person {
protected:
    size_t group;
public:
    Student() {}

    void print() const { … }
};

Student s = Student();
```

Which method will be called here? Person::print or Student::print?

According to late binding conception it should depend on the real (dynamic) type of this.

Currently constructing Student

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person() { this->print(); }
    virtual void print() const { … }

};

class Student: public Person {
protected:
    size_t group;
public:
    Student() {}

    void print() const { … }
};

Student s = Student();
```

Which method will be called here? Person::print or Student::print?

According to late binding conception it should depend on the real (dynamic) type of this. It could be Person*, or Student* or anyone from the hierarchy.

Currently constructing Student

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person() { this->print(); }
    virtual void print() const { … }

};

class Student: public Person {
protected:
    size_t group;
public:
    Student() {}

    void print() const { … }
};

Student s = Student();
```

Which method will be called here?
Person::print or Student::print?

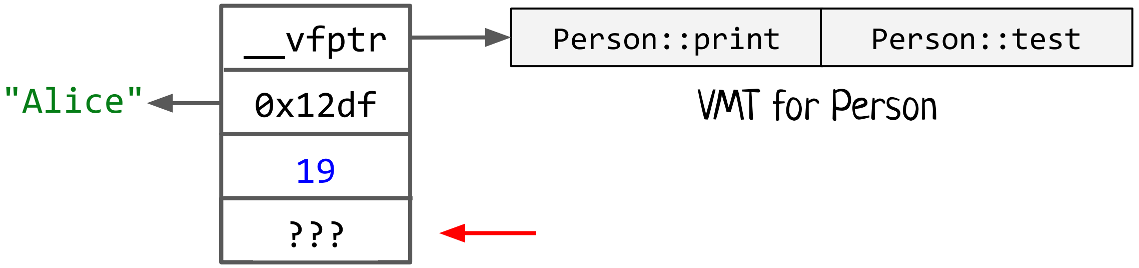
According to late binding conception it should depend on the real (dynamic) type of this. It could be Person*, or Student* or anyone from the hierarchy.

But that's not true! Currently __vfptr is set to VMT of Person! Person::print will be called.

Currently constructing Student

```cpp
class Student: public Person {
protected:
    size_t group;
public:
    Student(const char* n, size_t a, size_t g):
          Person(n, a), group(g) {}

    void print() const {
       std::cout << "Student " << name
                 << " from group " << group
                 << std::endl;
    }
};

Student s = Student("Alice", 19, 22126);
```

```asm
Student::Student(char const*,
                 unsigned long,
                 unsigned long)
[base object constructor]:
    push    rbp
    mov     rbp, rcx
    push    rbx
    mov     rbx, rdi
    sub     rsp, 8
    call    Person::Person(char const*,
                           unsigned long)
            [base object constructor]
    mov     QWORD PTR [rbx],
            OFFSET FLAT:vtable for Student+16
    mov     QWORD PTR [rbx+24], rbp
    add     rsp, 8
    pop     rbx
    pop     rbp
    ret
```



And that's absolutely right decision as fields of Student are 100% not yet ready!!!

https://godbolt.org/z/13x7db3Wf

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person() { this->print(); }
    virtual void print() const { … }

};

class Student: public Person {
protected:
    size_t group;
public:
    Student() {}

    void print() const { … }
};

Student s = Student();
```

Which method will be called here? Person::print or Student::print?

Person::print will be called.

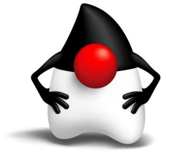
Currently constructing Student

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person() { this->print(); }
    virtual void print() const { … }

};

class Student: public Person {
protected:
    size_t group;
public:
    Student() {}

    void print() const { … }
};

Student s = Student();
```

Which method will be called here?
Person::print or Student::print?

Person::print will be called.

Fun fact: in Java behavior is different, Student::print will be called.



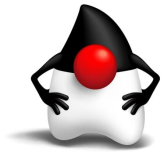
Currently constructing Student

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    Person() { this->print(); }
    virtual void print() const { … }

};

class Student: public Person {
protected:
    size_t group;
public:
    Student() {}

    void print() const { … }
};

Student s = Student();
```

Which method will be called here?
Person::print or Student::print?

Person::print will be called.

Fun fact: in Java behavior is different, Student::print will be called. The only excuse for that is default zeroing of fields there.



Currently constructing Student

VMT: more questions

Question #1: how and when this field __vfptr is initialized?
Answer: in the constructor of course!

Question #2: where else __vfptr can be changed?

VMT: more questions

Question #1: how and when this field __vfptr is initialized?
Answer: in the constructor of course!

Question #2: where else __vfptr can be changed?
Answer: in the destructor of course!

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    ...
    void print() const { … }
    virtual ~Person() { cout << "Bye"
                             << name << endl; }
};

class Student: public Person {
protected:
    size_t group;
public:
    ...
    void print() const { … }
    ~Student() { cout << group << "--" << endl;}
};

Student* s = new Student("Alice", 19, 22126);
delete s;
```

What will be printed?

https://godbolt.org/z/8KxvdjPTz

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    ...
    void print() const { … }
    virtual ~Person() { cout << "Bye"
                             << name << endl; }
};

class Student: public Person {
protected:
    size_t group;
public:
    ...
    void print() const { … }
    ~Student() { cout << group << "--" << endl;}
};

Student* s = new Student("Alice", 19, 22126);
delete s;
```

What will be printed?

```
22126--
Bye, Alice
```

https://godbolt.org/z/8KxvdjPTz

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    ...
    void print() const { … }
    virtual ~Person() { cout << "Bye"
                             << name << endl; }
};

class Student: public Person {
protected:
    size_t group;
public:
    ...
    void print() const { … }
    ~Student() { cout << group << "--" << endl;}
};

Student* s = new Student("Alice", 19, 22126);
delete s; ←
```

https://godbolt.org/z/8KxvdjPTz

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    ...
    void print() const { … }
    virtual ~Person() { cout << "Bye"
                             << name << endl; }
};

class Student: public Person {
protected:
    size_t group;
public:
    ...
    void print() const { … }
    ~Student() { cout << group << "--" << endl;}
};

Student* s = new Student("Alice", 19, 22126);
delete s;
```

https://godbolt.org/z/8KxvdjPTz

```cpp
class Person {
protected:
    const char* name;
    size_t age;
public:
    …
    void print() const { … }
    virtual ~Person() { cout << "Bye"
                             << name << endl; }
};

class Student: public Person {
protected:
    size_t group;
public:
    …
    void print() const { … }
    ~Student() { cout << group << "--" << endl;}
};

Student* s = new Student("Alice", 19, 22126);
delete s;
```

$\rightarrow$

```asm
Student::~Student():
        push    r12
        push    rbp
        mov     rbp, rdi
        push    rbx
        mov     rsi, QWORD PTR [rdi+24]
        mov     QWORD PTR [rdi],
                OFFSET FLAT:vtable for Student+16
        mov     edi, OFFSET FLAT:std::cout
        …
        jmp     Person::~Person()
```

https://godbolt.org/z/8KxvdjPTz

```cpp
class Person {
protected:
    const char* name;
    size_t age;
public:
    …
    void print() const { … }
    virtual ~Person() { cout << "Bye"
                         << name << endl; }
};

class Student: public Person {
protected:
    size_t group;
public:
    …
    void print() const { … }
    ~Student() { cout << group << "--" << endl;}
};

Student* s = new Student("Alice", 19, 22126);
delete s;
```

First of all, we update __vfptr (maybe this is not a Student, but its derived class instance)?

→

```asm
Student::~Student():
        push    r12
        push    rbp
        mov     rbp, rdi
        push    rbx
        mov     rsi, QWORD PTR [rdi+24]
        mov     QWORD PTR [rdi],
                OFFSET FLAT:vtable for Student+16
        mov     edi, OFFSET FLAT:std::cout
        …
        jmp     Person::~Person()
```

https://godbolt.org/z/8KxvdjPTz

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    …
    void print() const { … }
    virtual ~Person() { cout << "Bye"
                        << name << endl; }
};

class Student: public Person {
protected:
    size_t group;
public:
    …
    void print() const { … }
    ~Student() { cout << group << "--" << endl;}
};

Student* s = new Student("Alice", 19, 22126);
delete s;
```

First of all, we update __vfptr (maybe this is not a Student, but its derived class instance)?

Next: we execute destructor of the class

```
Student::~Student():
        push    r12
        push    rbp
        mov     rbp, rdi
        push    rbx
        mov     rsi, QWORD PTR [rdi+24]
        mov     QWORD PTR [rdi],
                OFFSET FLAT:vtable for Student+16
        mov     edi, OFFSET FLAT:std::cout
        …
        jmp     Person::~Person()
```

https://godbolt.org/z/8KxvdjPTz

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    …
    void print() const { … }
    virtual ~Person() { cout << "Bye"
                                << name << endl; }
};

class Student: public Person {
protected:
    size_t group;
public:
    …
    void print() const { … }
    ~Student() { cout << group << "--" << endl;}
};

Student* s = new Student("Alice", 19, 22126);
delete s;
```

First of all, we update __vfptr (maybe this is not a Student, but its derived class instance)?

Next: we execute destructor of the class

Finaly: call destructor of base

```
Student::~Student():
        push    r12
        push    rbp
        mov     rbp, rdi
        push    rbx
        mov     rsi, QWORD PTR [rdi+24]
        mov     QWORD PTR [rdi],
                OFFSET FLAT:vtable for Student+16
        mov     edi, OFFSET FLAT:std::cout
        …
        jmp     Person::~Person()
```

https://godbolt.org/z/8KxvdjPTz

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    …
    void print() const { … }
    virtual ~Person() { cout << "Bye"
                           << name << endl; }
};

class Student: public Person {
protected:
    size_t group;
public:
    …
    void print() const { … }
    ~Student() { cout << group << "--" << endl;}
};

Student* s = new Student("Alice", 19, 22126);
delete s;
```

Again, we start from updating __vfptr!

```
Person::~Person():
        push    rbx
        mov     edx, 3
        mov     rbx, rdi
        ...
        mov     QWORD PTR [rdi],
                OFFSET FLAT:vtable for Person+16
        ...
```

https://godbolt.org/z/8KxvdjPTz

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    ...
    void print() const { … }
    virtual ~Person() { this->print(); }
};

class Student: public Person {
protected:
    size_t group;
public:
    ...
    void print() const { … }
    ~Student() { cout << group << "--" << endl;}
};

Student* s = new Student("Alice", 19, 22126);
delete s;
```

Again, we start from updating __vfptr!

What will be called here?

https://godbolt.org/z/8KxvdjPTz

```cpp
class Person {
protected:
    const char* name;
    size_t age;
public:
    …
    void print() const { … }
    virtual ~Person() { this->print(); }
};

class Student: public Person {
protected:
    size_t group;
public:
    …
    void print() const { … }
    ~Student() { cout << group << "--" << endl;}
};

Student* s = new Student("Alice", 19, 22126);
delete s;
```

Again, we start from updating __vfptr!

What will be called here?

Again, Person::print!

https://godbolt.org/z/8KxvdjPTz

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    …
    void print() const { … }
    virtual ~Person() { this->print(); }
};

class Student: public Person {
protected:
    size_t group;
public:
    …
    void print() const { … }
    ~Student() { cout << group << "--" << endl;}
};

Student* s = new Student("Alice", 19, 22126);
delete s;
```

Again, we start from updating __vfptr!

What will be called here?

Again, Person::print! And it is again absolutely right, as fields of Student are already inaccessible.

https://godbolt.org/z/8KxvdjPTz

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    …
    void print() const { … }
    virtual ~Person() { this->print(); }  ←
};

class Student: public Person {
protected:
    size_t group;
public:
    …
    void print() const { … }
    ~Student() { cout << group << "--" << endl;}
};

Student* s = new Student("Alice", 19, 22126);
delete s;
```

Again, we start from updating __vfptr!

What will be called here?

Again, Person::print! And it is again absolutely right, as fields of Student are already inaccessible.

But be careful with that: such behavior of late binding in constructors and destructors can be counterintuitive.

https://godbolt.org/z/8KxvdjPTz

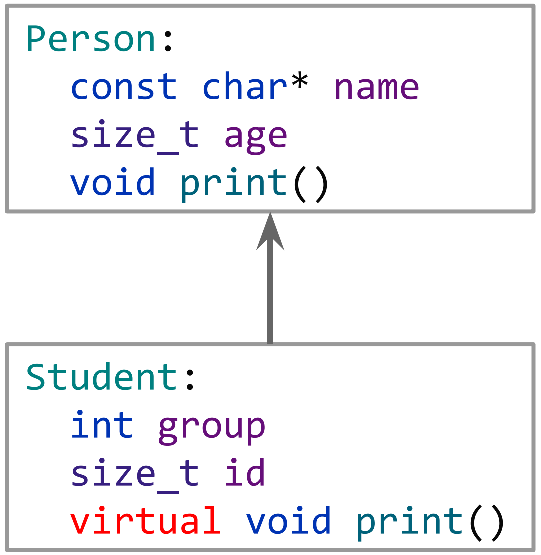
VMT: more questions

Question #1: how and when this field __vfptr is initialized?
Answer: in the constructor of course!

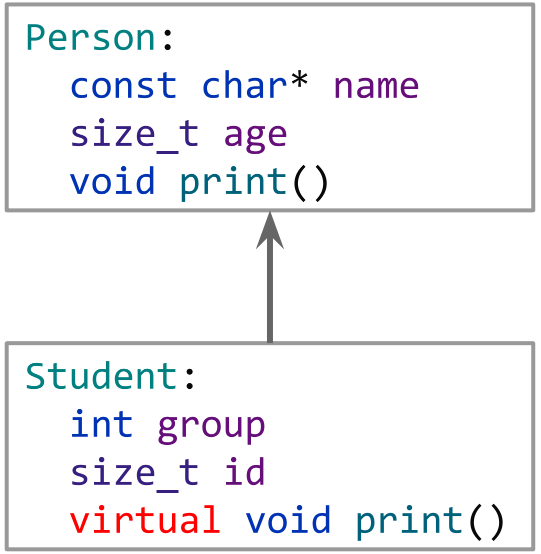
Question #2: where else __vfptr can be changed?
Answer: in the destructor of course!

Question #3: can derived class override some method AND make it virtual (if previously it wasn't virtual)?

```
class Person {
protected:
    const char* name;
    size_t age;
public:
    …
    void print() const { … }
    …
};

class Student: public Person {
protected:
    size_t group;
public:
    …
    virtual void print() const { … }
    …
};
```



```
class Person {
protected:
    const char* name;
    size_t age;
public:
    …
    void print() const { … }
    …
};

class Student: public Person {
protected:
    size_t group;
public:
    …
    virtual void print() const { … }
    …
};
```



Shouldn't have VMT

Should have VMT

```cpp
class Person {
protected:
    const char* name;
    size_t age;
    friend print_info(Person&);
public:
    …
    void print() const { … }
    …
};

class Student: public Person {
protected:
    size_t group;
    friend print_student(Student&);
public:
    …
    virtual void print() const { … }
    …
};
```

```cpp
void print_info(Person& k) {
    k.print();
    std::cout << k.name;
}

void print_student(Student& s) {
    s.print();
    std::cout << s.name;
}
```

```cpp
void print_student(Student& s) {
    s.print();
    std::cout << s.name;
}
```

```asm
print_student(Student&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax+8]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:_ZSt4cout
        ...
```

https://godbolt.org/z/GqvabshP7

```
void print_student(Student& s) {
    s.print();
    std::cout << s.name;
}
```

$\rightarrow$

```
print_student(Student&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax+8]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:_ZSt4cout
        ...
```

virtual call

https://godbolt.org/z/GqvabshP7

```cpp
void print_student(Student& s) {
    s.print();
    std::cout << s.name;
}
```

```asm
print_student(Student&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax+8]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:_ZSt4cout
        ...
```

virtual call

access to the first field
(offset is +8 as we also have __vfptr)

https://godbolt.org/z/GqvabshP7

```cpp
void print_student(Student& s) {
    s.print();
    std::cout << s.name;
}
```

```asm
print_student(Student&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax+8]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:_ZSt4cout
        ...
```

virtual call

access to the first field
(offset is +8 as we also have __vfptr)

Everything seems fine, as usual

https://godbolt.org/z/GqvabshP7

```
void print_info(Person& k) {
    k.print();
    std::cout << k.name;
}
```

```
void print_info(Person& k) {
    k.print();
    std::cout << k.name;
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rsi, rax
        ...
```

direct call

https://godbolt.org/z/GqvabshP7

```cpp
void print_info(Person& k) {
    k.print();
    std::cout << k.name;
}
```

```asm
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rsi, rax
        ...
```

direct call

first field usage… with +0 offset

https://godbolt.org/z/GqvabshP7

```cpp
void print_info(Person& k) {
    k.print();
    std::cout << k.name;
}
```

```asm
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rsi, rax
        ...
```

direct call

first field usage… with +0 offset

isn't it strange for you?

https://godbolt.org/z/GqvabshP7

```
void print_info(Person& k) {
    k.print();
    std::cout << k.name;
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rsi, rax
        ...
```

direct call

first field usage… with +0 offset

isn't it strange for you?

because actually we can have derived class here, right?

https://godbolt.org/z/GqvabshP7

```
void print_info(Person& k) {
    k.print();
    std::cout << k.name;
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rsi, rax
        ...
```

direct call

first field usage… with +0 offset

```
int main() {
    Person p = Person("Vasya", 30);
    print_info(p);
    Student s = Student("Alice", 19, 22126);
    print_info(s);
    print_student(s);
    return 0;
}
```

https://godbolt.org/z/hrcj4TMc8

```
void print_info(Person& k) {
    k.print();
    std::cout << k.name;
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rsi, rax
        ...
```

direct call

first field usage… with +0 offset

```
int main() {
    Person p = Person("Vasya", 30);
    print_info(p);
    Student s = Student("Alice", 19, 22126);
    print_info(s);
    print_student(s);
    return 0;
}
```

```
        lea     rax, [rbp-32]
        mov     rdi, rax
        call    print_info(Person&)
        ...
        lea     rax, [rbp-64]
        add     rax, 8
        mov     rdi, rax
        call    print_info(Person&)
        lea     rax, [rbp-64]
        mov     rdi, rax
        call    print_student(Student&)
```

https://godbolt.org/z/hrcj4TMc8

```cpp
void print_info(Person& k) {
    k.print();
    std::cout << k.name;
}
```

```asm
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rsi, rax
        ...
```

direct call

first field usage… with +0 offset

```cpp
int main() {
    Person p = Person("Vasya", 30);
    print_info(p);
    Student s = Student("Alice", 19, 22126);
    print_info(s);
    print_student(s);
    return 0;
}
```

```asm
lea     rax, [rbp-32]
mov     rdi, rax
call    print_info(Person&)
...
lea     rax, [rbp-64]
add     rax, 8
mov     rdi, rax
call    print_info(Person&)
lea     rax, [rbp-64]
mov     rdi, rax
call    print_student(Student&)
```

https://godbolt.org/z/hrcj4TMc8

```cpp
void print_info(Person& k) {
    k.print();
    std::cout << k.name;
}
```

direct call

first field usage… with +0 offset

```asm
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rsi, rax
        ...
```

```cpp
int main() {
    Person p = Person("Vasya", 30);
    print_info(p);
    Student s = Student("Alice", 19, 22126);
    print_info(s);
    print_student(s);
    return 0;
}
```

```asm
        lea     rax, [rbp-32]
        mov     rdi, rax
        call    print_info(Person&)
        ...
        lea     rax, [rbp-64]
        add     rax, 8
        mov     rdi, rax
        call    print_info(Person&)
        lea     rax, [rbp-64]
        mov     rdi, rax
        call    print_student(Student&)
```

https://godbolt.org/z/hrcj4TMc8

```cpp
void print_info(Person& k) {
    k.print();
    std::cout << k.name;
}
```

```asm
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rsi, rax
        ...
```

direct call

first field usage… with +0 offset

```cpp
int main() {
    Person p = Person("Vasya", 30);
    print_info(p);
    Student s = Student("Alice", 19, 22126);
    print_info(s);
    print_student(s);
    return 0;
}
```

```asm
        lea     rax, [rbp-32]
        mov     rdi, rax
        call    print_info(Person&)
        ...
        lea     rax, [rbp-64]
        add     rax, 8
        mov     rdi, rax
        call    print_info(Person&)
        lea     rax, [rbp-64]
        mov     rdi, rax
        call    print_student(Student&)
```

skip __vfptr!

https://godbolt.org/z/hrcj4TMc8

```cpp
void print_info(Person& k) {
    k.print();
    std::cout << k.name;
}
```

```asm
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rsi, rax
        ...
```

direct call

first field usage… with +0 offset

So, pointer to the derived is not always the same as pointer to based one.

```cpp
int main() {
    Person p = Person("Vasya", 30);
    print_info(p);
    Student s = Student("Alice", 19, 22126);
    print_info(s);
    print_student(s);
    return 0;
}
```

```asm
lea     rax, [rbp-32]
mov     rdi, rax
call    print_info(Person&)
...
lea     rax, [rbp-64]
add     rax, 8
mov     rdi, rax
call    print_info(Person&)
lea     rax, [rbp-64]
mov     rdi, rax
call    print_student(Student&)
```

skip __vfptr!

https://godbolt.org/z/hrcj4TMc8

VMT: more questions

Question #1: how and when this field __vfptr is initialized?
Answer: in the constructor of course!

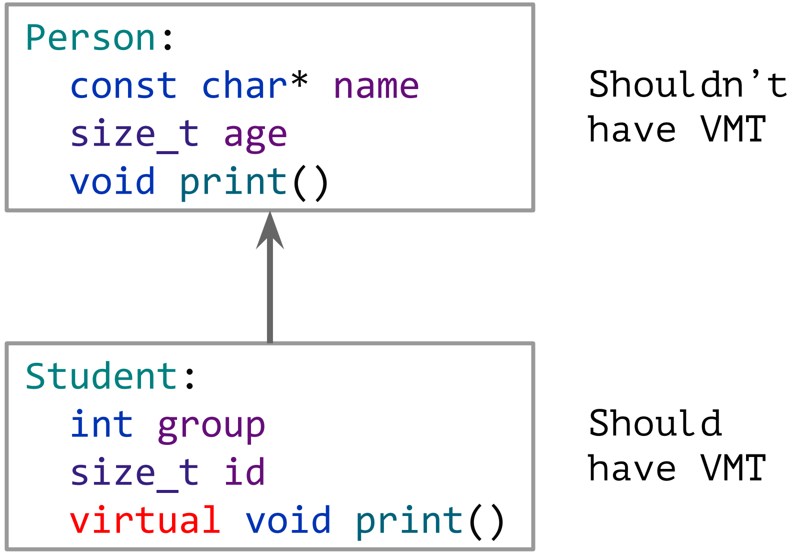
Question #2: where else __vfptr can be changed?
Answer: in the destructor of course!

Question #3: can derived class override some method AND make it virtual (if previously it wasn't virtual)?
Answer: yes, and additional adjusting for pointers could appear in generated code to "disable" late binding.

VMT: bonus

VMT: bonus

```
void print_info(Person& k) {
    k.print();
    Student& s = static_cast<Student&>(k);
    s.print();
}
```



```
Person:
  const char* name
  size_t age
  void print()
```

Shouldn't have VMT

```
Student:
  int group
  size_t id
  virtual void print()
```

Should have VMT

VMT: bonus

```
void print_info(Person& k) {
    k.print();
    Student& s = static_cast<Student&>(k);
    s.print();
}
```

$\longrightarrow$

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     QWORD PTR [rbp-24], rdi
        mov     rax, QWORD PTR [rbp-24]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-24]
        sub     rax, 8
        mov     QWORD PTR [rbp-8], rax
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

https://godbolt.org/z/daPY1dcW6

VMT: bonus

```
void print_info(Person& k) {
    k.print();
    Student& s = static_cast<Student&>(k);
    s.print();
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     QWORD PTR [rbp-24], rdi
        mov     rax, QWORD PTR [rbp-24]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-24]
        sub     rax, 8
        mov     QWORD PTR [rbp-8], rax
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

https://godbolt.org/z/daPY1dcW6

VMT: bonus

this is also
called downcast

```
void print_info(Person& k) {
    k.print();
    Student& s = static_cast<Student&>(k);
    s.print();
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     QWORD PTR [rbp-24], rdi
        mov     rax, QWORD PTR [rbp-24]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-24]
        sub     rax, 8
        mov     QWORD PTR [rbp-8], rax
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

https://godbolt.org/z/daPY1dcW6

VMT: bonus

```
void print_info(Person& k) {
    k.print();
    Student& s = static_cast<Student&>(k);
    s.print();
}
```

this is also
called downcast

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     QWORD PTR [rbp-24], rdi
        mov     rax, QWORD PTR [rbp-24]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-24]
        sub     rax, 8
        mov     QWORD PTR [rbp-8], rax
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

already virtual
call here

https://godbolt.org/z/daPY1dcW6

VMT: bonus

```
void print_info(Person& k) {
    k.print();
    Student& s = static_cast<Student&>(k);
    s.print();
}


int main() {
    Student s = Student("Alice", 19, 22126);
    print_info(s);
    Person p = Person("Vasya", 30);
    print_info(p);
    return 0;
}
```

this is also
called downcast

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     QWORD PTR [rbp-24], rdi
        mov     rax, QWORD PTR [rbp-24]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-24]
        sub     rax, 8
        mov     QWORD PTR [rbp-8], rax
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

https://godbolt.org/z/daPY1dcW6

VMT: bonus

this is also
called downcast

```cpp
void print_info(Person& k) {
    k.print();
    Student& s = static_cast<Student&>(k);
    s.print();
}


int main() {
    Student s = Student("Alice", 19, 22126);
    print_info(s);
    Person p = Person("Vasya", 30);
    print_info(p);
    return 0;
}
```

ok

```asm
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     QWORD PTR [rbp-24], rdi
        mov     rax, QWORD PTR [rbp-24]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-24]
        sub     rax, 8
        mov     QWORD PTR [rbp-8], rax
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

https://godbolt.org/z/daPY1dcW6

VMT: bonus

```
void print_info(Person& k) {
    k.print();
    Student& s = static_cast<Student&>(k);
    s.print();
}
```

this is also called downcast

```
int main() {
    Student s = Student("Alice", 19, 22126);
    print_info(s);
    Person p = Person("Vasya", 30);
    print_info(p);
    return 0;
}
```

ok

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     QWORD PTR [rbp-24], rdi
        mov     rax, QWORD PTR [rbp-24]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-24]
        sub     rax, 8
        mov     QWORD PTR [rbp-8], rax
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

https://godbolt.org/z/daPY1dcW6

VMT: bonus

```
void print_info(Person& k) {
    k.print();
    Student& s = static_cast<Student&>(k);
    s.print();
}
```

this is also called downcast

because it is indeed a student

```
int main() {
    Student s = Student("Alice", 19, 22126);
    print_info(s);
    Person p = Person("Vasya", 30);
    print_info(p);
    return 0;
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     QWORD PTR [rbp-24], rdi
        mov     rax, QWORD PTR [rbp-24]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-24]
        sub     rax, 8
        mov     QWORD PTR [rbp-8], rax
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

https://godbolt.org/z/daPY1dcW6

VMT: bonus

this is also called downcast

```cpp
void print_info(Person& k) {
    k.print();
    Student& s = static_cast<Student&>(k);
    s.print();
}


int main() {
    Student s = Student("Alice", 19, 22126);
    print_info(s);
    Person p = Person("Vasya", 30);
    print_info(p);
    return 0;
}
```

```asm
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     QWORD PTR [rbp-24], rdi
        mov     rax, QWORD PTR [rbp-24]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-24]
        sub     rax, 8
        mov     QWORD PTR [rbp-8], rax
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

https://godbolt.org/z/daPY1dcW6

VMT: bonus

```
void print_info(Person& k) {
    k.print();
    Student& s = static_cast<Student&>(k);
    s.print();
}


int main() {
    Student s = Student("Alice", 19, 22126);
    print_info(s);
    Person p = Person("Vasya", 30);
    print_info(p);
    return 0;
}
```

this is also
called downcast

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     QWORD PTR [rbp-24], rdi
        mov     rax, QWORD PTR [rbp-24]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-24]
        sub     rax, 8
        mov     QWORD PTR [rbp-8], rax
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

https://godbolt.org/z/daPY1dcW6

VMT: bonus

```cpp
void print_info(Person& k) {
    k.print();
    Student& s = static_cast<Student&>(k);
    s.print();
}


int main() {
    Student s = Student("Alice", 19, 22126);
    print_info(s);
    Person p = Person("Vasya", 30);
    print_info(p);
    return 0;
}
```

this is also
called downcast

```asm
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     QWORD PTR [rbp-24], rdi
        mov     rax, QWORD PTR [rbp-24]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-24]
        sub     rax, 8
        mov     QWORD PTR [rbp-8], rax
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

https://godbolt.org/z/daPY1dcW6

VMT: bonus

```cpp
void print_info(Person& k) {
    k.print();
    Student& s = static_cast<Student&>(k);
    s.print();
}

int main() {
    Student s = Student("Alice", 19, 22126);
    print_info(s);
    Person p = Person("Vasya", 30);
    print_info(p);
    return 0;
}
```

this is also
called downcast

```asm
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     QWORD PTR [rbp-24], rdi
        mov     rax, QWORD PTR [rbp-24]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-24]
        sub     rax, 8
        mov     QWORD PTR [rbp-8], rax
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

https://godbolt.org/z/daPY1dcW6

VMT: bonus

this is also
called downcast

```
void print_info(Person& k) {
    k.print();
    Student& s = static_cast<Student&>(k);
    s.print();
}


int main() {
    Student s = Student("Alice", 19, 22126);
    print_info(s);
    Person p = Person("Vasya", 30);
    print_info(p);
    return 0;
}
```

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     QWORD PTR [rbp-24], rdi
        mov     rax, QWORD PTR [rbp-24]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-24]
        sub     rax, 8
        mov     QWORD PTR [rbp-8], rax
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

https://godbolt.org/z/daPY1dcW6

VMT: bonus

```cpp
void print_info(Person& k) {
    k.print();
    Student& s = static_cast<Student&>(k);
    s.print();
}

int main() {
    Student s = Student("Alice", 19, 22126);
    print_info(s);
    Person p = Person("Vasya", 30);
    print_info(p);
    return 0;
}
```

this is also called downcast

```
print_info(Person&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     QWORD PTR [rbp-24], rdi
        mov     rax, QWORD PTR [rbp-24]
        mov     rdi, rax
        call    Person::print() const
        mov     rax, QWORD PTR [rbp-24]
        sub     rax, 8
        mov     QWORD PTR [rbp-8], rax
        mov     rax, QWORD PTR [rbp-8]
        mov     rax, QWORD PTR [rax]
        mov     rdx, QWORD PTR [rax]
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    rdx
        nop
        leave
        ret
```

https://godbolt.org/z/daPY1dcW6

Downcast of the instance of the base class is UB of course, now you see one of the reasons why.

VMT: takeaways

- Early (static) and late (dynamic) binding
- VMT as an implementation of late binding in C++
- Virtual functions are expensive (both performance and memory costs)
- Beware of non-obvious behaviour for virtual calls in constructors and destructors
- Pointer adjustments and downcast pitfalls