Not So Tiny Task №8 (2 + 1 points)

Implement a hierarchy for reading/writing data from/to some source.

2 points

- Base class: IO; Should provide some basic information: if source is still open or not (can be closed manually by close() method), was eof reached or not.
- 1st level of derived classes: Reader and Writer; They provide functions for reading/writing primitive types (and std::strings).
- 2nd level of derived classes: ReaderWriter. It provides functions for reading and writing at the same time.
- 3rd level: specific implementation for different sources 1) std::string as a source, 2) FILE* as a source.

Not So Tiny Task №8 (2 + 1 points)

Implement a hierarchy for reading/writing data from/to some source.

…

- 3rd level: specific implementation for different sources 1) std::string as a source, 2) FILE* as a source.

- 4th level: implementation for both string and FILE* sources with buffer.

+1 point

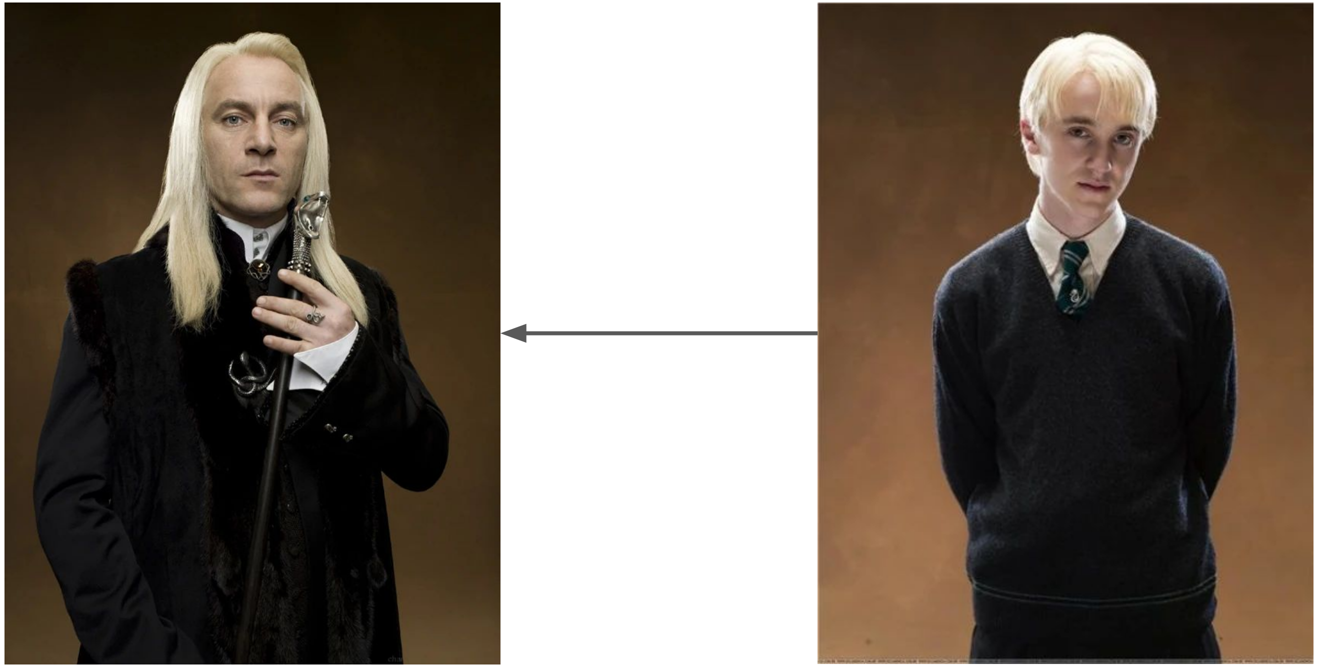
  - Operations firstly read/write from/to the preallocated buffer of fixed size.

  - If buffer is empty/full, classes should read/write to the real source (string or file).

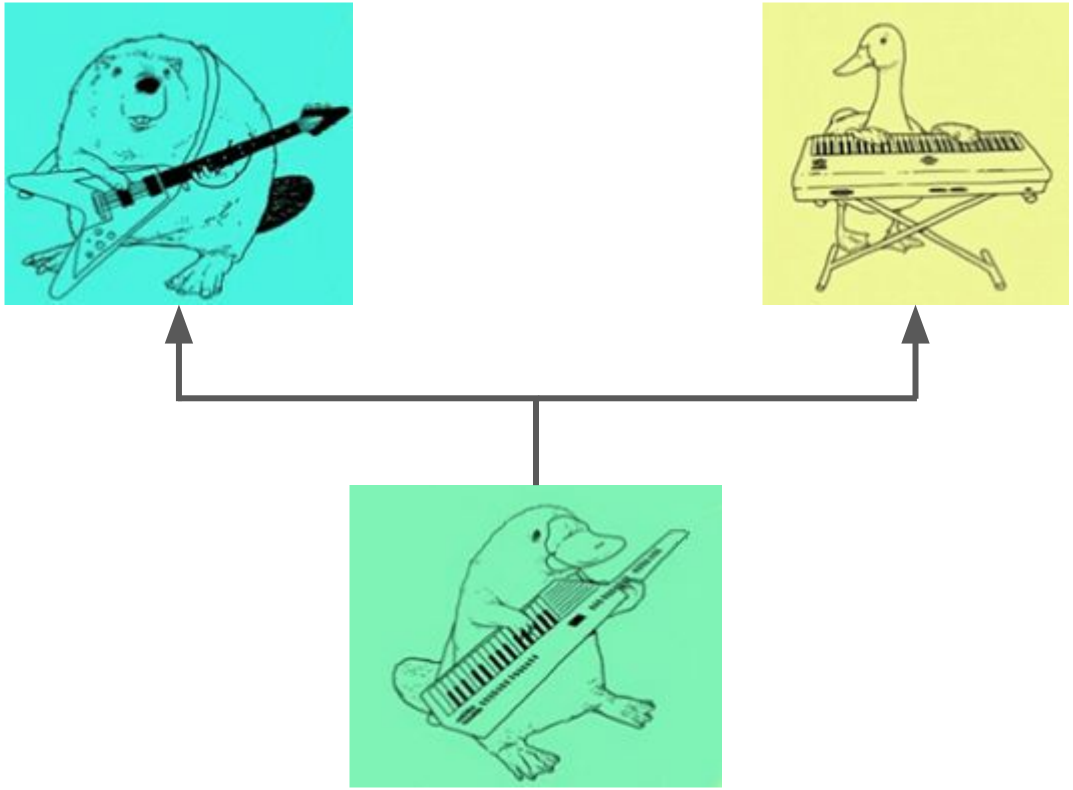
System Programming with C++

Multiple inheritance

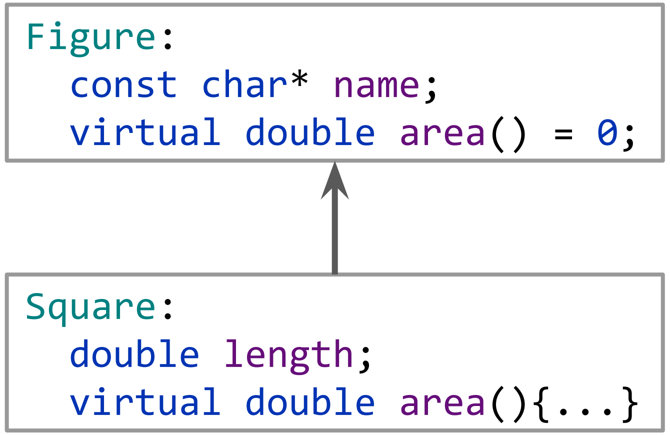
Inheritance



Multiple Inheritance



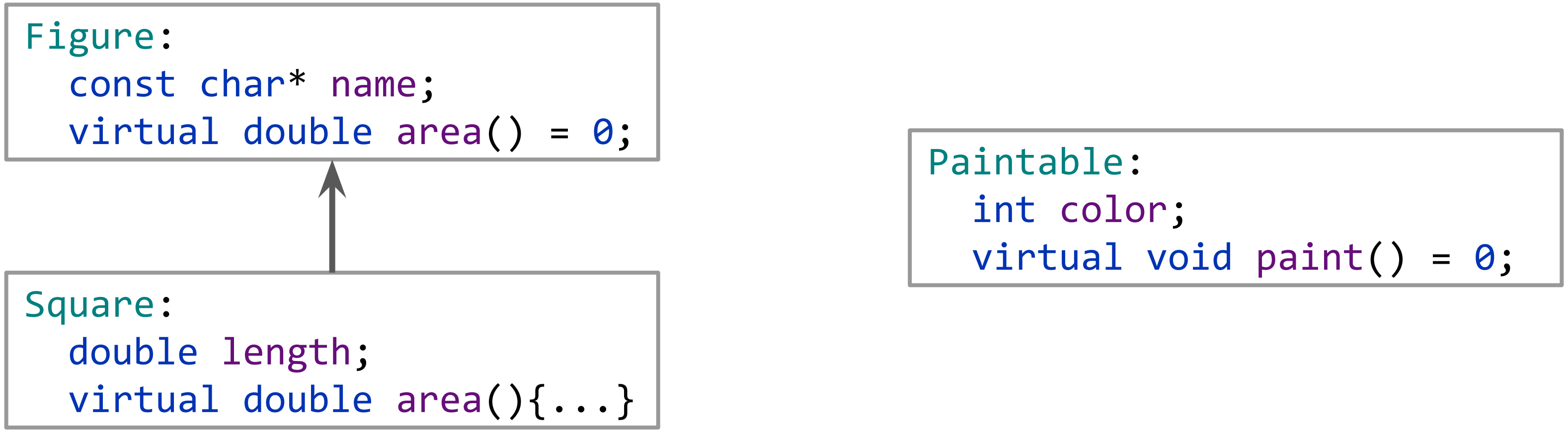
Multiple Inheritance



```
Figure:
  const char* name;
  virtual double area() = 0;
```

```
Square:
  double length;
  virtual double area(){...}
```

Multiple Inheritance

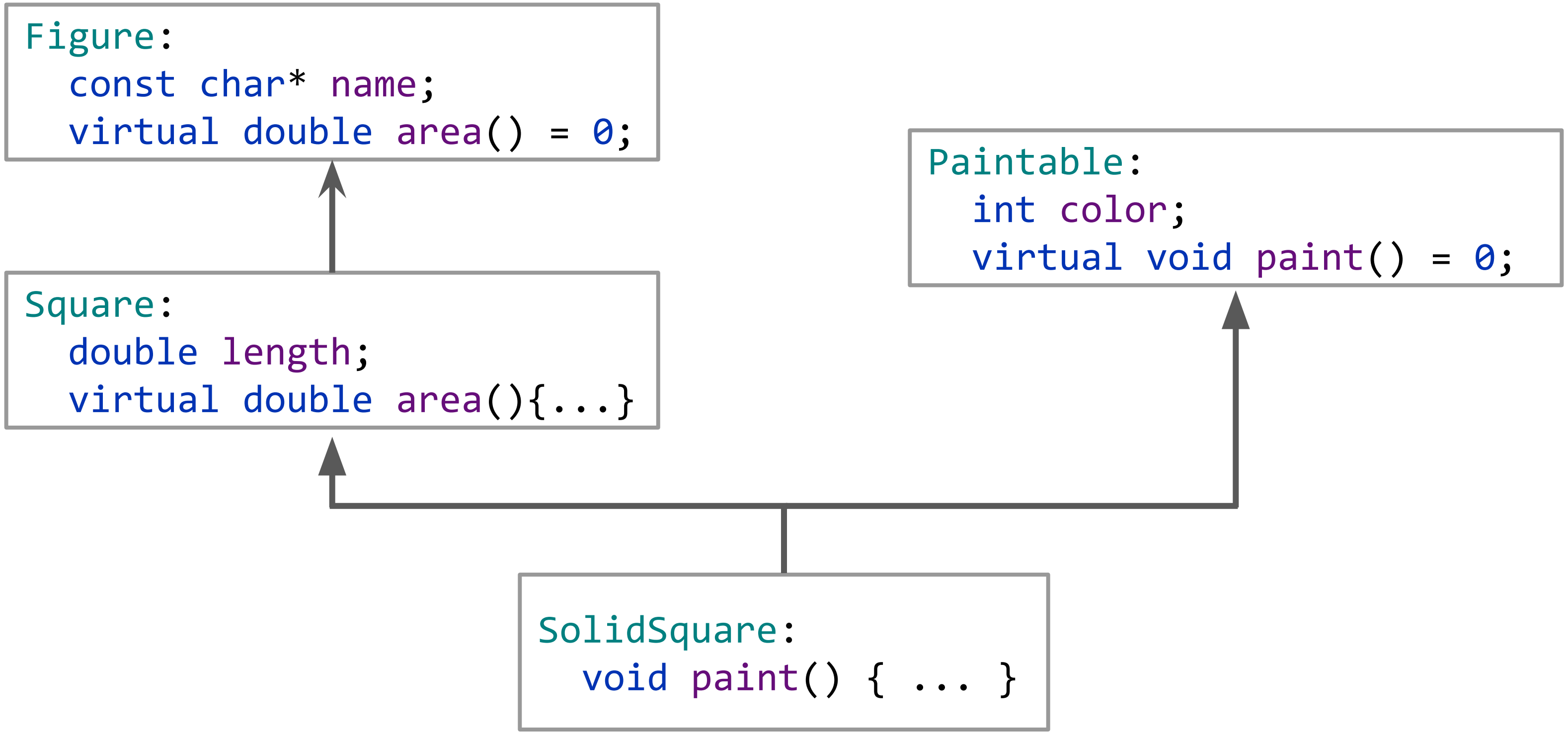


```
Figure:
  const char* name;
  virtual double area() = 0;
```

```
Square:
  double length;
  virtual double area(){...}
```

```
Paintable:
  int color;
  virtual void paint() = 0;
```

Multiple Inheritance



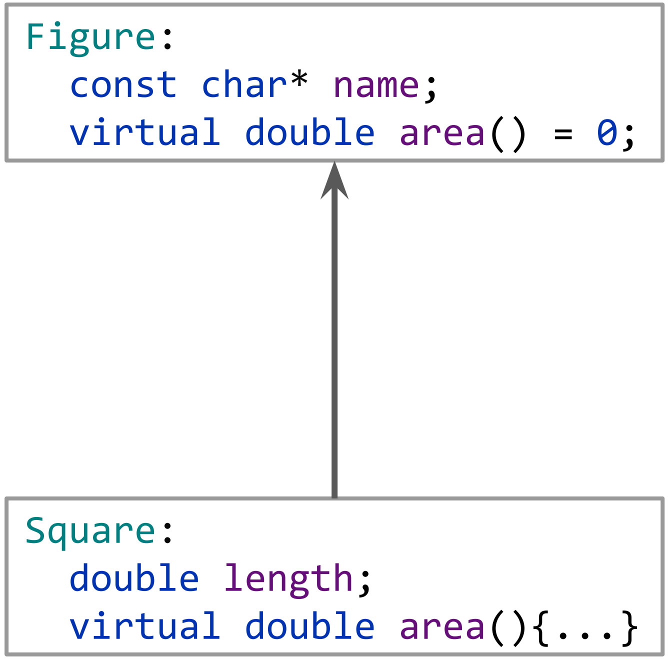
```
Figure:
  const char* name;
  virtual double area() = 0;
```

```
Square:
  double length;
  virtual double area(){...}
```

```
Paintable:
  int color;
  virtual void paint() = 0;
```

```
SolidSquare:
  void paint() { ... }
```

```
class Figure {
protected:
    const char* name;
public:
    Figure(const char* name): name(name) {}
    virtual double area() = 0;
};


class Square: public Figure {
protected:
    double length;
public:
    Square(double l):
                Figure("Square"), length(l) {}
    double area() {
         return length*length;
    }
};
```



```
Figure:
  const char* name;
  virtual double area() = 0;

Square:
  double length;
  virtual double area(){...}
```

```cpp
class Paintable {
protected:
    int color;
public:
    Paintable(int color): color(color) {}
    virtual void paint() = 0;
};
```

```
Paintable:
  int color;
  virtual void paint() = 0;
```

```cpp
class SolidSquare : public Square, public Paintable {
public:
   SolidSquare(double length, int color) :
         Square(length), Paintable(color) {}

   void paint() {
      std::cout << "We are painting square with length = "
               << this->length
               << " and color = "
               << this->color << std::endl;
   }
};
```

```
SolidSquare:
  void paint() { ... }
```

```cpp
void paintP(Paintable* test) {
    test->paint();
}

void printArea(Figure* test) {
    std::cout << "Area = "
              << test->area()
              << std::endl;
}


int main() {
    SolidSquare bs(10, 1);
    printArea(&bs);
    paintP(&bs);
}
```

Substitution of derived class (pointers or references to them) instead of base one still works!

```cpp
void paintP(Paintable* test) {
    test->paint();
}

void printArea(Figure* test) {
    std::cout << "Area = "
              << test->area()
              << std::endl;
}


int main() {
    SolidSquare bs(10, 1);
    printArea(&bs);
    paintP(&bs);
}
```

Substitution of derived class (pointers or references to them) instead of base one still works!

As well as virtual calls.

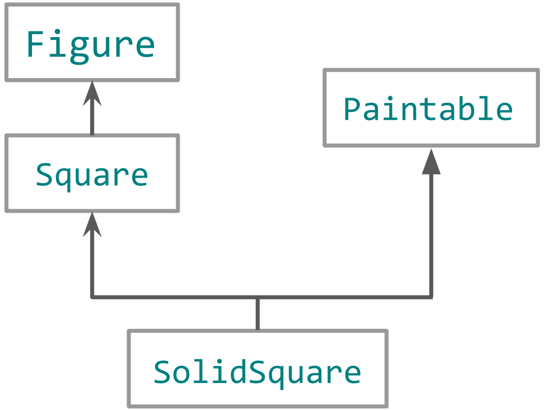
```
void paintP(Paintable* test) {
    test->paint();
}

void printArea(Figure* test) {
    std::cout << "Area = "
              << test->area()
              << std::endl;
}


int main() {
    SolidSquare bs(10, 1);
    printArea(&bs);
    paintP(&bs);
}
```

Substitution of derived class (pointers or references to them) instead of base one still works!

As well as virtual calls.

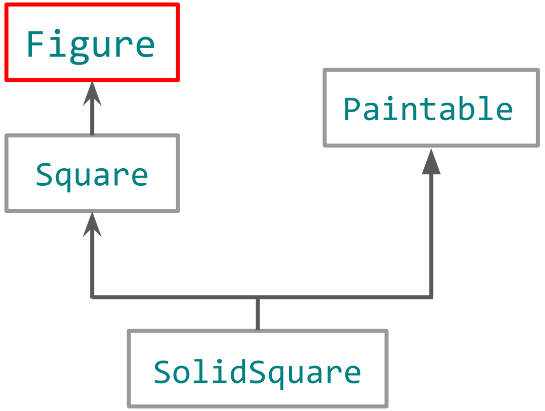
But do you see any complications here?

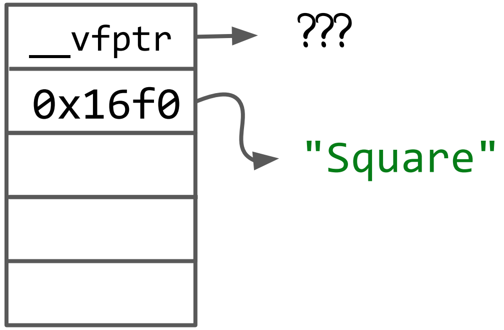
```
void paintP(Paintable* test) {
    test->paint();
}

void printArea(Figure* test) {
    std::cout << "Area = "
              << test->area()
              << std::endl;
}


int main() {
    SolidSquare bs(10, 1);
    printArea(&bs);
    paintP(&bs);
}
```



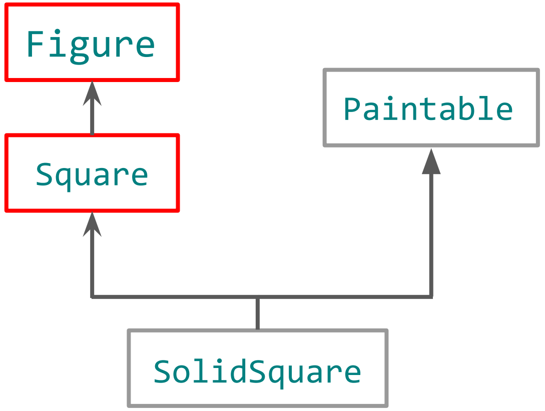
```
class SolidSquare : public Square, public Paintable {
public:
   SolidSquare(double length, int color) :
           Square(length), Paintable(color) {}

   void paint() {
       std::cout << "We are painting square with length = "
                 << this->length
                 << " and color = "
                 << this->color << std::endl;
   }
};
```

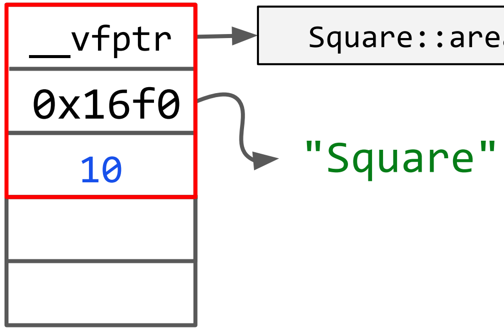
```
void paintP(Paintable* test) {
    test->paint();
}

void printArea(Figure* test) {
    std::cout << "Area = "
              << test->area()
              << std::endl;
}


int main() {
    SolidSquare bs(10, 1);
    printArea(&bs);
    paintP(&bs);
}
```



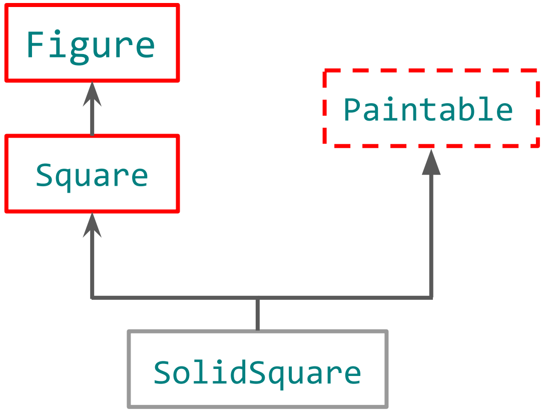


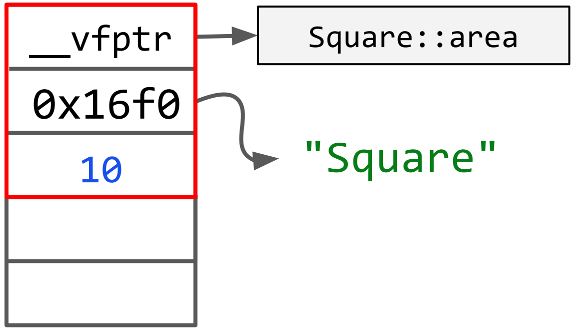
```
void paintP(Paintable* test) {
    test->paint();
}

void printArea(Figure* test) {
    std::cout << "Area = "
              << test->area()
              << std::endl;
}


int main() {
    SolidSquare bs(10, 1);
    printArea(&bs);
    paintP(&bs);
}
```



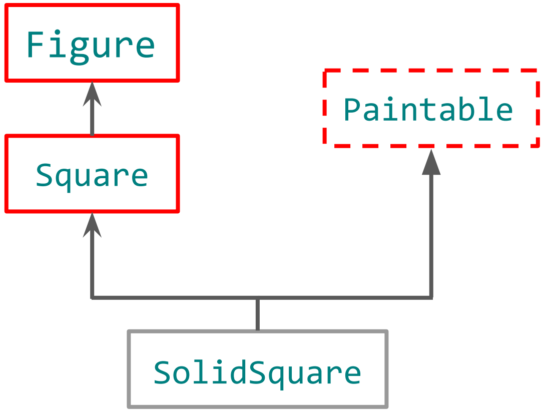


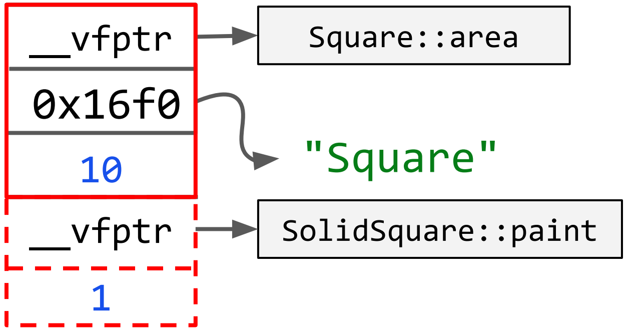
```
void paintP(Paintable* test) {
    test->paint();
}

void printArea(Figure* test) {
    std::cout << "Area = "
              << test->area()
              << std::endl;
}


int main() {
    SolidSquare bs(10, 1);
    printArea(&bs);
    paintP(&bs);
}
```



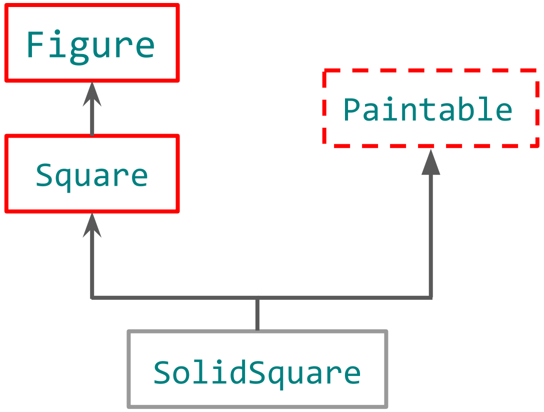


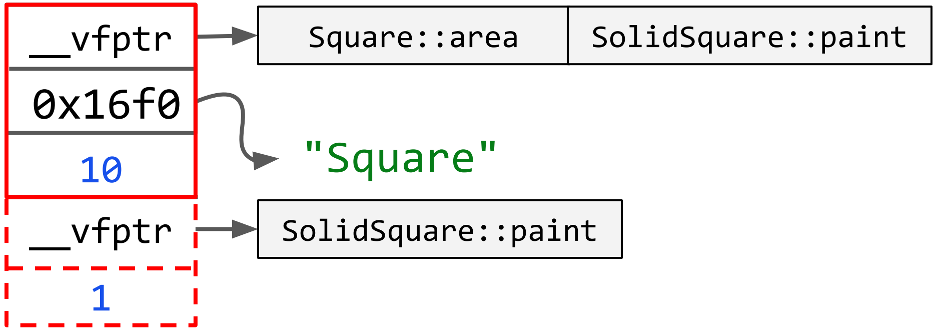
```
void paintP(Paintable* test) {
    test->paint();
}

void printArea(Figure* test) {
    std::cout << "Area = "
              << test->area()
              << std::endl;
}


int main() {
    SolidSquare bs(10, 1);
    printArea(&bs);
    paintP(&bs);
}
```





```
void paintP(Paintable* test) {
    test->paint();
}

void printArea(Figure* test) {
    std::cout << "Area = "
              << test->area()
              << std::endl;
}


int main() {
    SolidSquare bs(10, 1);
    printArea(&bs);
    paintP(&bs);
}
```





```cpp
class SolidSquare : public Square,
                    public Paintable {
public:
   SolidSquare(double length, int color) :
           Square(length),
           Paintable(color) {}

   …
};
```

→

```
SolidSquare::SolidSquare(double, int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     QWORD PTR [rbp-8], rdi
        movsd   QWORD PTR [rbp-16], xmm0
        mov     DWORD PTR [rbp-20], esi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdx, QWORD PTR [rbp-16]
        movq    xmm0, rdx
        mov     rdi, rax
        call    Square::Square(double)
        mov     rax, QWORD PTR [rbp-8]
        lea     rdx, [rax+24]
        mov     eax, DWORD PTR [rbp-20]
        mov     esi, eax
        mov     rdi, rdx
        call    Paintable::Paintable(int)
        mov     edx, OFFSET FLAT:vtable for SolidSquare+16
        mov     rax, QWORD PTR [rbp-8]
        mov     QWORD PTR [rax], rdx
        mov     edx, OFFSET FLAT:vtable for SolidSquare+48
        mov     rax, QWORD PTR [rbp-8]
        mov     QWORD PTR [rax+24], rdx
        nop
        leave
        ret
```

https://godbolt.org/z/6hcT5zo18

```
SolidSquare::SolidSquare(double, int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     QWORD PTR [rbp-8], rdi
        movsd   QWORD PTR [rbp-16], xmm0
        mov     DWORD PTR [rbp-20], esi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdx, QWORD PTR [rbp-16]
        movq    xmm0, rdx
        mov     rdi, rax
        call    Square::Square(double)
        mov     rax, QWORD PTR [rbp-8]
        lea     rdx, [rax+24]
        mov     eax, DWORD PTR [rbp-20]
        mov     esi, eax
        mov     rdi, rdx
        call    Paintable::Paintable(int)
        mov     edx, OFFSET FLAT:vtable for SolidSquare+16
        mov     rax, QWORD PTR [rbp-8]
        mov     QWORD PTR [rax], rdx
        mov     edx, OFFSET FLAT:vtable for SolidSquare+48
        mov     rax, QWORD PTR [rbp-8]
        mov     QWORD PTR [rax+24], rdx
        nop
        leave
        ret
```

building
first base

```
class SolidSquare : public Square,
                    public Paintable {
public:
   SolidSquare(double length, int color) :
           Square(length),
           Paintable(color) {}

   …
};
```

https://godbolt.org/z/6hcT5zo18

[rbp-8] contains address
of object to initialize

building
first base

```
class SolidSquare : public Square,
                    public Paintable {
public:
   SolidSquare(double length, int color) :
        Square(length),
        Paintable(color) {}

   …
};
```

```
SolidSquare::SolidSquare(double, int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     QWORD PTR [rbp-8], rdi
        movsd   QWORD PTR [rbp-16], xmm0
        mov     DWORD PTR [rbp-20], esi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdx, QWORD PTR [rbp-16]
        movq    xmm0, rdx
        mov     rdi, rax
        call    Square::Square(double)
        mov     rax, QWORD PTR [rbp-8]
        lea     rdx, [rax+24]
        mov     eax, DWORD PTR [rbp-20]
        mov     esi, eax
        mov     rdi, rdx
        call    Paintable::Paintable(int)
        mov     edx, OFFSET FLAT:vtable for SolidSquare+16
        mov     rax, QWORD PTR [rbp-8]
        mov     QWORD PTR [rax], rdx
        mov     edx, OFFSET FLAT:vtable for SolidSquare+48
        mov     rax, QWORD PTR [rbp-8]
        mov     QWORD PTR [rax+24], rdx
        nop
        leave
        ret
```

https://godbolt.org/z/6hcT5zo18

[rbp-8] contains address
of object to initialize

```
class SolidSquare : public Square,
                    public Paintable {
public:
   SolidSquare(double length, int color) :
          Square(length),
          Paintable(color) {}

   …
};
```

building
second base

```
SolidSquare::SolidSquare(double, int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     QWORD PTR [rbp-8], rdi
        movsd   QWORD PTR [rbp-16], xmm0
        mov     DWORD PTR [rbp-20], esi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdx, QWORD PTR [rbp-16]
        movq    xmm0, rdx
        mov     rdi, rax
        call    Square::Square(double)
        mov     rax, QWORD PTR [rbp-8]
        lea     rdx, [rax+24]
        mov     eax, DWORD PTR [rbp-20]
        mov     esi, eax
        mov     rdi, rdx
        call    Paintable::Paintable(int)
        mov     edx, OFFSET FLAT:vtable for SolidSquare+16
        mov     rax, QWORD PTR [rbp-8]
        mov     QWORD PTR [rax], rdx
        mov     edx, OFFSET FLAT:vtable for SolidSquare+48
        mov     rax, QWORD PTR [rbp-8]
        mov     QWORD PTR [rax+24], rdx
        nop
        leave
        ret
```

https://godbolt.org/z/6hcT5zo18

rdi contains address of object to initialize +24

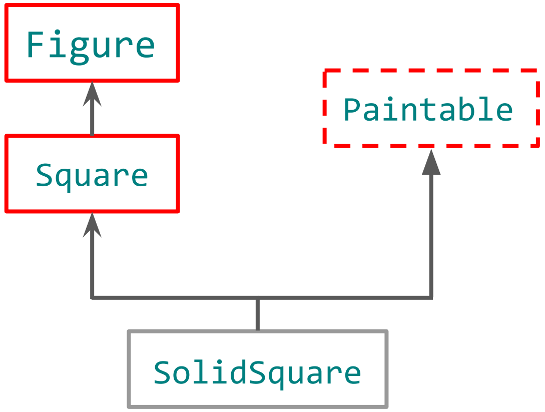
```
class SolidSquare : public Square,
                    public Paintable {
public:
   SolidSquare(double length, int color) :
          Square(length),
          Paintable(color) {}

   …
};
```

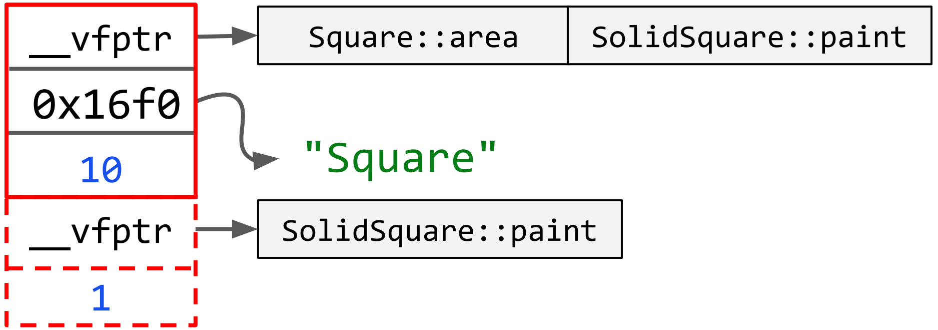
building second base

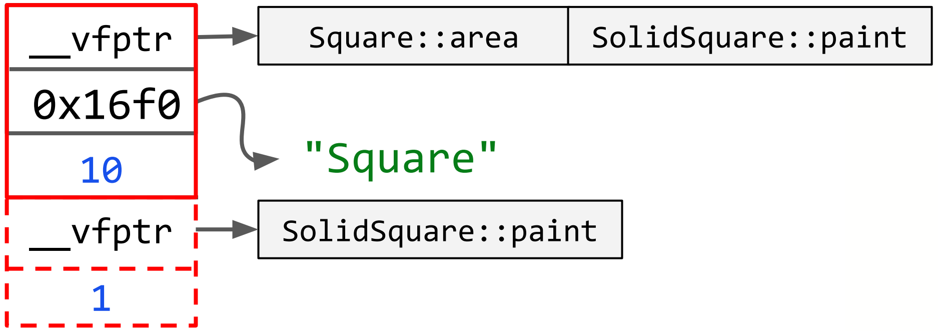
```
SolidSquare::SolidSquare(double, int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     QWORD PTR [rbp-8], rdi
        movsd   QWORD PTR [rbp-16], xmm0
        mov     DWORD PTR [rbp-20], esi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdx, QWORD PTR [rbp-16]
        movq    xmm0, rdx
        mov     rdi, rax
        call    Square::Square(double)
        mov     rax, QWORD PTR [rbp-8]
        lea     rdx, [rax+24]
        mov     eax, DWORD PTR [rbp-20]
        mov     esi, eax
        mov     rdi, rdx
        call    Paintable::Paintable(int)
        mov     edx, OFFSET FLAT:vtable for SolidSquare+16
        mov     rax, QWORD PTR [rbp-8]
        mov     QWORD PTR [rax], rdx
        mov     edx, OFFSET FLAT:vtable for SolidSquare+48
        mov     rax, QWORD PTR [rbp-8]
        mov     QWORD PTR [rax+24], rdx
        nop
        leave
        ret
```

https://godbolt.org/z/6hcT5zo18

```
void paintP(Paintable* test) {
    test->paint();
}

void printArea(Figure* test) {
    std::cout << "Area = "
              << test->area()
              << std::endl;
}


int main() {
    SolidSquare bs(10, 1);
    printArea(&bs);
    paintP(&bs);
}
```





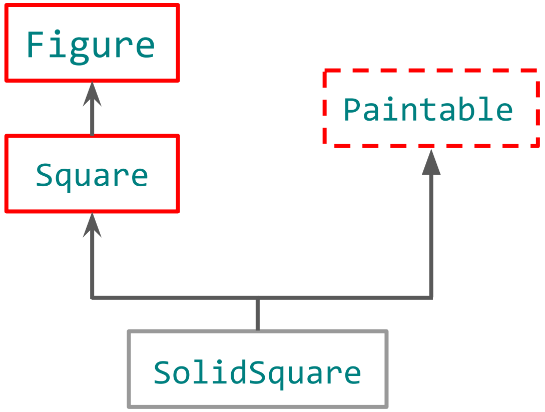


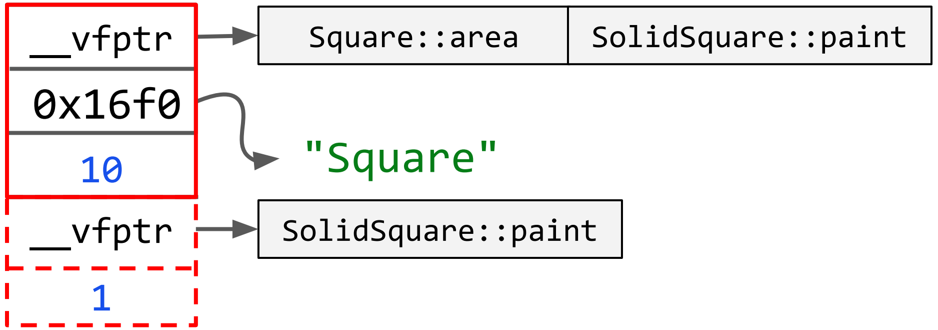
```
SolidSquare::SolidSquare(double, int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     QWORD PTR [rbp-8], rdi
        movsd   QWORD PTR [rbp-16], xmm0
        mov     DWORD PTR [rbp-20], esi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdx, QWORD PTR [rbp-16]
        movq    xmm0, rdx
        mov     rdi, rax
        call    Square::Square(double)
        mov     rax, QWORD PTR [rbp-8]
        lea     rdx, [rax+24]
        mov     eax, DWORD PTR [rbp-20]
        mov     esi, eax
        mov     rdi, rdx
        call    Paintable::Paintable(int)
        mov     edx, OFFSET FLAT:vtable for SolidSquare+16
        mov     rax, QWORD PTR [rbp-8]
        mov     QWORD PTR [rax], rdx
        mov     edx, OFFSET FLAT:vtable for SolidSquare+48
        mov     rax, QWORD PTR [rbp-8]
        mov     QWORD PTR [rax+24], rdx
        nop
        leave
        ret
```

updating VMTs →

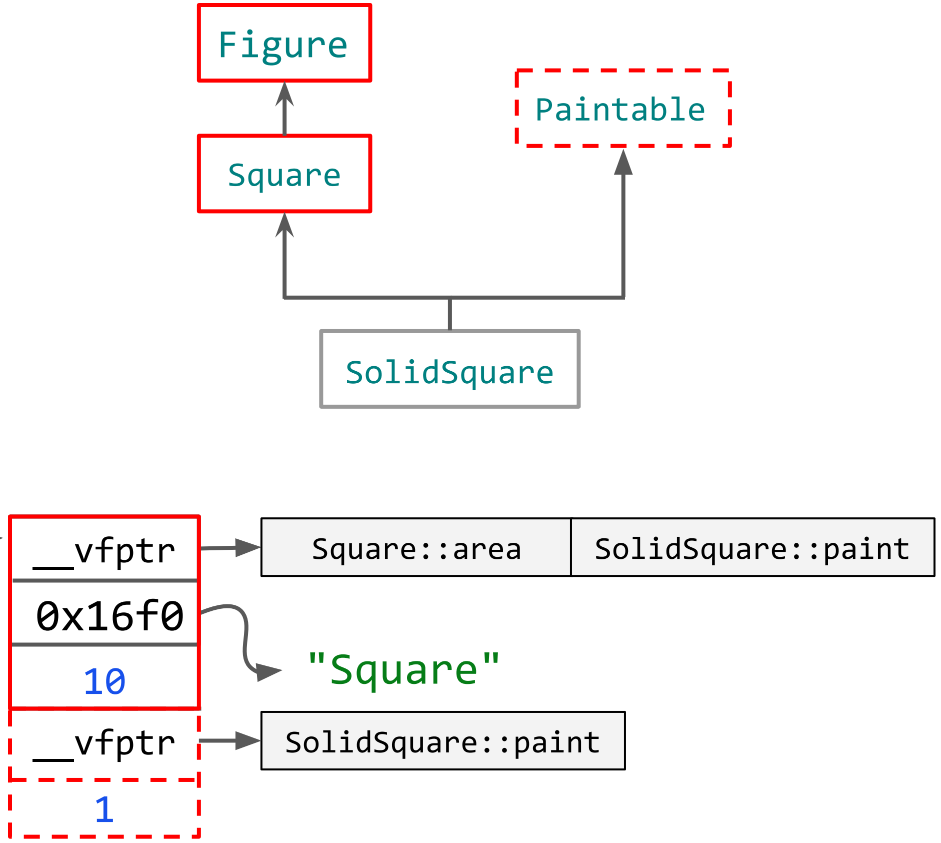
https://godbolt.org/z/6hcT5zo18

```
void paintP(Paintable* test) {
    test->paint();
}

void printArea(Figure* test) {
    std::cout << "Area = "
              << test->area()
              << std::endl;
}


int main() {
    SolidSquare bs(10, 1);
    printArea(&bs);
    paintP(&bs);
}
```

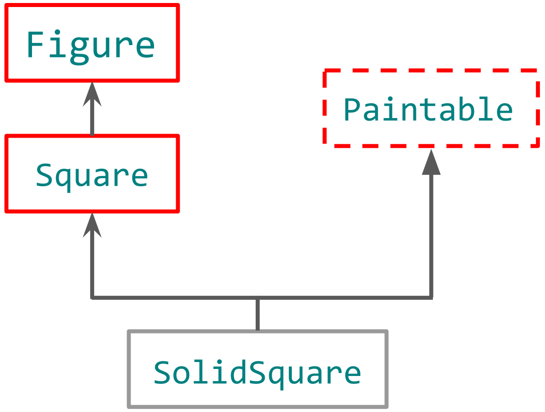


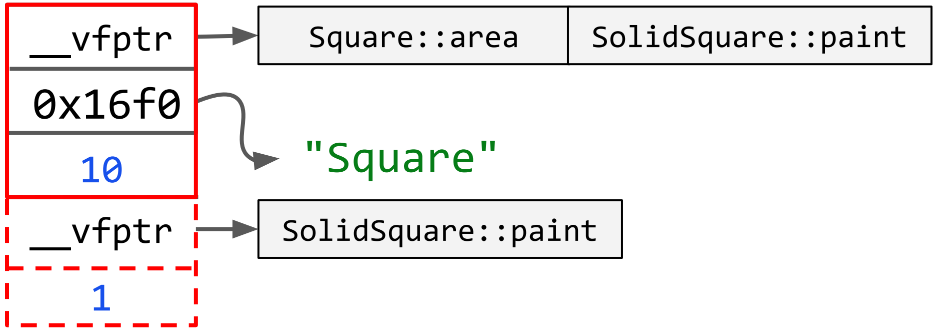


```
void paintP(Paintable* test) {
    test->paint();
}

void printArea(Figure* test) {
    std::cout << "Area = "
              << test->area()
              << std::endl;
}


int main() {
    SolidSquare bs(10, 1);
    printArea(&bs);
    paintP(&bs);
}
```

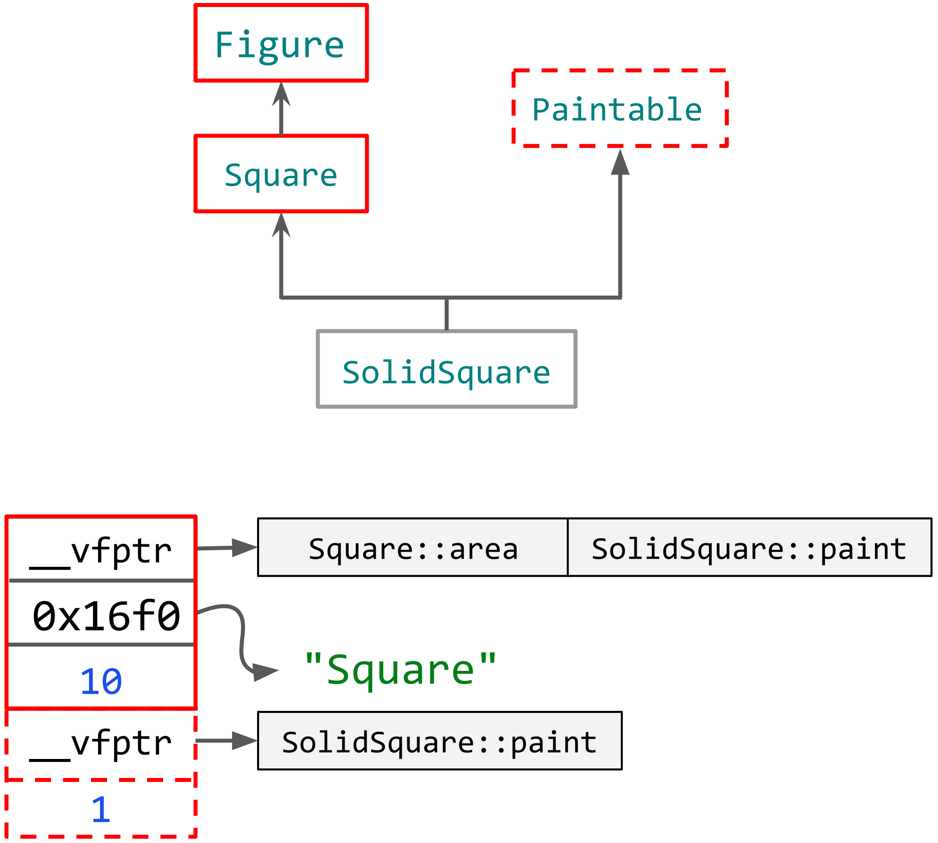


```
void paintP(Paintable* test) {
    test->paint();
}

void printArea(Figure* test) {
    std::cout << "Area = "
              << test->area()
              << std::endl;
}


int main() {
    SolidSquare bs(10, 1);
    printArea(&bs);
    paintP(&bs);
}
```

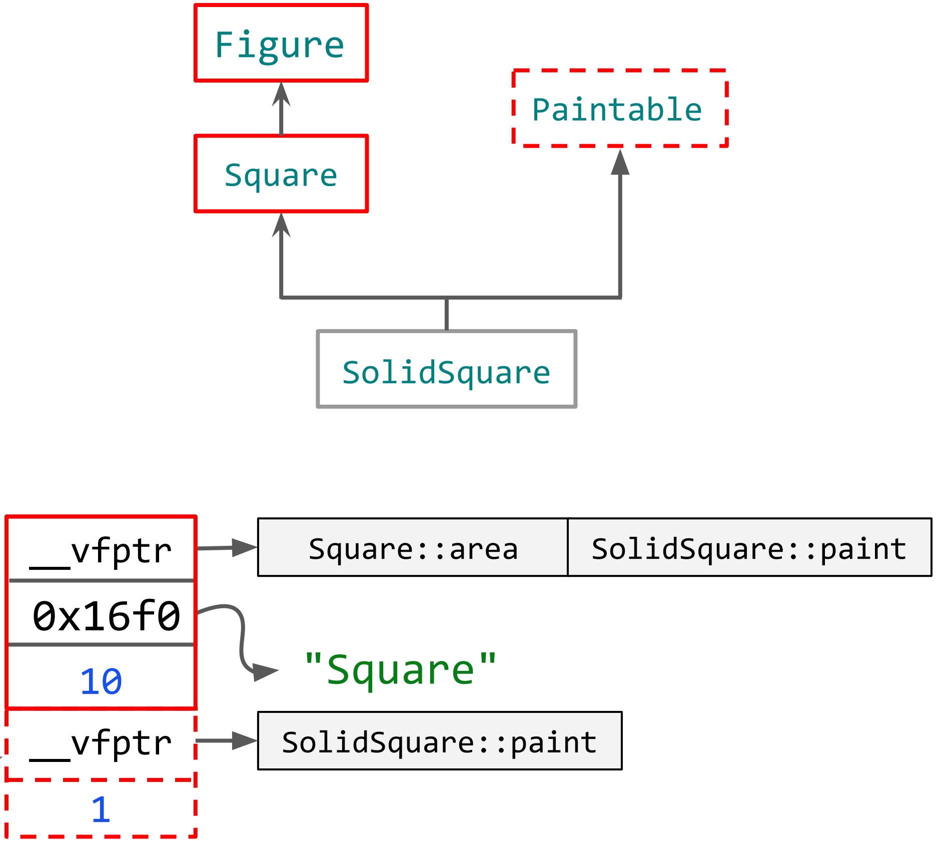




```
void paintP(Paintable* test) {
    test->paint();
}

void printArea(Figure* test) {
    std::cout << "Area = "
              << test->area()
              << std::endl;
}


int main() {
    SolidSquare bs(10, 1);
    printArea(&bs);
    paintP(&bs);
}
```

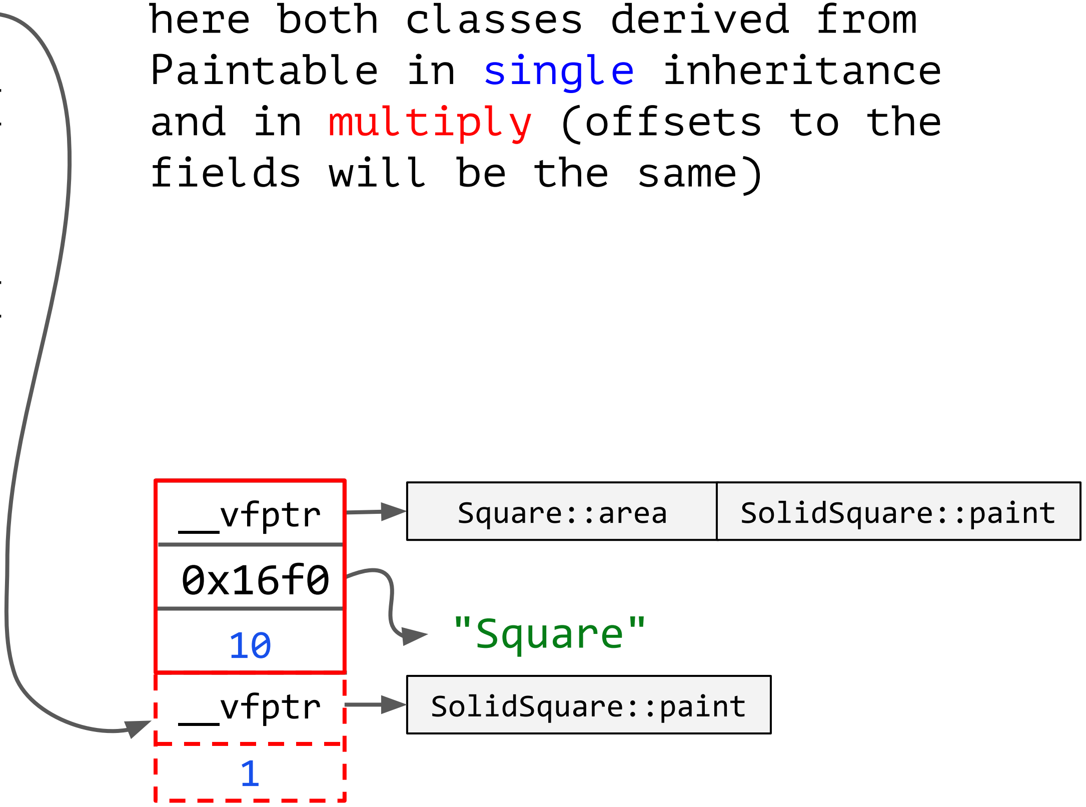


```
void paintP(Paintable* test) {
    test->paint();
}

void printArea(Figure* test) {
    std::cout << "Area = "
              << test->area()
              << std::endl;
}


int main() {
    SolidSquare bs(10, 1);
    printArea(&bs);
    paintP(&bs);
}
```



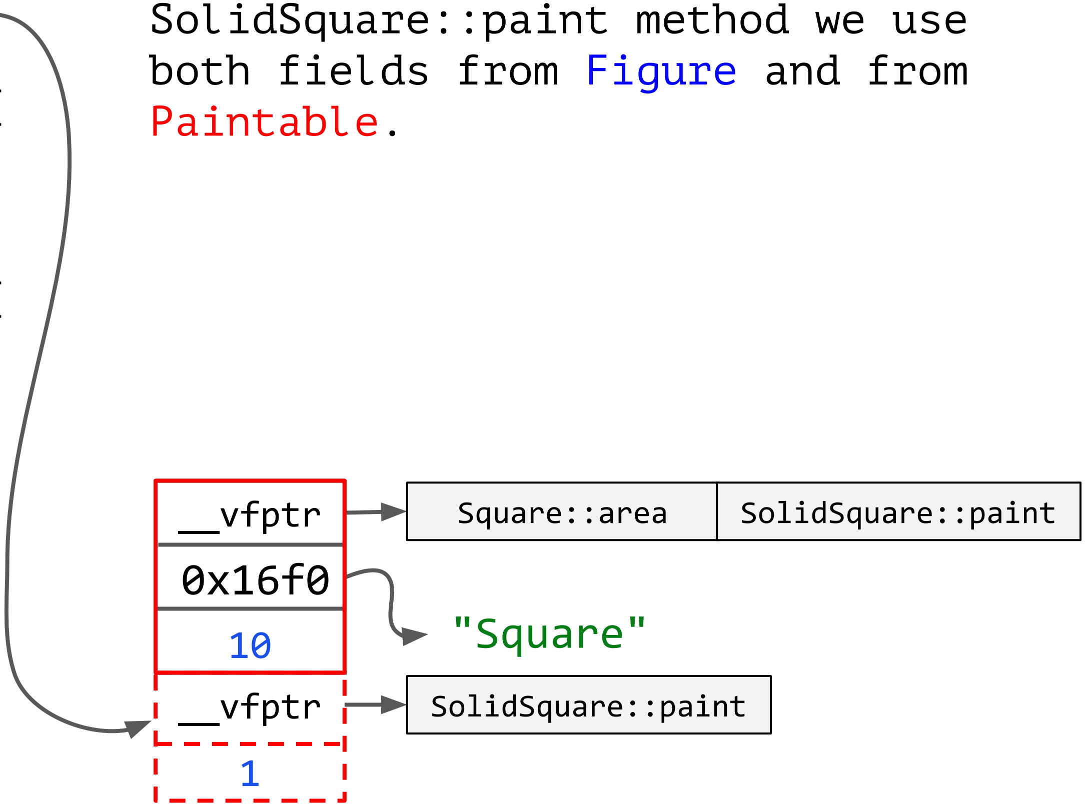
```
void paintP(Paintable* test) {
    test->paint();
}

void printArea(Figure* test) {
    std::cout << "Area = "
              << test->area()
              << std::endl;
}


int main() {
    SolidSquare bs(10, 1);
    printArea(&bs);
    paintP(&bs);
}
```

Such system allows you to pass here both classes derived from Paintable in single inheritance and in multiply (offsets to the fields will be the same)



```
void paintP(Paintable* test) {
    test->paint();
}

void printArea(Figure* test) {
    std::cout << "Area = "
              << test->area()
              << std::endl;
}


int main() {
    SolidSquare bs(10, 1);
    printArea(&bs);
    paintP(&bs);
}
```

But there is a problem: in SolidSquare::paint method we use both fields from Figure and from Paintable.

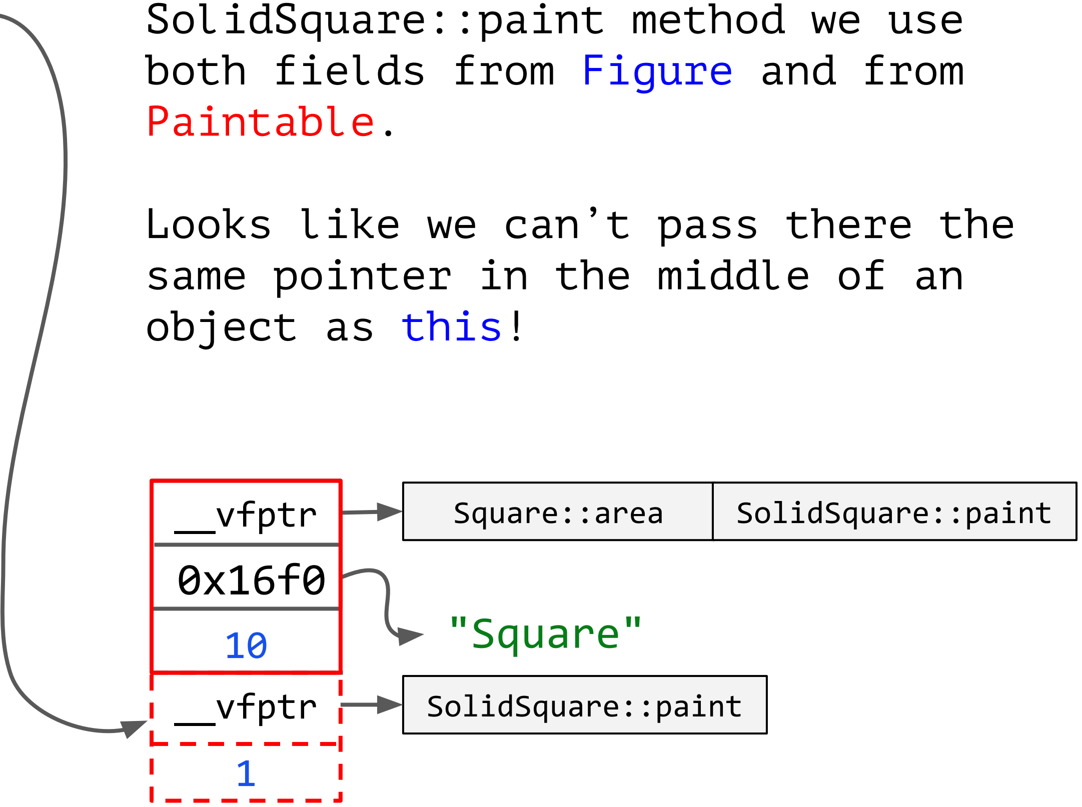


```
class SolidSquare : public Square, public Paintable {
public:
   SolidSquare(double length, int color) :
           Square(length), Paintable(color) {}

   void paint() {
       std::cout << "We are painting square with length = "
                 << this->length ←—— which offset will it have?
                 << " and color = "
                 << this->color << std::endl;
   }
};
```

```
void paintP(Paintable* test) {
    test->paint();
}

void printArea(Figure* test) {
    std::cout << "Area = "
              << test->area()
              << std::endl;
}


int main() {
    SolidSquare bs(10, 1);
    printArea(&bs);
    paintP(&bs);
}
```

But there is a problem: in SolidSquare::paint method we use both fields from Figure and from Paintable.

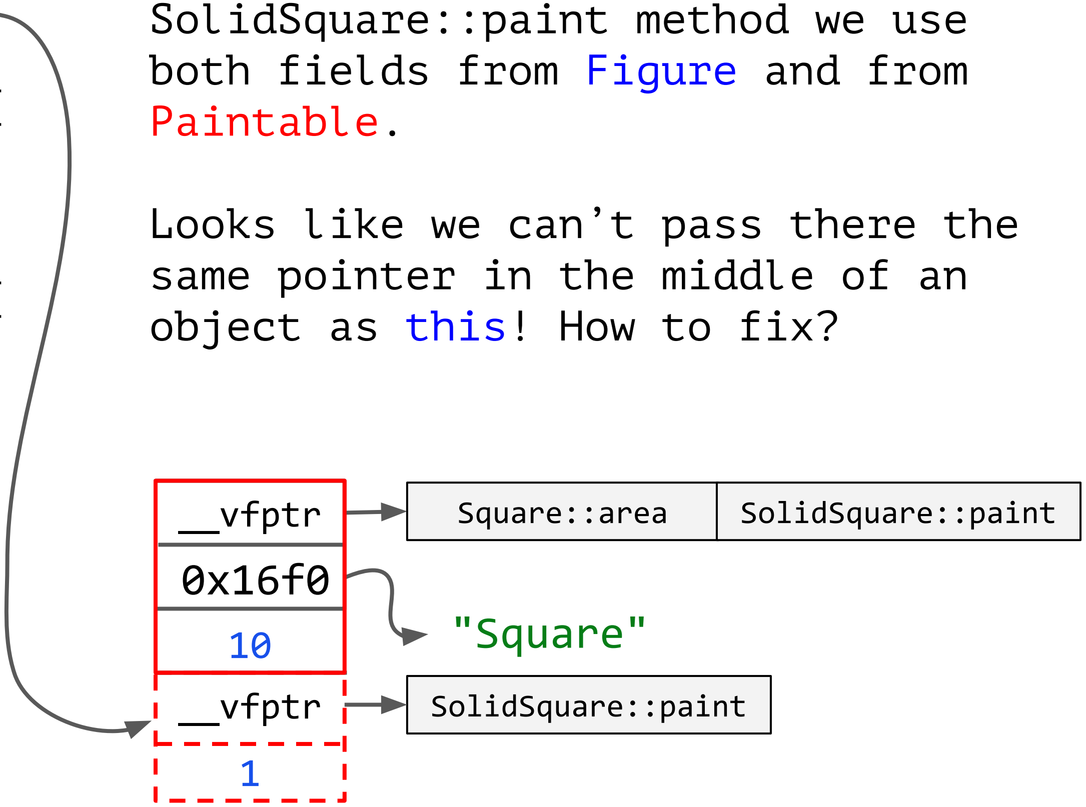
Looks like we can't pass there the same pointer in the middle of an object as this!



```
void paintP(Paintable* test) {
    test->paint();
}

void printArea(Figure* test) {
    std::cout << "Area = "
              << test->area()
              << std::endl;
}


int main() {
    SolidSquare bs(10, 1);
    printArea(&bs);
    paintP(&bs);
}
```

But there is a problem: in SolidSquare::paint method we use both fields from Figure and from Paintable.

Looks like we can't pass there the same pointer in the middle of an object as this! How to fix?

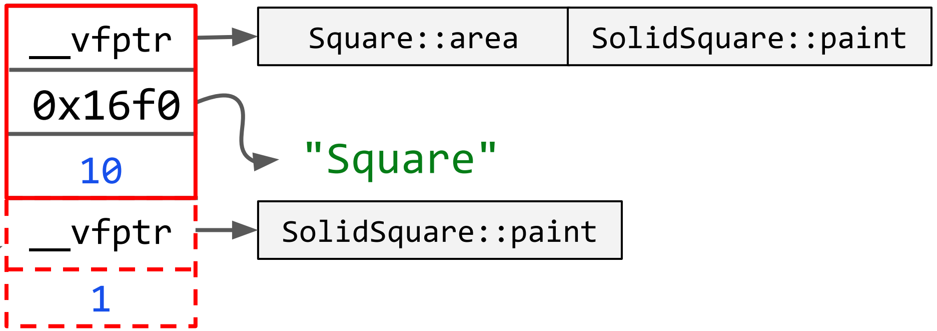


```
void paintP(Paintable* test) {
    test->paint();
}

void printArea(Figure* test) {
    std::cout << "Area = "
              << test->area()
              << std::endl;
}


int main() {
    SolidSquare bs(10, 1);
    printArea(&bs);
    paintP(&bs);
}
```

But there is a problem: in SolidSquare::paint method we use both fields from Figure and from Paintable.

Looks like we can't pass there the same pointer in the middle of an object as this! How to fix?

We need an offset backward to head!



```
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

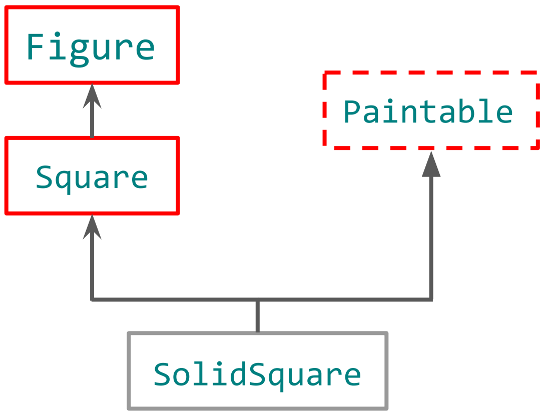
```
Person::Person(char const*, unsigned long)
[base object constructor]:

       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

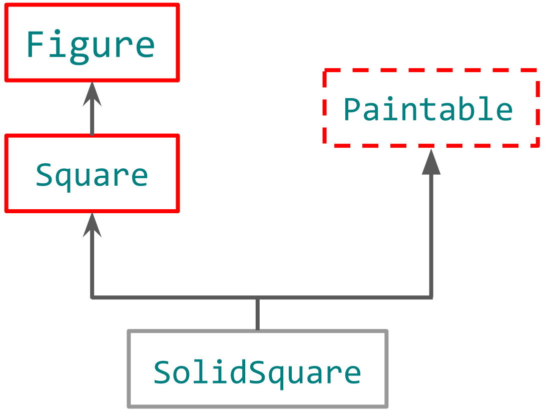
smth interesting we will discuss later

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

static data



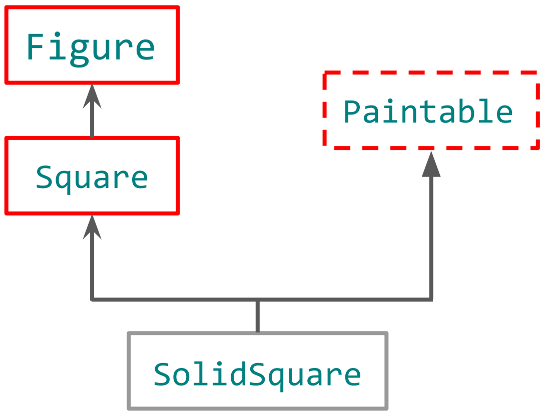
```
vtable for SolidSquare:
        .quad   0
        .quad   typeinfo for SolidSquare
        .quad   Square::area()
        .quad   SolidSquare::paint()
        .quad   -24
        .quad   typeinfo for SolidSquare
        .quad   non-virtual thunk to SolidSquare::paint()
```



static data

```
vtable for SolidSquare:
        .quad   0
        .quad   typeinfo for SolidSquare
        .quad   Square::area()
        .quad   SolidSquare::paint()
        .quad   -24
        .quad   typeinfo for SolidSquare
        .quad   non-virtual thunk to SolidSquare::paint()
```

This part of VMT is actually used for Paintable part of SolidSquare

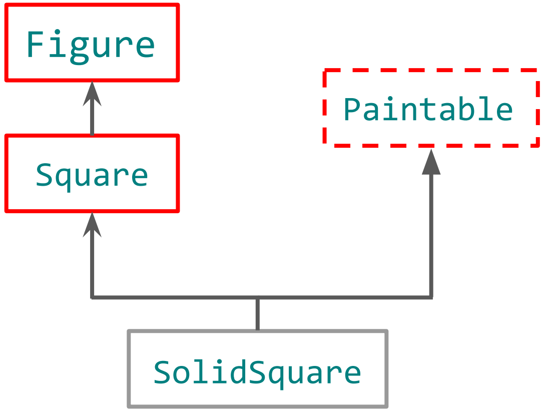


static data

```
vtable for SolidSquare:
        .quad   0
        .quad   typeinfo for SolidSquare
        .quad   Square::area()
        .quad   SolidSquare::paint()
        .quad   -24
        .quad   typeinfo for SolidSquare
        .quad   non-virtual thunk to SolidSquare::paint()
```

offset
to head

This part of VMT is actually used
for Paintable part of SolidSquare



static data

```
vtable for SolidSquare:
        .quad   0
        .quad   typeinfo for SolidSquare
        .quad   Square::area()
        .quad   SolidSquare::paint()
        .quad   -24
        .quad   typeinfo for SolidSquare
        .quad   non-virtual thunk to SolidSquare::paint()
```

offset to head

This part of VMT is actually used for Paintable part of SolidSquare

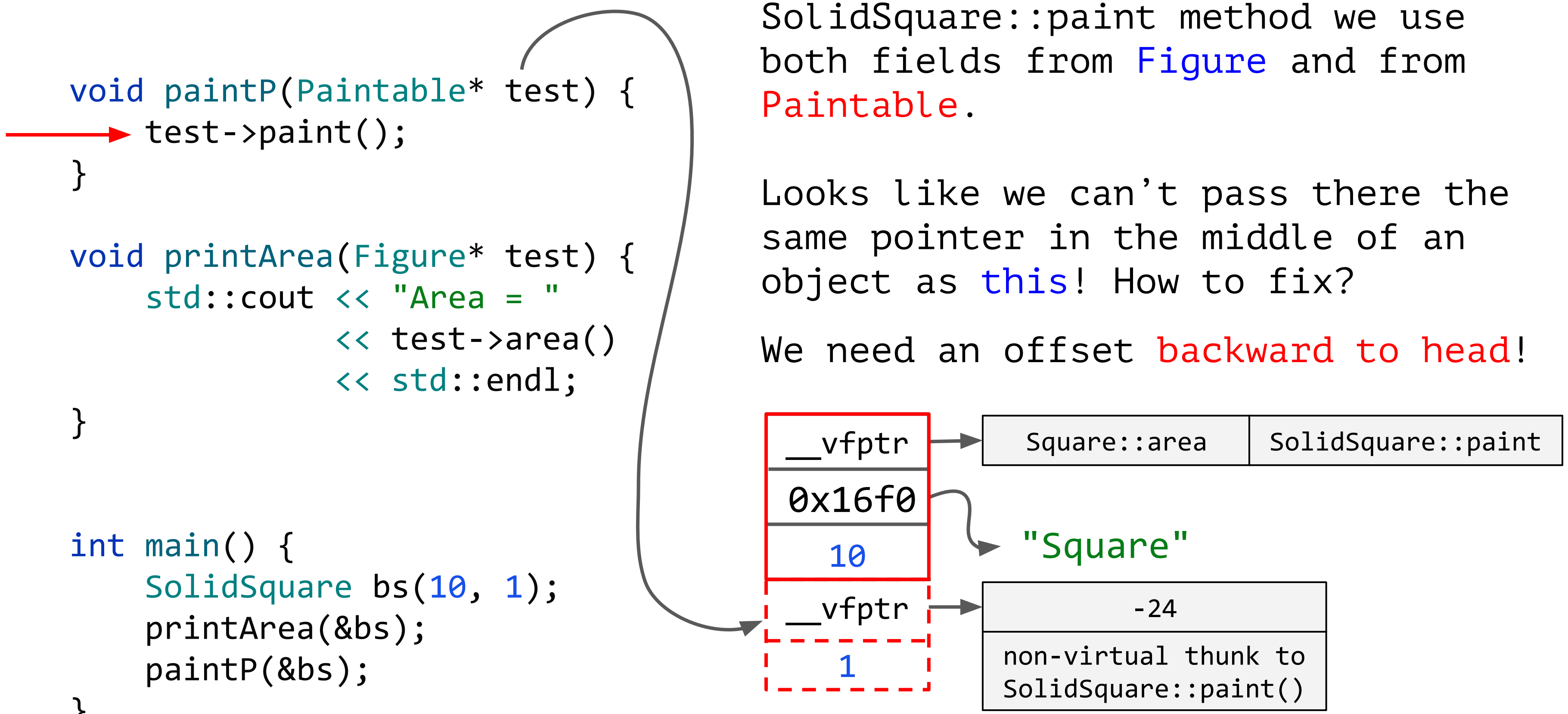
this means "call SolidSquare::paint(), but to get this add -24 to current this"

```
void paintP(Paintable* test) {
    test->paint();
}

void printArea(Figure* test) {
    std::cout << "Area = "
              << test->area()
              << std::endl;
}


int main() {
    SolidSquare bs(10, 1);
    printArea(&bs);
    paintP(&bs);
}
```

But there is a problem: in SolidSquare::paint method we use both fields from Figure and from Paintable.

Looks like we can't pass there the same pointer in the middle of an object as this! How to fix?

We need an offset backward to head!

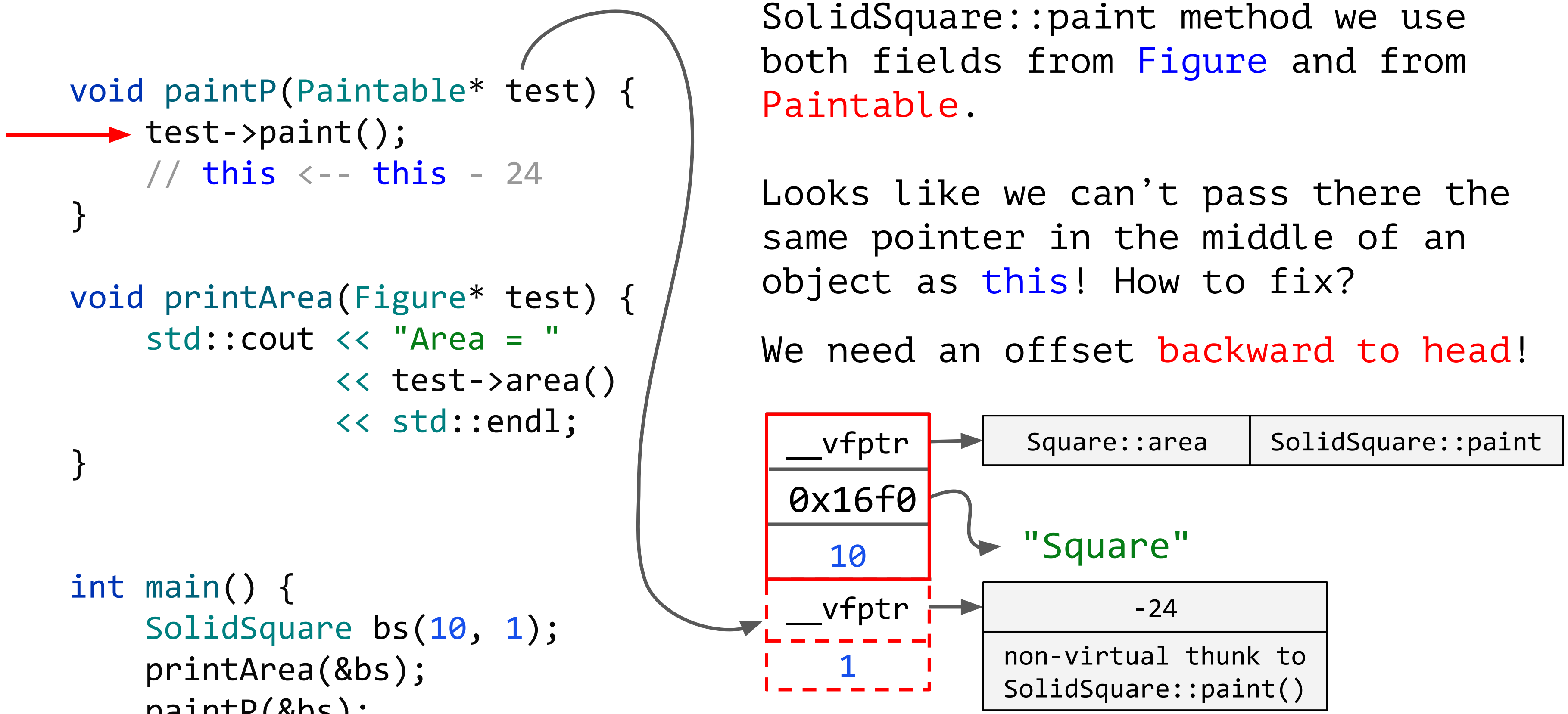


```
void paintP(Paintable* test) {
    test->paint();
    // this <-- this - 24
}

void printArea(Figure* test) {
    std::cout << "Area = "
              << test->area()
              << std::endl;
}


int main() {
    SolidSquare bs(10, 1);
    printArea(&bs);
    paintP(&bs);
}
```

But there is a problem: in SolidSquare::paint method we use both fields from Figure and from Paintable.

Looks like we can't pass there the same pointer in the middle of an object as this! How to fix?

We need an offset backward to head!



Multiple inheritance: *so far*

- Virtual calls can be even more expensive!
- Objects are even more fatty.
- VMTs contains new meta-information

Multiple inheritance: *so far*

- Virtual calls can be even more expensive!
- Objects are even more fatty.
- VMTs contains new meta-information

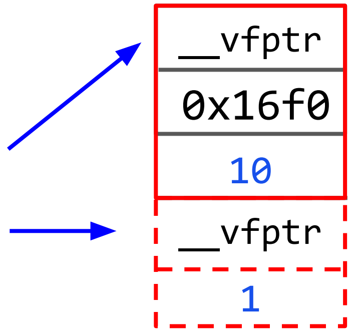
- Pointers to the base class and to the derived class can be very different (not the first time for us, though)

Multiple inheritance: so far

- Virtual calls can be even more expensive!
- Objects are even more fatty.
- VMTs contains new meta-information

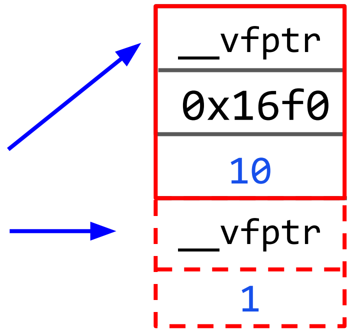
- Pointers to the base class and to the derived class can be very different (not the first time for us, though)

```
int main() {
    SolidSquare bs(10, 1);
    Square* s = &bs;
    printf("%p\n", s);           ----> 0x7ffc85524eb0
    Paintable* p = &bs;
    printf("%p\n", p);           ----> 0x7ffc85524ec8
}
```



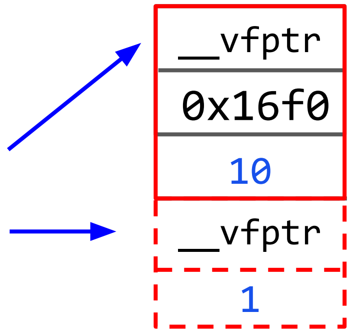
Multiple inheritance: so far

```
int main() {
    SolidSquare bs(10, 1);
    Square* s = &bs;
    printf("%p\n", s);         ----> 0x7ffc85524eb0
    Paintable* p = &bs;
    printf("%p\n", p);         ----> 0x7ffc85524ec8

    SolidSquare* sq = p;       // ???
    printf("%p\n", sq);
}
```



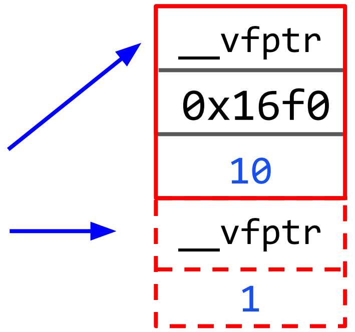
Multiple inheritance: so far

```
int main() {
    SolidSquare bs(10, 1);
    Square* s = &bs;
    printf("%p\n", s);        ----> 0x7ffc85524eb0
    Paintable* p = &bs;
    printf("%p\n", p);        ----> 0x7ffc85524ec8

    SolidSquare* sq = p;      // compilation error
    printf("%p\n", sq);
}
```



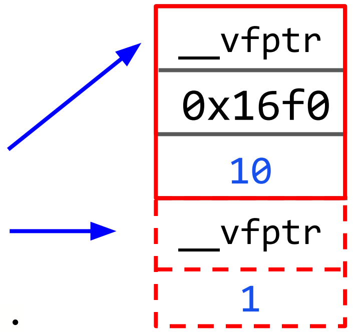
Multiple inheritance: so far

```
int main() {
    SolidSquare bs(10, 1);
    Square* s = &bs;
    printf("%p\n", s);        ----> 0x7ffc85524eb0
    Paintable* p = &bs;
    printf("%p\n", p);        ----> 0x7ffc85524ec8

    SolidSquare* sq = p;      // compilation error
    printf("%p\n", sq);
}
```



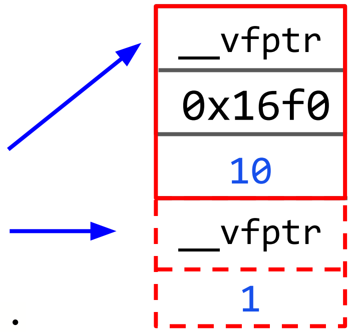
Multiple inheritance: so far

```
int main() {
    SolidSquare bs(10, 1);
    Square* s = &bs;
    printf("%p\n", s);        ----> 0x7ffc85524eb0
    Paintable* p = &bs;
    printf("%p\n", p);        ----> 0x7ffc85524ec8

    SolidSquare* sq = static_cast<SolidSquare*>(p);

    printf("%p\n", sq);
}
```

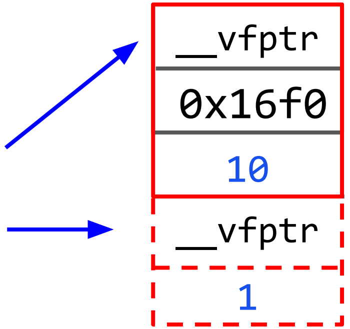


Multiple inheritance: so far

```
int main() {
    SolidSquare bs(10, 1);
    Square* s = &bs;
    printf("%p\n", s);          ----> 0x7ffc85524eb0
    Paintable* p = &bs;
    printf("%p\n", p);          ----> 0x7ffc85524ec8

    SolidSquare* sq = static_cast<SolidSquare*>(p);

    printf("%p\n", sq);         ----> 0x7ffc85524eb0
}
```



Multiple inheritance: so far

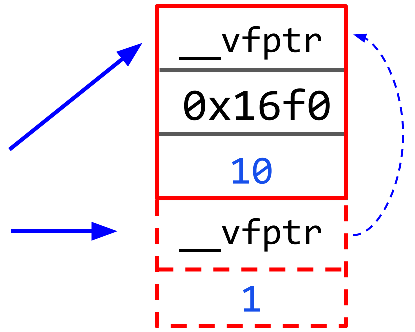


```
int main() {
    SolidSquare bs(10, 1);
    Square* s = &bs;
    printf("%p\n", s);       ----> 0x7ffc85524eb0
    Paintable* p = &bs;
    printf("%p\n", p);       ----> 0x7ffc85524ec8

    SolidSquare* sq = static_cast<SolidSquare*>(p);

    printf("%p\n", sq);      ----> 0x7ffc85524eb0
}
```

```
   ...
   cmp QWORD PTR [rbp-16], 0
   je .L9
   mov rax, QWORD PTR [rbp-16]
   sub rax, 24
   jmp .L10
.L9:
   mov eax, 0
.L10:
   mov QWORD PTR [rbp-24], rax
...
```

https://godbolt.org/z/Ec99d71fb

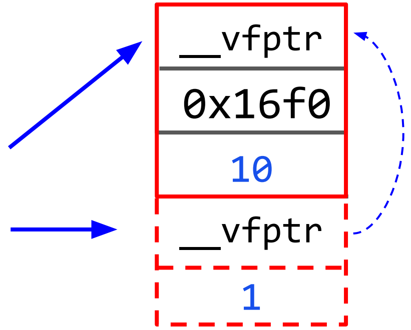
Multiple inheritance: so far



```
int main() {
    SolidSquare bs(10, 1);
    Square* s = &bs;
    printf("%p\n", s);        ----> 0x7ffc85524eb0
    Paintable* p = &bs;
    printf("%p\n", p);        ----> 0x7ffc85524ec8

    SolidSquare* sq = static_cast<SolidSquare*>(p);

    printf("%p\n", sq);       ----> 0x7ffc85524eb0
}
```

```
    ...
  cmp QWORD PTR [rbp-16], 0
  je .L9
  mov rax, QWORD PTR [rbp-16]
  sub rax, 24
  jmp .L10
.L9:
  mov eax, 0
.L10:
  mov QWORD PTR [rbp-24], rax
...
```

Multiple inheritance: so far



```
int main() {
    SolidSquare bs(10, 1);
    Square* s = &bs;
    printf("%p\n", s);        ----> 0x7ffc85524eb0
    Paintable* p = &bs;
    printf("%p\n", p);        ----> 0x7ffc85524ec8

    SolidSquare* sq = static_cast<SolidSquare*>(p);

    printf("%p\n", sq);       ----> 0x7ffc85524eb0
}
```

```
   ...
   cmp QWORD PTR [rbp-16], 0
   je .L9
   mov rax, QWORD PTR [rbp-16]
   sub rax, 24
   jmp .L10
 .L9:
   mov eax, 0
 .L10:
   mov QWORD PTR [rbp-24], rax
 ...
```

Static casts work perfectly fine in both directions (until you don’t try to cast an object to the type it actually doesn’t belong to).

```
class Square: public Figure {
protected:
    double length;
public:
    Square(double l):
            Figure("Square"),
            length(l) {}
    double area() {
        return length*length;
    }
};
```

```
class Paintable {
protected:
    int color;
public:
    Paintable(int color):
                    color(color) {}
    virtual void paint() = 0;
};
```

```
class Square: public Figure {
protected:
    double length;
public:
    Square(double l):
             Figure("Square"),
             length(l) {}
    double area() {
         return length*length;
    }

    void foo() { };
};
```

```
class Paintable {
protected:
    int color;
public:
    Paintable(int color):
                     color(color) {}
    virtual void paint() = 0;



    void foo() { };
};
```

```cpp
class Square: public Figure {
protected:
    double length;
public:
    Square(double l):
             Figure("Square"),
             length(l) {}
    double area() {
         return length*length;
    }

    void foo() { };
};
```

```cpp
class Paintable {
protected:
    int color;
public:
    Paintable(int color):
                     color(color) {}
    virtual void paint() = 0;



    void foo() { };
};
```

```cpp
int main() {
   SolidSquare bs(10, 1);
   bs.foo();
}
```

```cpp
class Square: public Figure {
protected:
    double length;
public:
    Square(double l):
             Figure("Square"),
             length(l) {}
    double area() {
         return length*length;
    }

    void foo() { };
};
```

```cpp
class Paintable {
protected:
    int color;
public:
    Paintable(int color):
                     color(color) {}
    virtual void paint() = 0;



    void foo() { };
};
```

```cpp
int main() {
   SolidSquare bs(10, 1);
 ✗ bs.foo();
}
```

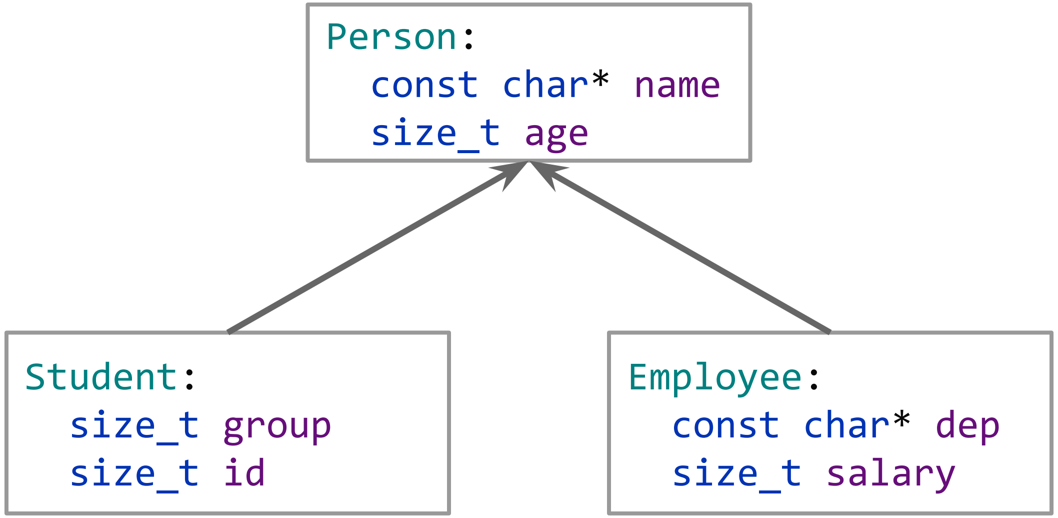
which one to call?

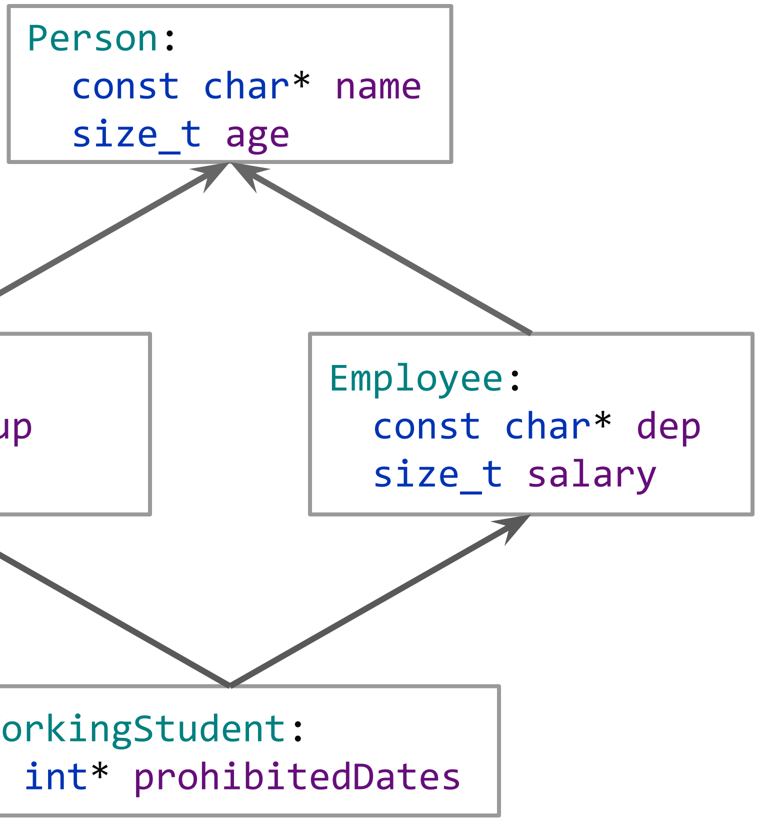
```
class Square: public Figure {
protected:
    double length;
public:
    Square(double l):
             Figure("Square"),
             length(l) {}
    double area() {
         return length*length;
    }

    void foo() { };
};
```

```
class Paintable {
protected:
    int color;
public:
    Paintable(int color):
                     color(color) {}
    virtual void paint() = 0;



    void foo() { };
};
```

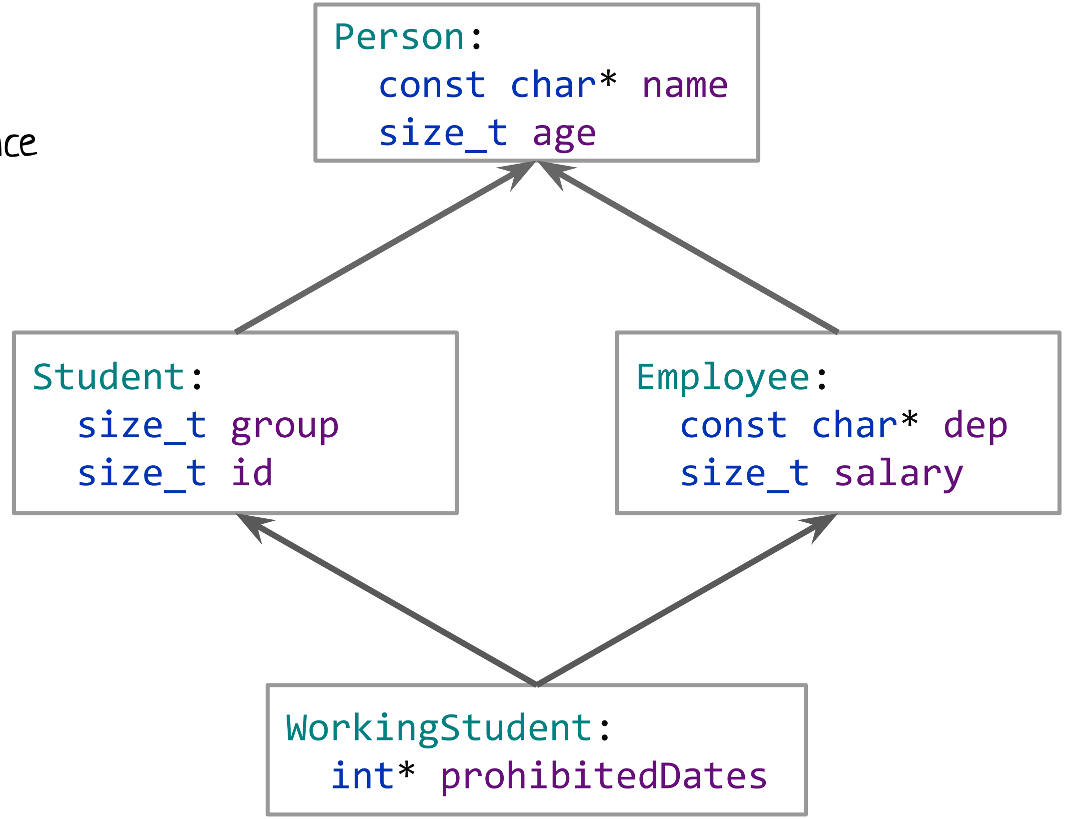
```
int main() {
   SolidSquare bs(10, 1);
✓  bs.Square::foo();
✓  bs.Paintable::foo();
}
```

ok, now it is clear

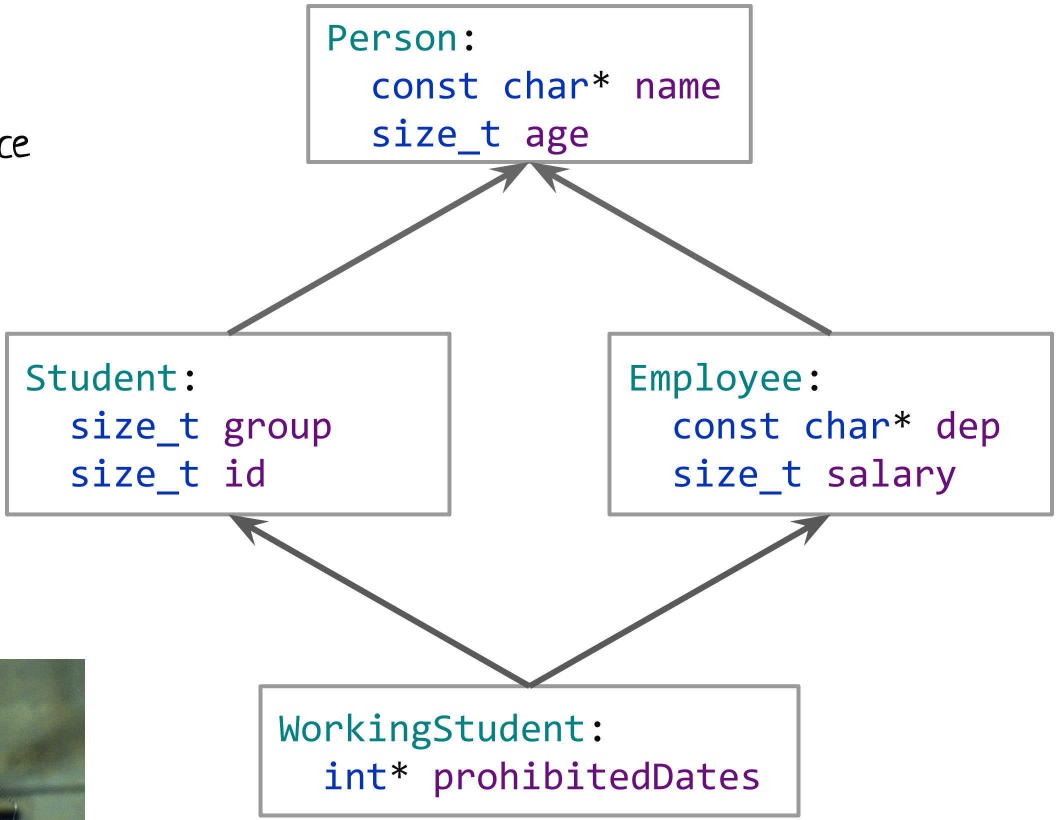

Person:
const char* name
size_t age
Student:
size_t group
size_t id
Employee:
const char* dep
size_t salary


Person:
const char* name
size_t age
Student:
size_t group
size_t id
Employee:
const char* dep
size_t salary
WorkingStudent:
int* prohibitedDates

- Diamond inheritance



♦ Diamond inheritance



```
Person:
  const char* name
  size_t age
```

```
Student:
  size_t group
  size_t id
```

```
Employee:
  const char* dep
  size_t salary
```

```
WorkingStudent:
  int* prohibitedDates
```

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

```cpp
class Student: public Person {
protected:
    size_t group;
    size_t id;
public:
    void print() {
        std::cout << name
                  << age
                  << group << id;
    }
};
```

```cpp
class Employee: public Person {
protected:
    const char* department;
    size_t salary;
public:
    void print() {
        std::cout << name << age
                  << department
                  << salary;
    }
};
```

```cpp
class WorkingStudent: public Student, public Employee {
protected:
    int* prohibitedDates;
};
```

```
class WorkingStudent: public Student, public Employee {
protected:
    int* prohibitedDates;
};
```

```
int main() {
    WorkingStudent ws;
    ws.print(); // compilation error. It is expected.
}
```

```
class WorkingStudent: public Student, public Employee {
protected:
    int* prohibitedDates;
};
```

```
int main() {
    WorkingStudent ws;
  ✓ ws.Student::print();
  ✓ ws.Employee::print();
}
```

```
class WorkingStudent: public Student, public Employee {
protected:
    int* prohibitedDates;
public:
    void print() {
        std::cout << name;
    }
};



int main() {
    WorkingStudent ws;
  ✓ ws.print();
}
```

```
class WorkingStudent: public Student, public Employee {
protected:
    int* prohibitedDates;
public:
    void print() {
        std::cout << name; // compilation error
    }
};



int main() {
    WorkingStudent ws;
  ✓ ws.print();
}
```

```
class WorkingStudent: public Student, public Employee {
protected:
    int* prohibitedDates;
public:
    void print() {
        std::cout << name; // compilation error ...why?
    }
};



int main() {
    WorkingStudent ws;
    ws.print();
}
```

```
class WorkingStudent: public Student, public Employee {
protected:
    int* prohibitedDates;
public:
    void print() {
        std::cout << name; // compilation error ...why?
    }
};



int main() {
    WorkingStudent ws;
    ws.print();

    Person* p = &ws; // compilation error
}
```

```
class WorkingStudent: public Student, public Employee {
protected:
    int* prohibitedDates;
public:
    void print() {
        std::cout << name; // compilation error ...why?
    }
};
```

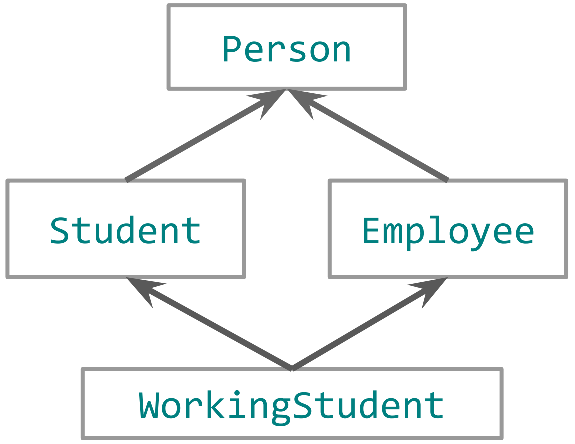
```
void test(Person* p) {
    p->print();
}
```

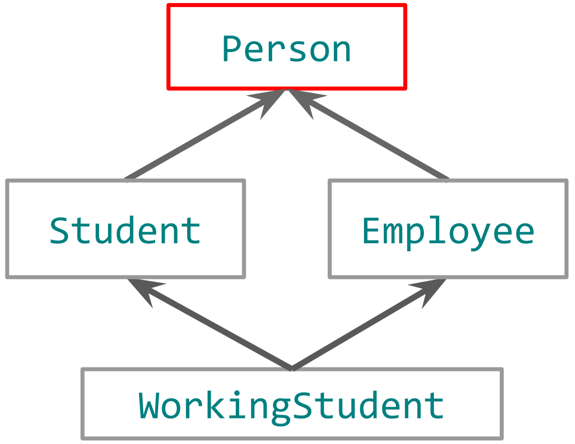
```
int main() {
    WorkingStudent ws;
    ws.print();

    Person* p = &ws; // compilation error
    test(&ws);       // compilation error
}
```

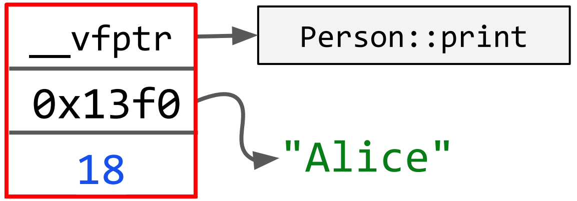
```
class WorkingStudent: public Student, public Employee {
protected:
    int* prohibitedDates;
public:
    void print() {
        std::cout << name; // compilation error ...why?
    }
};
```

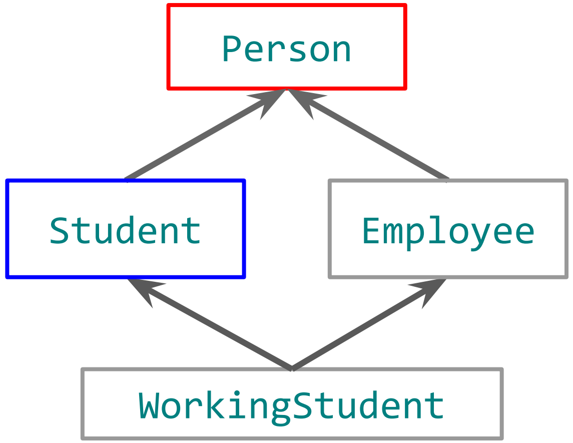
```
void test(Person* p) {
    p->print();
}
```

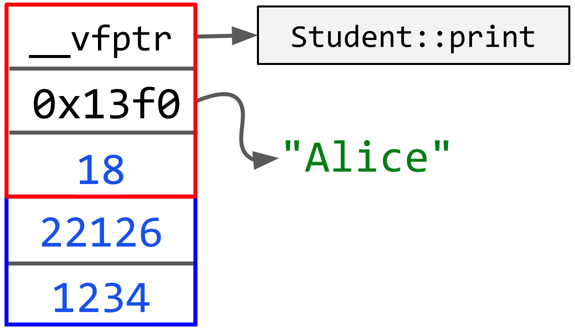
```
int main() {
    WorkingStudent ws;
    ws.print();

    Person* p = &ws; // compilation error
    test(&ws);       // compilation error
}
```

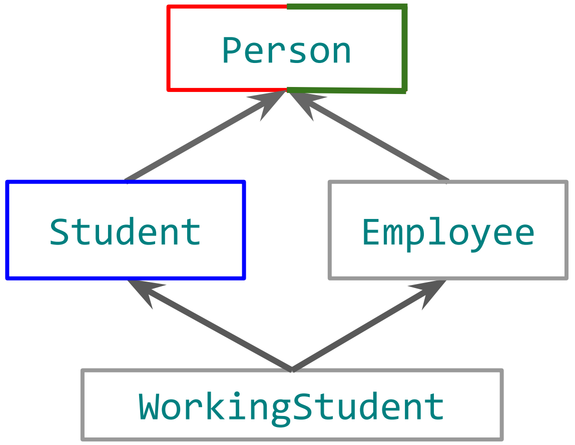

Person
Student
Employee
WorkingStudent

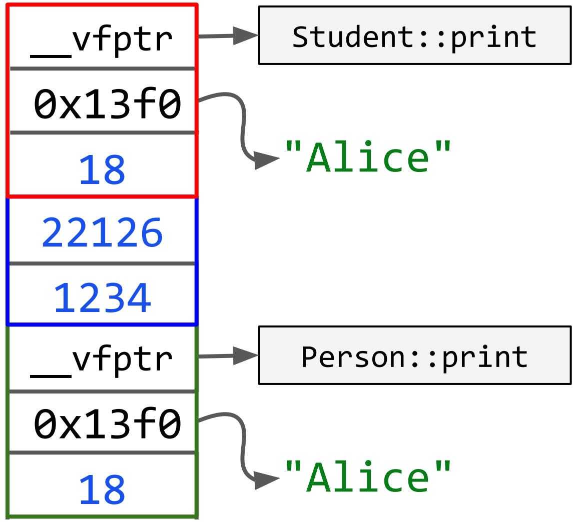

Person
Student
Employee
WorkingStudent

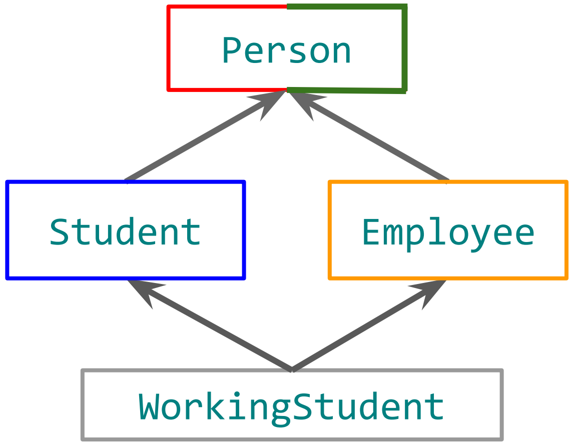

__vfptr
0x13f0
18
Person::print
"Alice"

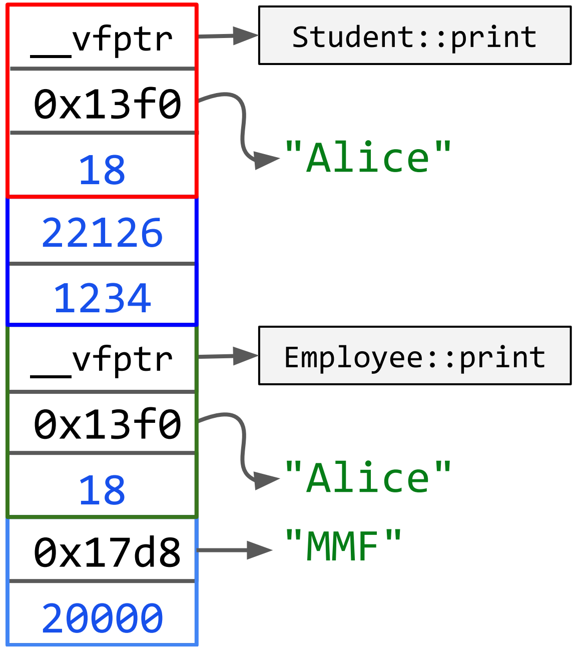

Person
Student
Employee
WorkingStudent

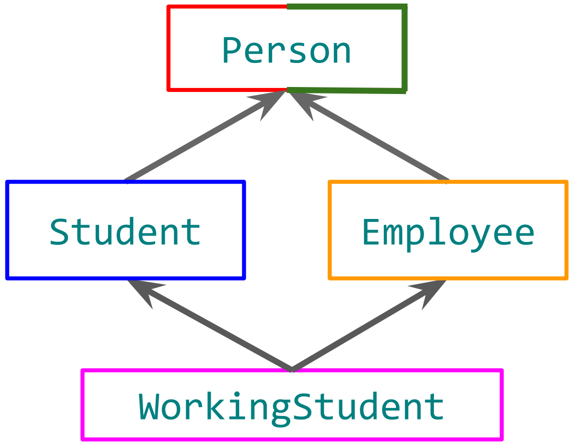

__vfptr
Student::print
0x13f0
"Alice"
18
22126
1234

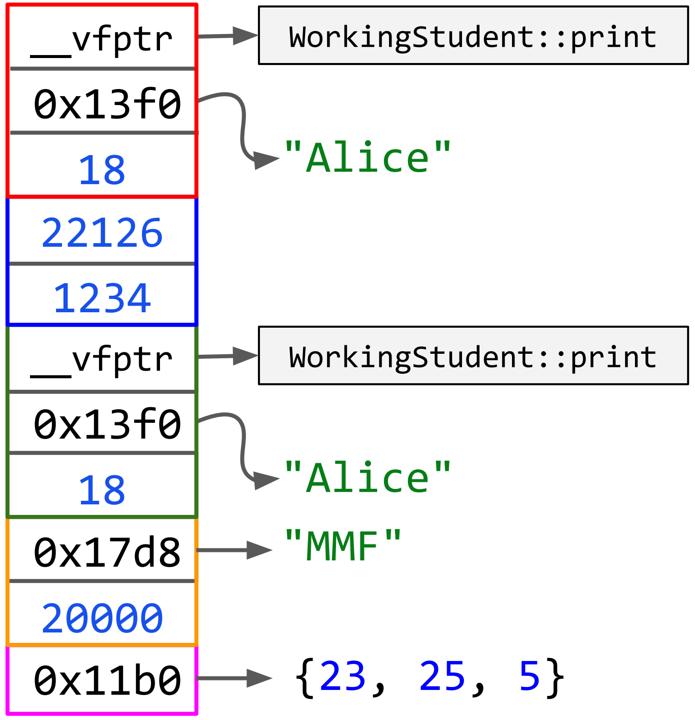

Person
Student
Employee
WorkingStudent

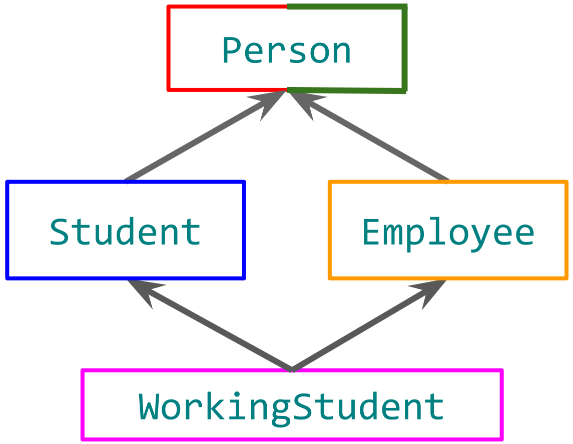

__vfptr
Student::print
0x13f0
"Alice"
18
22126
1234
__vfptr
Person::print
0x13f0
"Alice"
18

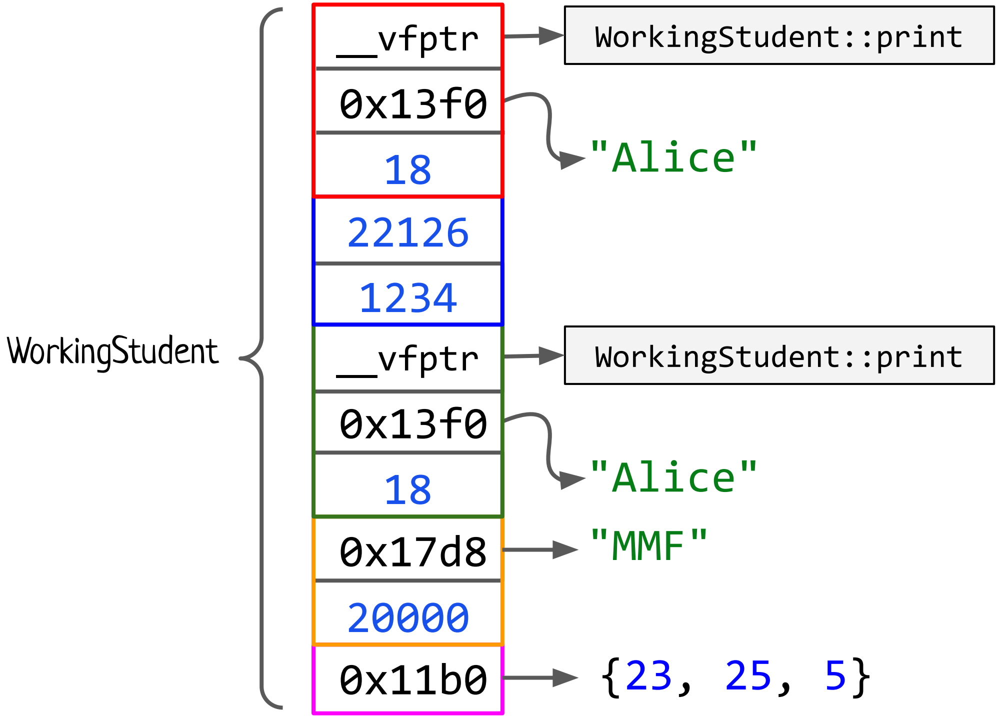

Person
Student
Employee
WorkingStudent

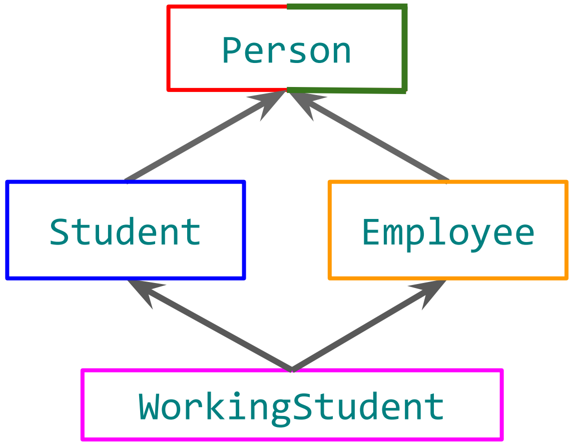

__vfptr
Student::print
0x13f0
"Alice"
18
22126
1234
__vfptr
Employee::print
0x13f0
"Alice"
18
0x17d8
"MMF"
20000

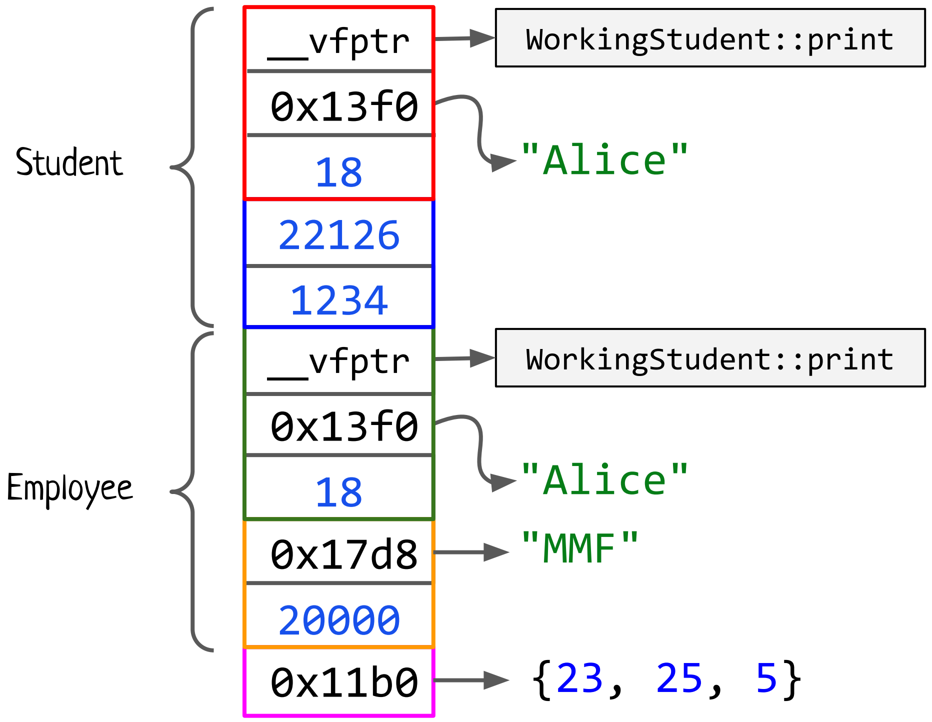

Person
Student
Employee
WorkingStudent

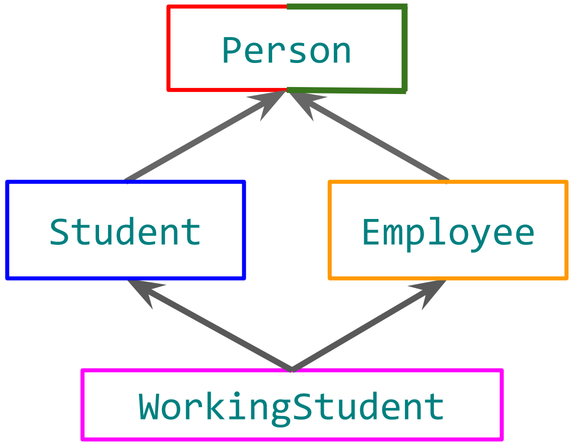

__vfptr
WorkingStudent::print
0x13f0
"Alice"
18
22126
1234
__vfptr
WorkingStudent::print
0x13f0
"Alice"
18
0x17d8
"MMF"
20000
0x11b0
{23, 25, 5}

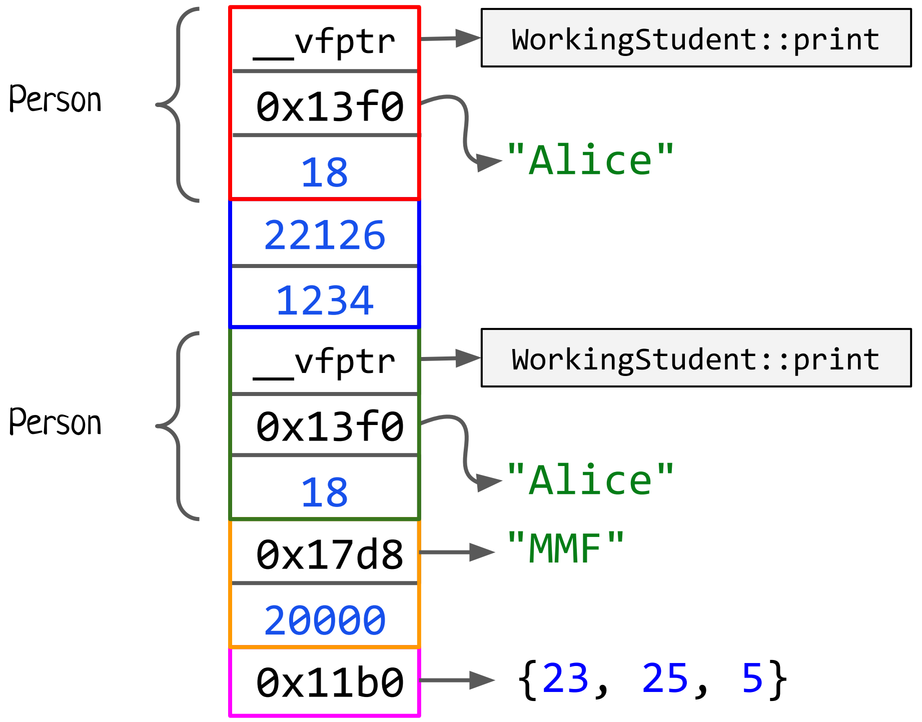

Person
Student
Employee
WorkingStudent

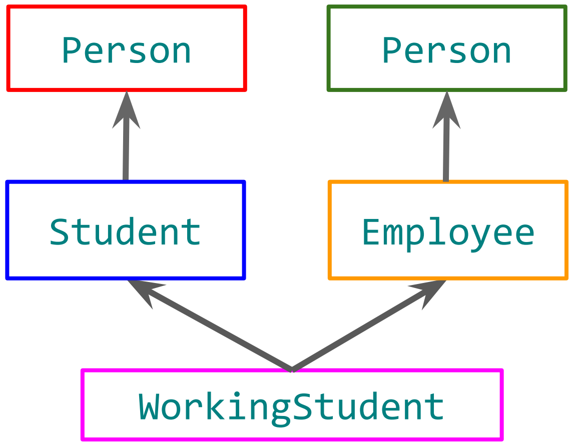

WorkingStudent
__vfptr
0x13f0
18
22126
1234
__vfptr
0x13f0
18
0x17d8
20000
0x11b0
WorkingStudent::print
"Alice"
WorkingStudent::print
"Alice"
"MMF"
{23, 25, 5}

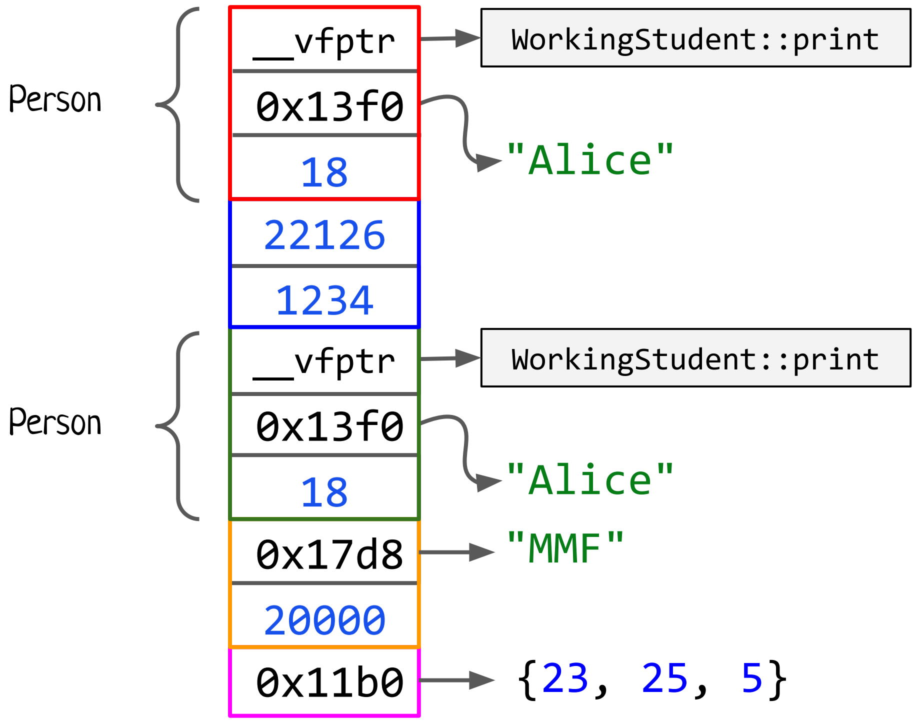

Person
Student
Employee
WorkingStudent


Student
Employee
__vfptr
WorkingStudent::print
0x13f0
"Alice"
18
22126
1234
__vfptr
WorkingStudent::print
0x13f0
"Alice"
18
0x17d8
"MMF"
20000
0x11b0
{23, 25, 5}


Person
Student
Employee
WorkingStudent


Person
__vfptr
WorkingStudent::print
0x13f0
"Alice"
18
22126
1234
Person
__vfptr
WorkingStudent::print
0x13f0
"Alice"
18
0x17d8
"MMF"
20000
0x11b0
{23, 25, 5}


Person
Person
Student
Employee
WorkingStudent


Person
__vfptr
WorkingStudent::print
0x13f0
"Alice"
18
22126
1234
Person
__vfptr
WorkingStudent::print
0x13f0
"Alice"
18
0x17d8
"MMF"
20000
0x11b0
{23, 25, 5}

```
class WorkingStudent: public Student, public Employee {
protected:
    int* prohibitedDates;
public:
    void print() {
        std::cout << name; // compilation error ...why?
    }
};
```

```
void test(Person* p) {
    p->print();
}
```

```
int main() {
    WorkingStudent ws;
    ws.print();

    Person* p = &ws; // compilation error
    test(&ws);       // compilation error
}
```

```cpp
class WorkingStudent: public Student, public Employee {
protected:
    int* prohibitedDates;
public:
    void print() {
        std::cout << name; // compilation error ...because which name?
    }
};
```

```cpp
void test(Person* p) {
    p->print();
}
```

```cpp
int main() {
    WorkingStudent ws;
    ws.print();

    Person* p = &ws; // compilation error
    test(&ws);       // compilation error
}
```

} because to which base?

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

```
class Student: public Person {
protected:
    size_t group;
    size_t id;
public:
    void print() {
        std::cout << name
                  << age
                  << group << id;
    }
};
```

```
class Employee: public Person {
protected:
    const char* department;
    size_t salary;
public:
    void print() {
        std::cout << name << age
                  << department
                  << salary;
    }
};
```

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

```cpp
class Student: virtual public Person {
protected:
    size_t group;
    size_t id;
public:
    void print() {
        std::cout << name
                  << age
                  << group << id;
    }
};
```

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

Virtual inheritance

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

```
class Student: virtual public Person {
protected:
    size_t group;
    size_t id;
public:
    void print() {
        std::cout << name
                  << age
                  << group << id;
    }
};
```

```
class Employee: virtual public Person {
protected:
    const char* department;
    size_t salary;
public:
    void print() {
        std::cout << name << age
                  << department
                  << salary;
    }
};
```

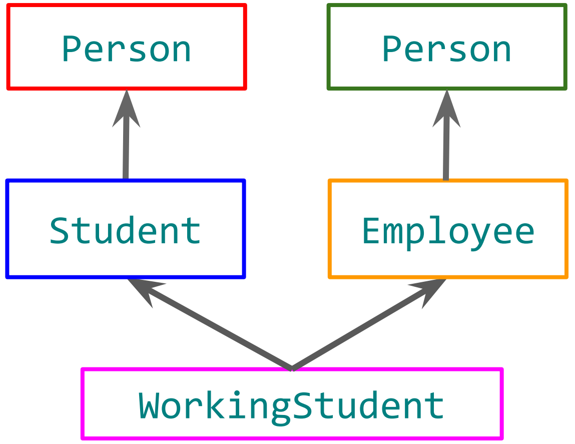
Virtual inheritance

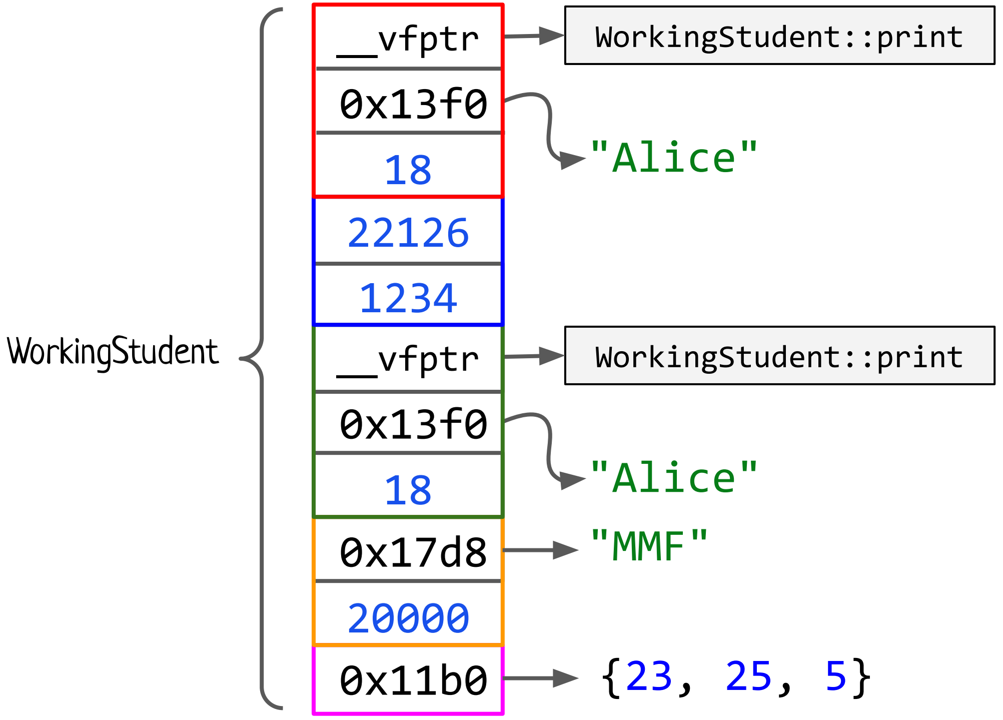
```
class Person {
protected:
    const char* name;
    size_t age;
public:
    virtual void print() {
        std::cout << name << age;
    }
};
```

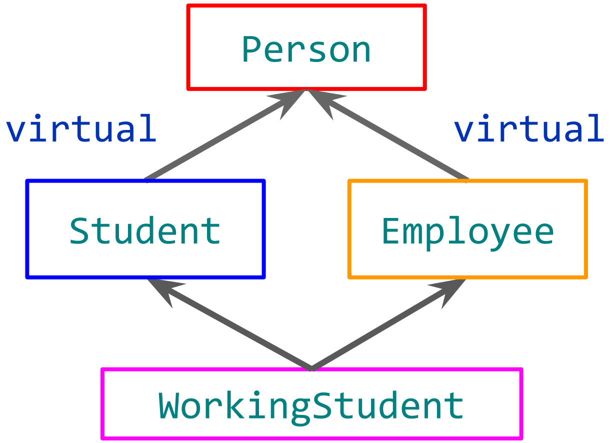
It means that even if there could be several Person parts inside Derived class (because of inheritance), only one will left.

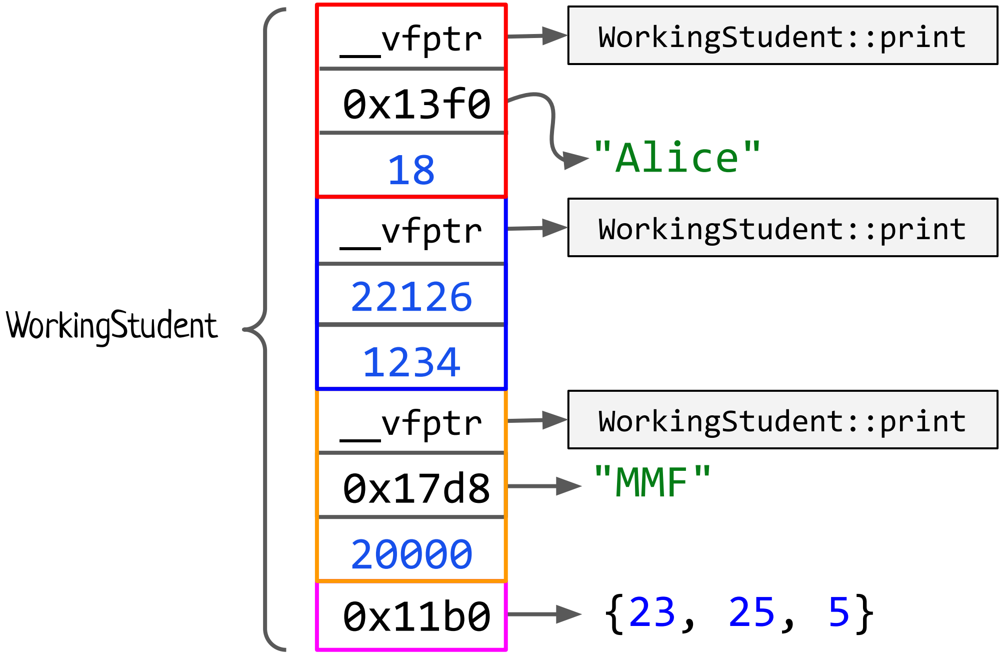
```
class Student: virtual public Person {
protected:
    size_t group;
    size_t id;
public:
    void print() {
        std::cout << name
                  << age
                  << group << id;
    }
};
```

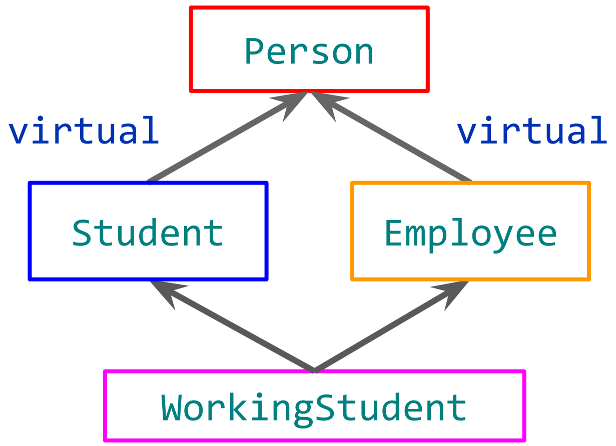
```
class Employee: virtual public Person {
protected:
    const char* department;
    size_t salary;
public:
    void print() {
        std::cout << name << age
                  << department
                  << salary;
    }
};
```

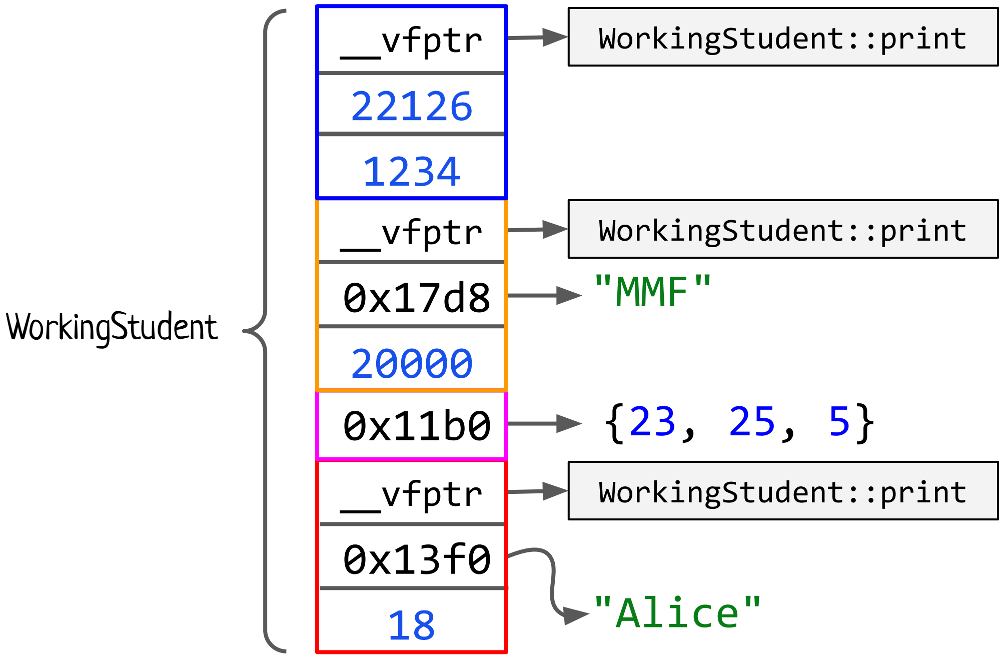

Person
Person
Student
Employee
WorkingStudent


WorkingStudent
__vfptr
WorkingStudent::print
0x13f0
"Alice"
18
22126
1234
__vfptr
WorkingStudent::print
0x13f0
"Alice"
18
0x17d8
"MMF"
20000
0x11b0
{23, 25, 5}


Person
virtual
virtual
Student
Employee
WorkingStudent


WorkingStudent
__vfptr
WorkingStudent::print
0x13f0
"Alice"
18
__vfptr
WorkingStudent::print
22126
1234
__vfptr
WorkingStudent::print
0x17d8
"MMF"
20000
0x11b0
{23, 25, 5}





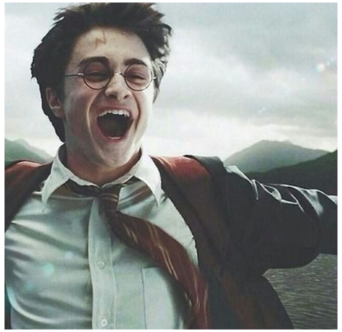
Basically, it doesn't matter, where to place this "common" part, let's put all at the end of structure (later will be clear one).

```
class WorkingStudent: public Student, public Employee {
protected:
    int* prohibitedDates;
public:
    void print() {
        std::cout << name; // compilation error ...why?
    }
};
```

```
void test(Person* p) {
    p->print();
}
```

```
int main() {
    WorkingStudent ws;
    ws.print();

    Person* p = &ws; // compilation error
    test(&ws);       // compilation error
}
```

```
class WorkingStudent: public Student, public Employee {
protected:
    int* prohibitedDates;
public:
    void print() {
        std::cout << name; // ok
    }
};
```



```
int main() {
    WorkingStudent ws;
    ws.print();

    Person* p = &ws; // ok
    test(&ws);       // ok
}
```

```
void test(Person* p) {
    p->print();
}
```

```
void test_p(Person& p) { ... }

void test_s(Student& s) { ... }

void test_e(Employee& e) { ... }

void test_ws(WorkingStudent& ws) { ... }

int main() {
    WorkingStudent ws;
    test_p(ws);
    test_s(ws);
    test_e(ws);
    test_ws(ws);
}
```

```cpp
void test_p(Person& p) { ... }

void test_s(Student& s) { ... }

void test_e(Employee& e) { ... }

void test_ws(WorkingStudent& ws) { ... }

int main() {
    WorkingStudent ws;
    test_p(ws);
    test_s(ws);
    test_e(ws);
    test_ws(ws);
}
```

→

```asm
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 80
        lea     rax, [rbp-80]
        mov     rdi, rax
        call    WorkingStudent::WorkingStudent()
        lea     rax, [rbp-80]
        add     rax, 48
        mov     rdi, rax
        call    test_p(Person&)
        lea     rax, [rbp-80]
        mov     rdi, rax
        call    test_s(Student&)
        lea     rax, [rbp-80]
        add     rax, 16
        mov     rdi, rax
        call    test_e(Employee&)
        lea     rax, [rbp-80]
        mov     rdi, rax
        call    test_ws(WorkingStudent&)
        mov     eax, 0
        leave
        ret
```

https://godbolt.org/z/hxPdnKx5K

```cpp
void test_p(Person& p) { ... }

void test_s(Student& s) { ... }

void test_e(Employee& e) { ... }

void test_ws(WorkingStudent& ws) { ... }

int main() {
    WorkingStudent ws;
    test_p(ws);
    test_s(ws);
    test_e(ws);
    test_ws(ws);
}
```

```asm
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 80
        lea     rax, [rbp-80]
        mov     rdi, rax
        call    WorkingStudent::WorkingStudent()
        lea     rax, [rbp-80]
        add     rax, 48
        mov     rdi, rax
        call    test_p(Person&)
        lea     rax, [rbp-80]
        mov     rdi, rax
        call    test_s(Student&)
        lea     rax, [rbp-80]
        add     rax, 16
        mov     rdi, rax
        call    test_e(Employee&)
        lea     rax, [rbp-80]
        mov     rdi, rax
        call    test_ws(WorkingStudent&)
        mov     eax, 0
        leave
        ret
```

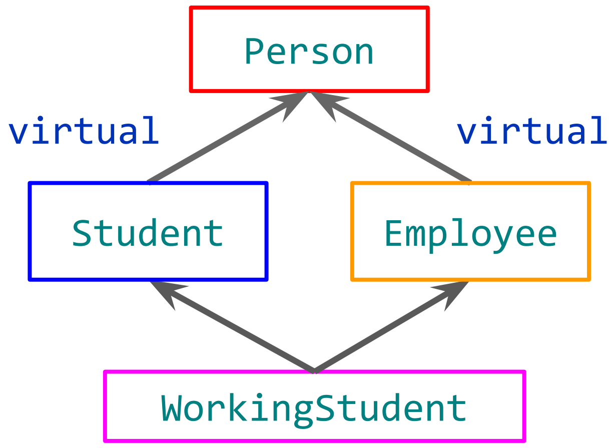
ctr call

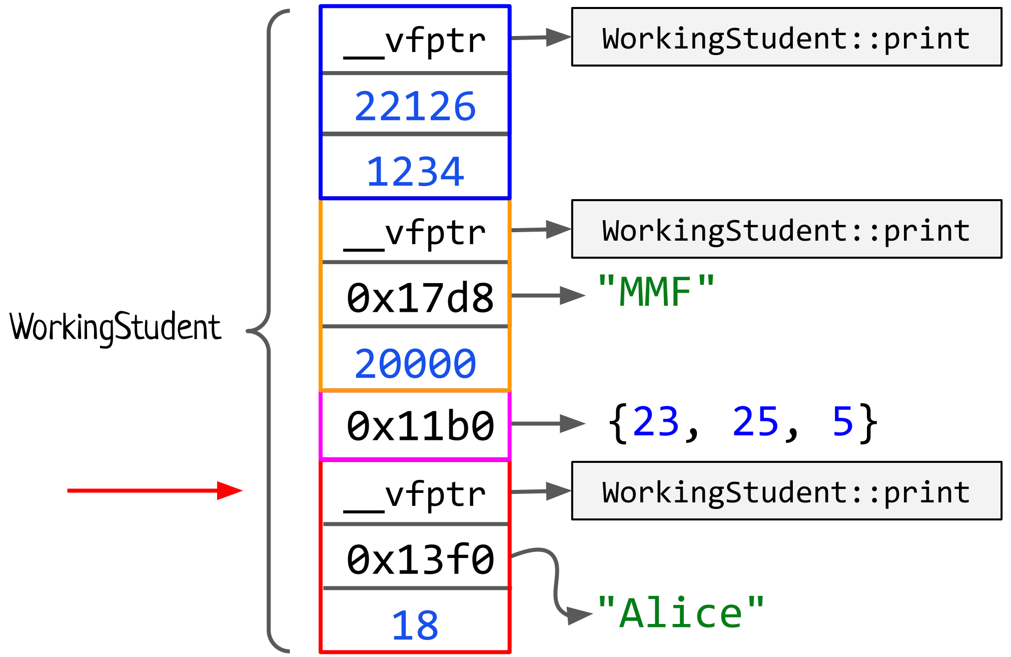
https://godbolt.org/z/hxPdnKx5K

```cpp
void test_p(Person& p) { ... }

void test_s(Student& s) { ... }

void test_e(Employee& e) { ... }

void test_ws(WorkingStudent& ws) { ... }

int main() {
    WorkingStudent ws;
    test_p(ws);  <----
    test_s(ws);
    test_e(ws);
    test_ws(ws);
}
```

```asm
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 80
        lea     rax, [rbp-80]
        mov     rdi, rax
        call    WorkingStudent::WorkingStudent()
        lea     rax, [rbp-80]
        add     rax, 48  <-- shift to the end
        mov     rdi, rax
        call    test_p(Person&)
        lea     rax, [rbp-80]
        mov     rdi, rax
        call    test_s(Student&)
        lea     rax, [rbp-80]
        add     rax, 16
        mov     rdi, rax
        call    test_e(Employee&)
        lea     rax, [rbp-80]
        mov     rdi, rax
        call    test_ws(WorkingStudent&)
        mov     eax, 0
        leave
        ret
```

https://godbolt.org/z/hxPdnKx5K



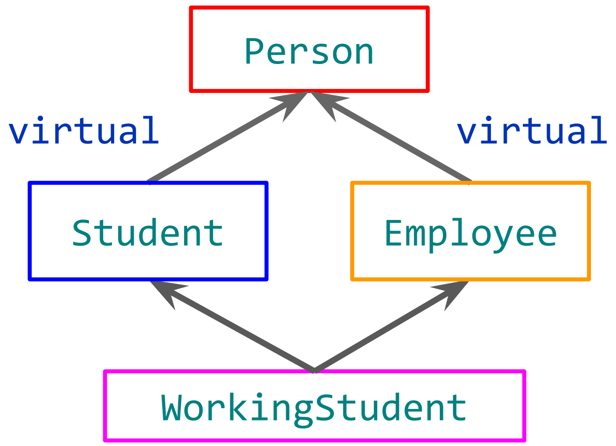


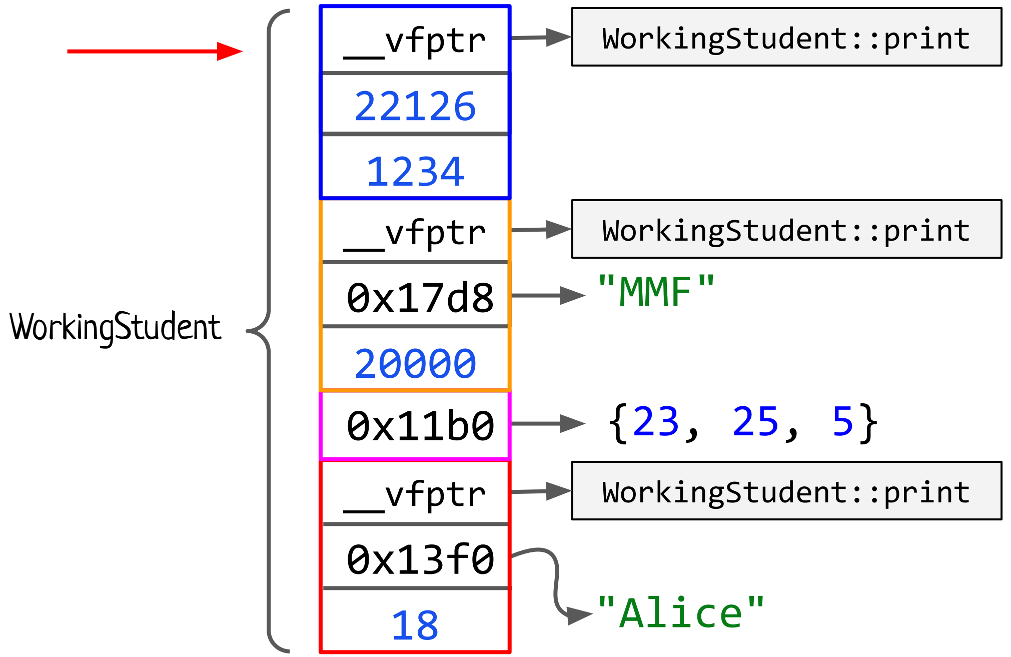
Basically, it doesn't matter, where to place this "common" part, let's put all at the end of structure (later will be clear one).

```cpp
void test_p(Person& p) { ... }

void test_s(Student& s) { ... }

void test_e(Employee& e) { ... }

void test_ws(WorkingStudent& ws) { ... }

int main() {
    WorkingStudent ws;
    test_p(ws);
    test_s(ws);
    test_e(ws);
    test_ws(ws);
}
```

```asm
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 80
        lea     rax, [rbp-80]
        mov     rdi, rax
        call    WorkingStudent::WorkingStudent()
        lea     rax, [rbp-80]
        add     rax, 48
        mov     rdi, rax
        call    test_p(Person&)
        lea     rax, [rbp-80]
        mov     rdi, rax
        call    test_s(Student&)
        lea     rax, [rbp-80]
        add     rax, 16
        mov     rdi, rax
        call    test_e(Employee&)
        lea     rax, [rbp-80]
        mov     rdi, rax
        call    test_ws(WorkingStudent&)
        mov     eax, 0
        leave
        ret
```

https://godbolt.org/z/hxPdnKx5K





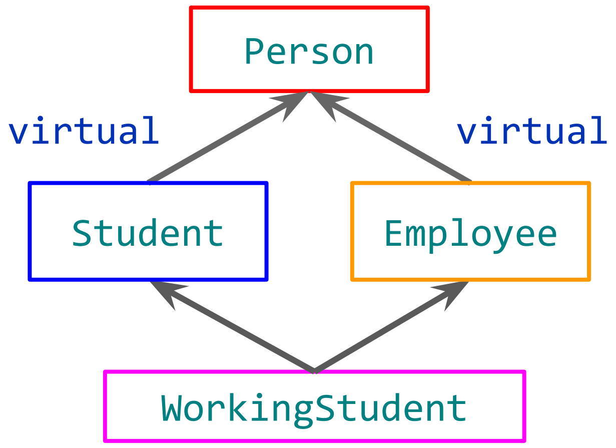
Basically, it doesn't matter, where to place this "common" part, let's put all at the end of structure (later will be clear one).

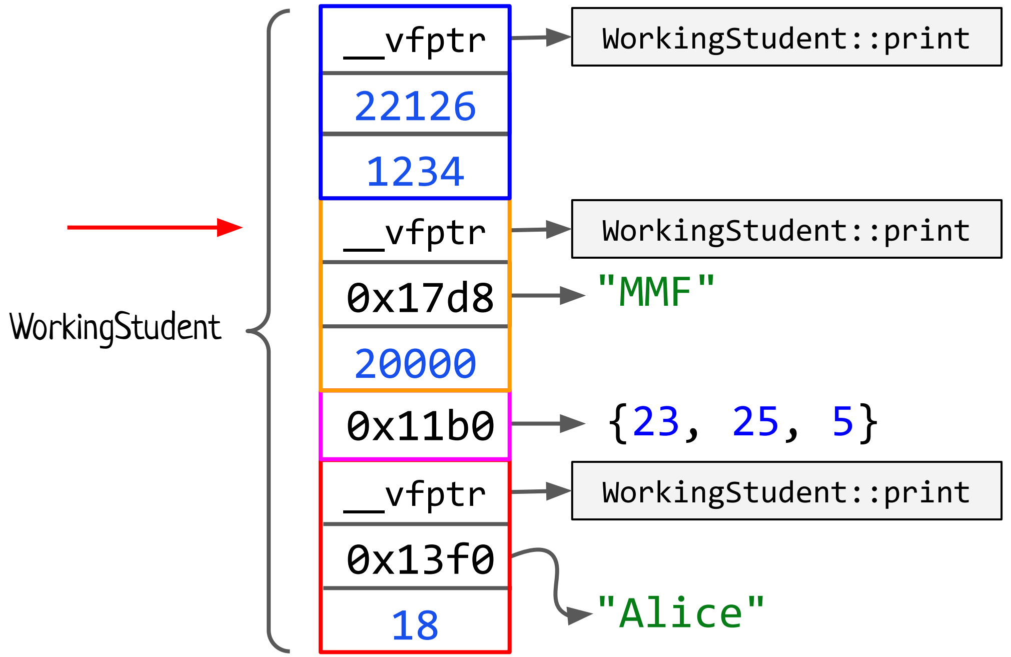
```cpp
void test_p(Person& p) { ... }

void test_s(Student& s) { ... }

void test_e(Employee& e) { ... }

void test_ws(WorkingStudent& ws) { ... }

int main() {
    WorkingStudent ws;
    test_p(ws);
    test_s(ws);
    test_e(ws);   ←
    test_ws(ws);
}
```

→

```asm
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 80
        lea     rax, [rbp-80]
        mov     rdi, rax
        call    WorkingStudent::WorkingStudent()
        lea     rax, [rbp-80]
        add     rax, 48
        mov     rdi, rax
        call    test_p(Person&)
        lea     rax, [rbp-80]
        mov     rdi, rax
        call    test_s(Student&)
        lea     rax, [rbp-80]
        add     rax, 16  ←
        mov     rdi, rax
        call    test_e(Employee&)
        lea     rax, [rbp-80]
        mov     rdi, rax
        call    test_ws(WorkingStudent&)
        mov     eax, 0
        leave
        ret
```

https://godbolt.org/z/hxPdnKx5K





Basically, it doesn't matter, where to place this "common" part, let's put all at the end of structure (later will be clear one).

```cpp
void test_p(Person& p) { ... }

void test_s(Student& s) { ... }

void test_e(Employee& e) { ... }

void test_ws(WorkingStudent& ws) { ... }

int main() {
    WorkingStudent ws;
    test_p(ws);
    test_s(ws);
    test_e(ws);
    test_ws(ws);
}
```

```asm
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 80
        lea     rax, [rbp-80]
        mov     rdi, rax
        call    WorkingStudent::WorkingStudent()
        lea     rax, [rbp-80]
        add     rax, 48
        mov     rdi, rax
        call    test_p(Person&)
        lea     rax, [rbp-80]
        mov     rdi, rax
        call    test_s(Student&)
        lea     rax, [rbp-80]
        add     rax, 16
        mov     rdi, rax
        call    test_e(Employee&)
        lea     rax, [rbp-80]
        mov     rdi, rax
        call    test_ws(WorkingStudent&)
        mov     eax, 0
        leave
        ret
```

https://godbolt.org/z/hxPdnKx5K

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

void test_s(Student& s) {
    printf("%s", s.name);
}


int main() {
    WorkingStudent ws;
    test_p(ws);
    test_s(ws);

}
```

Nothing strange here

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

void test_s(Student& s) {
    printf("%s", s.name);
}


int main() {
    WorkingStudent ws;
    test_p(ws);
    test_s(ws);

}
```

```
.LC0:
        .string "%s"
test_p(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
```

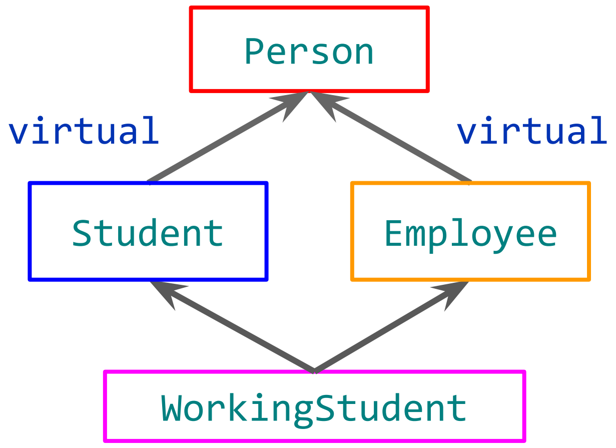
https://godbolt.org/z/vefbo56cP

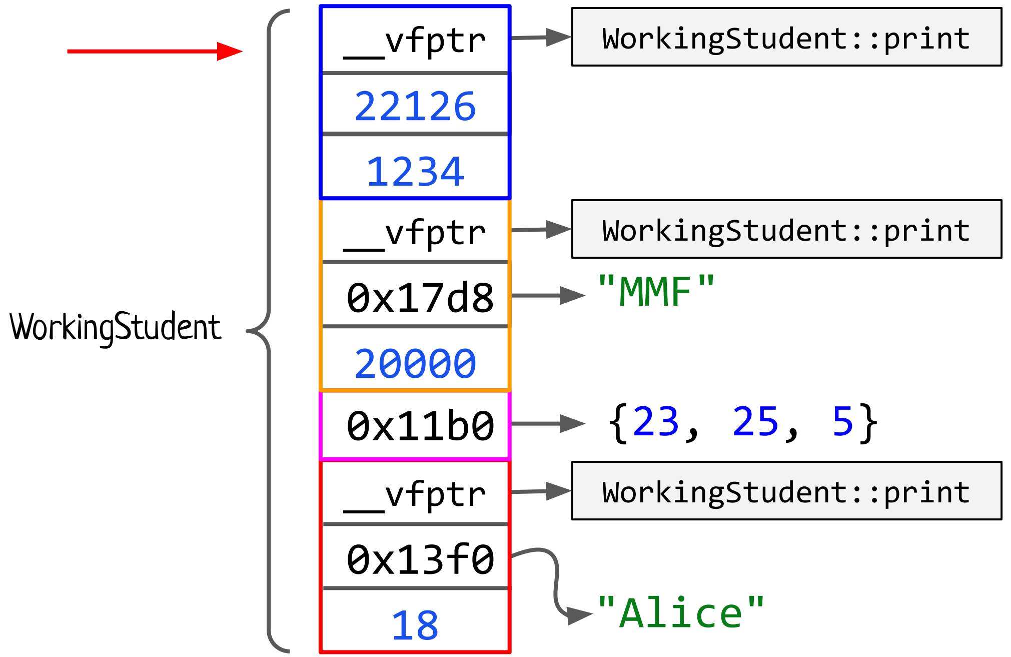
```
void test_p(Person& p) {
    printf("%s", p.name);
}

void test_s(Student& s) {
    printf("%s", s.name);
}


int main() {
    WorkingStudent ws;
    test_p(ws);
    test_s(ws);

}
```

$\rightarrow$

```
test_s(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]
        sub     rax, 24
        mov     rax, QWORD PTR [rax]
        mov     rdx, rax
        mov     rax, QWORD PTR [rbp-8]
        add     rax, rdx
        mov     rax, QWORD PTR [rax+8]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        nop
        leave
        ret
```

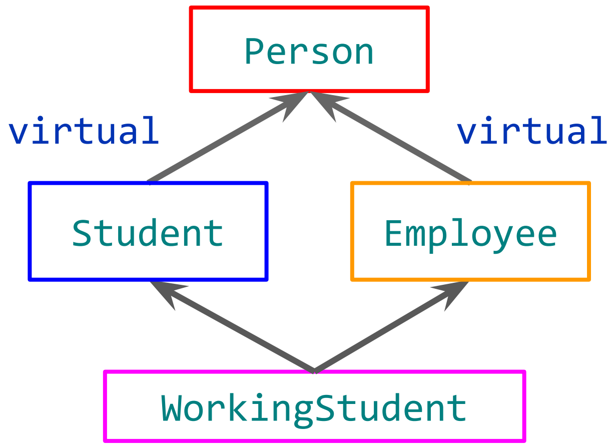
https://godbolt.org/z/vefbo56cP

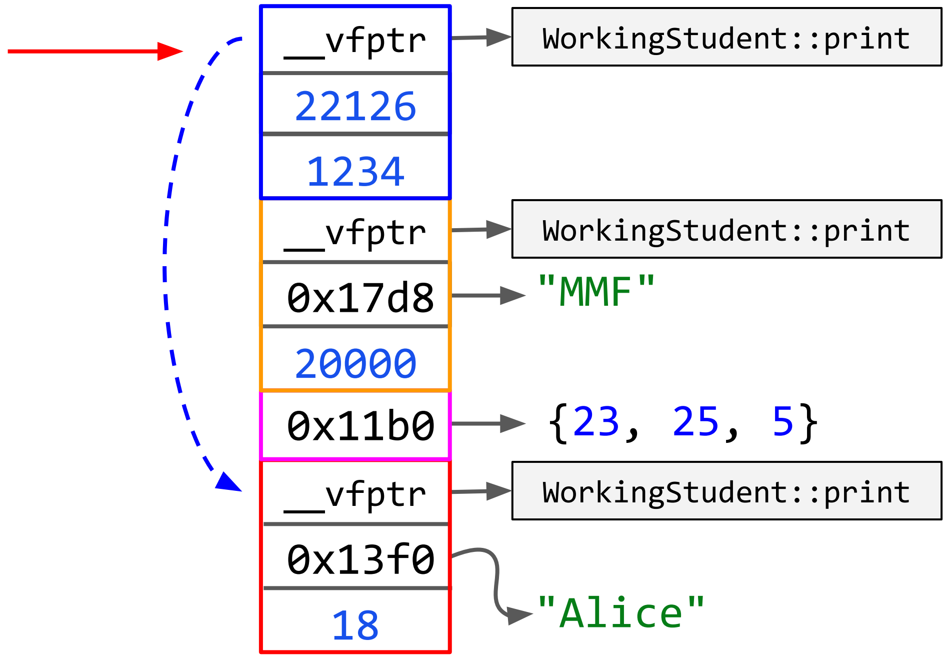




Basically, it doesn't matter, where to place this "common" part, let's put all at the end of structure (later will be clear one).

```
How can I access Person::name,
if this points to Student?
```

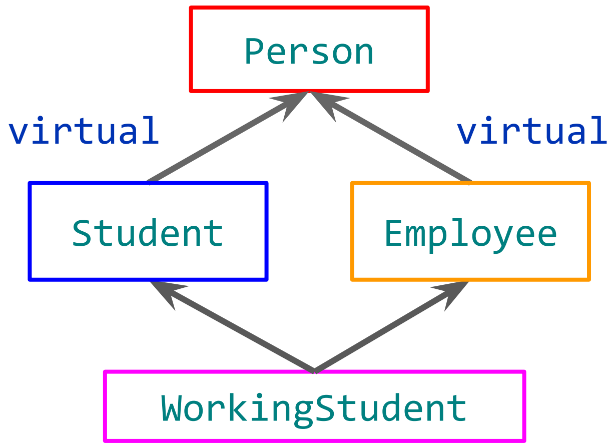


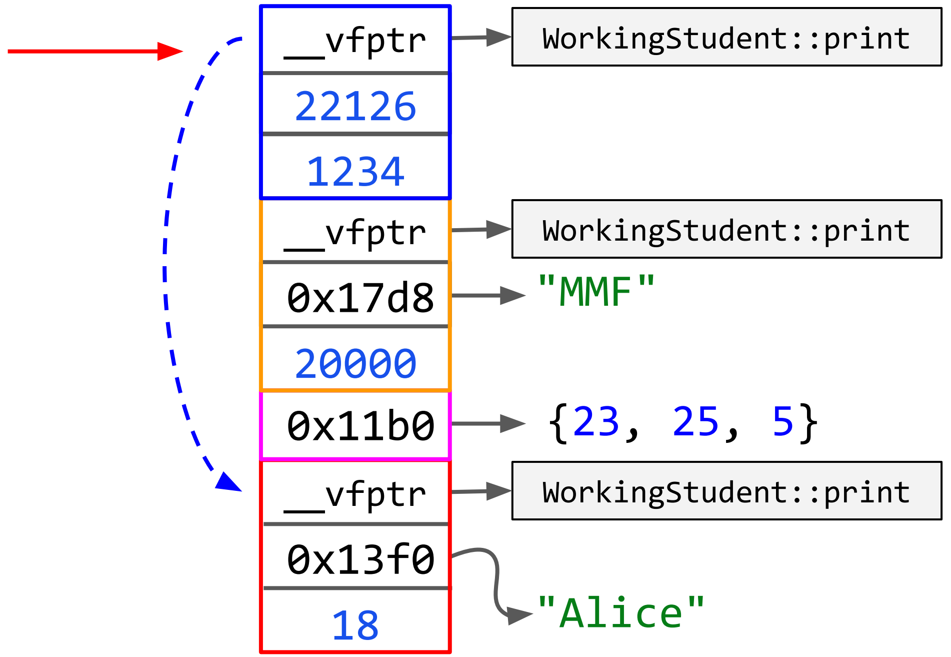


Basically, it doesn't matter, where to place this "common" part, let's put all at the end of structure (later will be clear one).

How can I access Person::name, if this points to Student?

The problem is that the offset to name field is… unknown in compile time!

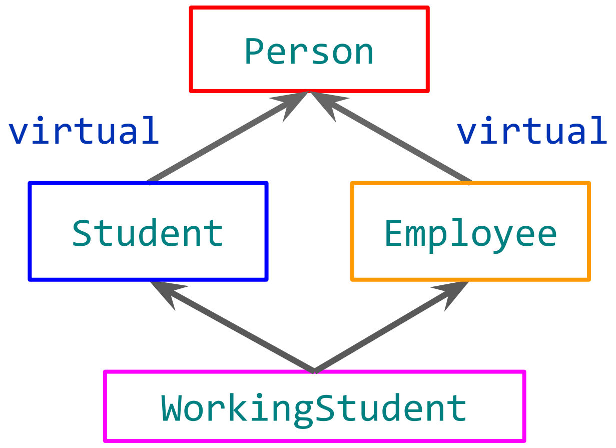


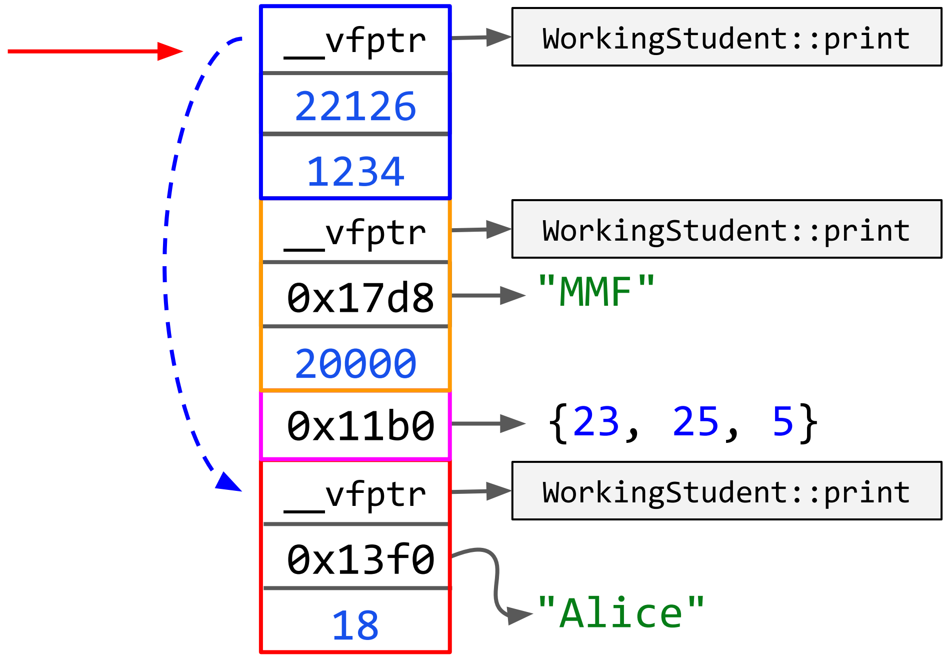


Basically, it doesn't matter, where to place this "common" part, let's put all at the end of structure (later will be clear one).

How can I access Person::name, if this points to Student?

The problem is that the offset to name field is… unknown in compile time! (can’t know if you have 2 bases or 1, or 3, except Student)

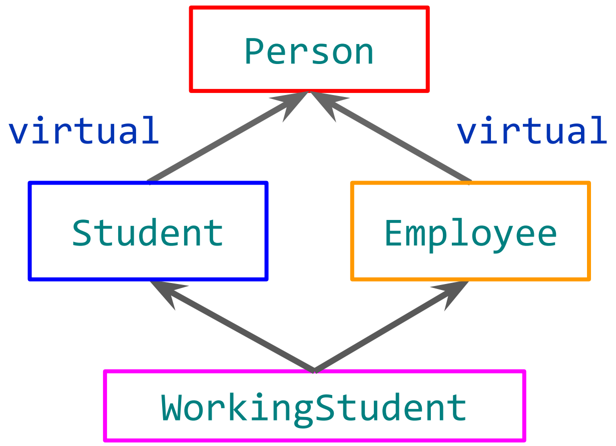


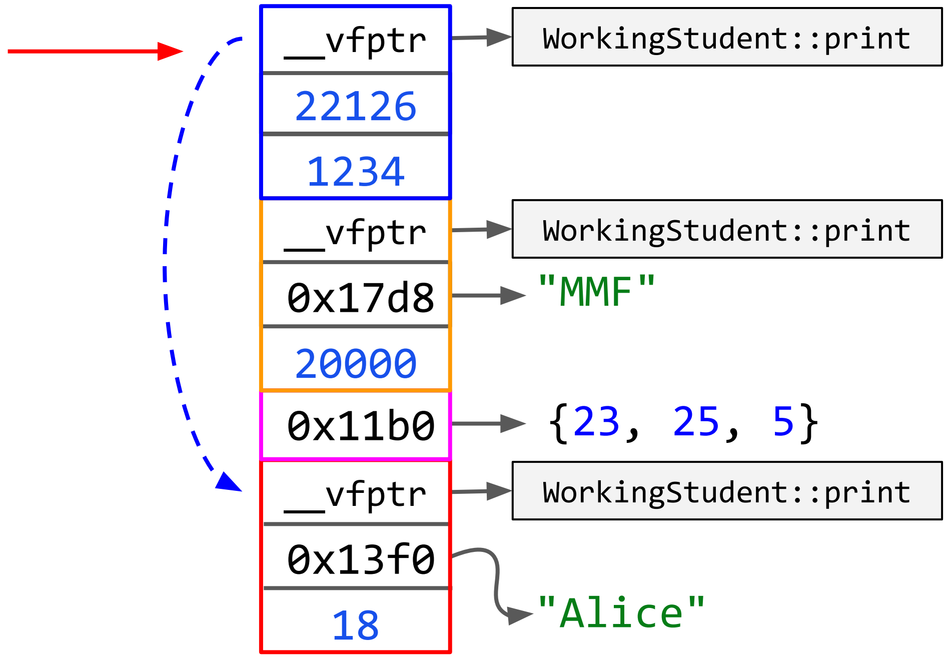


Basically, it doesn’t matter, where to place this "common" part, let’s put all at the end of structure (later will be clear one).

How can I access Person::name, if this points to Student?

The problem is that the offset to name field is... unknown in compile time! (can't know if you have 2 bases or 1, or 3, except Student)



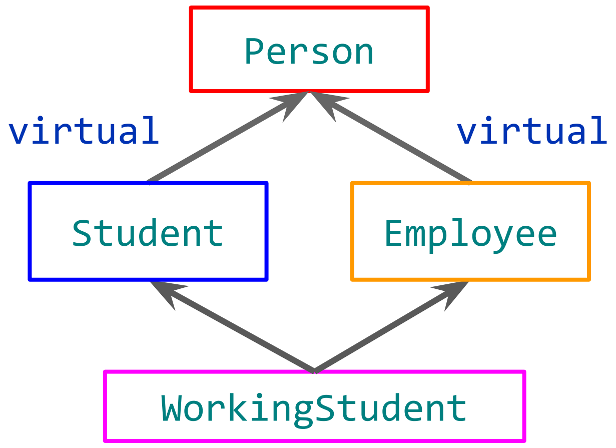


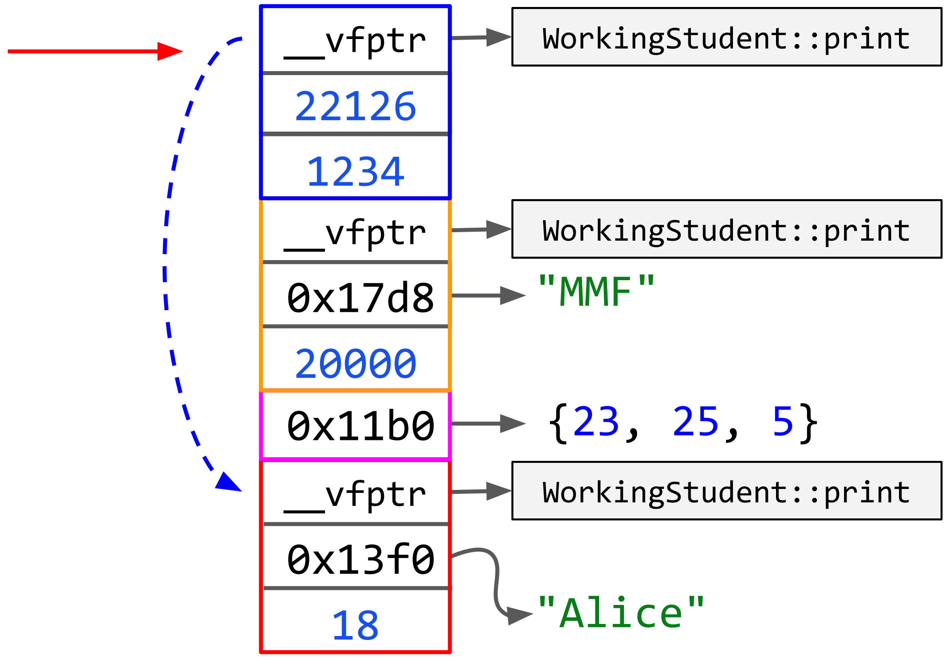
Basically, it doesn't matter, where to place this "common" part, let's put all at the end of structure (later will be clear one).

How to fix that?

How can I access Person::name, if this points to Student?

The problem is that the offset to name field is... unknown in compile time! (can't know if you have 2 bases or 1, or 3, except Student)





Basically, it doesn't matter, where to place this "common" part, let's put all at the end of structure (later will be clear one).

How to fix that? VMT!

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

void test_s(Student& s) {
    printf("%s", s.name);
}


int main() {
    WorkingStudent ws;
    test_p(ws);
    test_s(ws);

}
```

```asm
test_s(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]
        sub     rax, 24
        mov     rax, QWORD PTR [rax]
        mov     rdx, rax
        mov     rax, QWORD PTR [rbp-8]
        add     rax, rdx
        mov     rax, QWORD PTR [rax+8]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        nop
        leave
        ret
```

after some dereferences, rdx contains offset to Person.

https://godbolt.org/z/vefbo56cP

VMT has much more complex structure

(because of offsets to bases)

```
vtable for WorkingStudent:
        .quad   48
        .quad   0
        .quad   typeinfo for WorkingStudent
        .quad   WorkingStudent::print()
        .quad   32
        .quad   -16
        .quad   typeinfo for WorkingStudent
        .quad   non-virtual thunk to WorkingStudent::print()
        .quad   -48
        .quad   -48
        .quad   typeinfo for WorkingStudent
        .quad   virtual thunk to WorkingStudent::print()
```

https://godbolt.org/z/vefbo56cP

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

void test_s(Student& s) {
    printf("%s", s.name);
}


int main() {
    WorkingStudent ws;
    test_p(ws);
    test_s(ws);

}
```

```
test_s(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]
        sub     rax, 24
        mov     rax, QWORD PTR [rax]
        mov     rdx, rax
        mov     rax, QWORD PTR [rbp-8]
        add     rax, rdx
        mov     rax, QWORD PTR [rax+8]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        nop
        leave
        ret
```

after some dereferences, rdx contains offset to Person.

https://godbolt.org/z/vefbo56cP

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

void test_s(Student& s) {
    printf("%s", s.name);
}


int main() {
    WorkingStudent ws;
    test_p(ws);
    test_s(ws);

}
```

```asm
test_s(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]
        sub     rax, 24
        mov     rax, QWORD PTR [rax]
        mov     rdx, rax
        mov     rax, QWORD PTR [rbp-8]
        add     rax, rdx
        mov     rax, QWORD PTR [rax+8]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        nop
        leave
        ret
```

offset to the base

https://godbolt.org/z/vefbo56cP

VMT has much more complex structure

(because of offsets to bases)

```
vtable for WorkingStudent:
        .quad   48
        .quad   0
        .quad   typeinfo for WorkingStudent
        .quad   WorkingStudent::print()
        .quad   32
        .quad   -16
        .quad   typeinfo for WorkingStudent
        .quad   non-virtual thunk to WorkingStudent::print()
        .quad   -48
        .quad   -48
        .quad   typeinfo for WorkingStudent
        .quad   virtual thunk to WorkingStudent::print()
```

https://godbolt.org/z/vefbo56cP

VMT has much more complex structure

(because of offsets to bases)

```
vtable for WorkingStudent:
        .quad   48
        .quad   0
        .quad   typeinfo for WorkingStudent
        .quad   WorkingStudent::print()
        .quad   32
        .quad   -16
        .quad   typeinfo for WorkingStudent
        .quad   non-virtual thunk to WorkingStudent::print()
        .quad   -48
        .quad   -48
        .quad   typeinfo for WorkingStudent
        .quad   virtual thunk to WorkingStudent::print()
```

VTT stands for virtual-table table

(used to update vmts in different constructors)

```
VTT for WorkingStudent:
        .quad   vtable for WorkingStudent+24
        .quad   construction vtable for Student-in-WorkingStudent+24
        .quad   construction vtable for Student-in-WorkingStudent+56
        .quad   construction vtable for Employee-in-WorkingStudent+24
        .quad   construction vtable for Employee-in-WorkingStudent+56
        .quad   vtable for WorkingStudent+88
        .quad   vtable for WorkingStudent+56
```

https://godbolt.org/z/vefbo56cP

Problem solved, but what did it cost?

Problem solved, but what did it cost?

1. Access to field of virtual bases are not free anymore (need to consult with VMT)

2. Objects could be even fattier (more __vfptrs, despite the removing of base duplication)

3. Constructors code is just crazy.

Do we really need non-virtual multiple inheritance?

Do we really need non-virtual multiple inheritance?

Sure! Just not in our original sample.

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

```cpp
class Student: virtual public Person {
protected:
    size_t group;
    size_t id;
public:
    void print() {
        std::cout << name
                  << age
                  << group << id;
    }
};
```

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

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

```cpp
class AccountHolder {
protected:
   size_t accountNumber;
   size_t balance;
};
```

```cpp
class Student: virtual public Person {
protected:
    size_t group;
    size_t id;
public:
    void print() {
        std::cout << name
                  << age
                  << group << id;
    }
};
```

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

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

```cpp
class AccountHolder {
protected:
   size_t accountNumber;
   size_t balance;
};
```

```cpp
class Student: virtual public Person,
               public AccountHolder {
protected:
    size_t group;
    size_t id;
public:
    void print() {
        ...
    }
};
```

```cpp
class Employee: virtual public Person,
                public AccountHolder {
protected:
    const char* department;
    size_t salary;
public:
    void print() {
        ...
    }
};
```

```cpp
class WorkingStudent: public Student,
                      public Employee {
protected:
   int* prohibitedDates;
public:
   void paySalary() {
       this->Employee::balance += salary;
   }

   void payScholarship(int scholarship) {
       this->Student::balance += scholarship;
   }
   ...
};
```

```
class WorkingStudent: public Student,
                      public Employee {
protected:
   int* prohibitedDates;
public:
   void paySalary() {
       this->Employee::balance += salary;
   }

   void payScholarship(int scholarship) {
       this->Student::balance += scholarship;
   }
   ...
};
```

Two bank accounts, for two purposes, very logical!

And again quite understandable,
why by default we don't have
virtual inheritance.

It is too expensive!

```
class WorkingStudent: public Student,
                      public Employee {
protected:
   int* prohibitedDates;
public:
   void paySalary() {
       this->Employee::balance += salary;
   }

   void payScholarship(int scholarship) {
       this->Student::balance += scholarship;
   }
   ...
};
```

Two bank accounts, for
two purposes, very logical!

Virtual Inheritance: static_cast

```cpp
void test_p(Person& p) { ... }

void test_s(Student& s) { ... }

void test_e(Employee& e) { ... }

void test_ws(WorkingStudent& ws) { ... }

int main() {
    WorkingStudent ws;
    test_p(ws);
    test_s(ws);
    test_e(ws);
    test_ws(ws);
}
```

```asm
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 80
        lea     rax, [rbp-80]
        mov     rdi, rax
        call    WorkingStudent::WorkingStudent()
        lea     rax, [rbp-80]
        add     rax, 48
        mov     rdi, rax
        call    test_p(Person&)
        lea     rax, [rbp-80]
        mov     rdi, rax
        call    test_s(Student&)
        lea     rax, [rbp-80]
        add     rax, 16
        mov     rdi, rax
        call    test_e(Employee&)
        lea     rax, [rbp-80]
        mov     rdi, rax
        call    test_ws(WorkingStudent&)
        mov     eax, 0
        leave
        ret
```

https://godbolt.org/z/hxPdnKx5K

Virtual Inheritance: static_cast

```
int main() {
    WorkingStudent ws;
    Person* p = &ws;
    Student* s = &ws;
    Employee* e = &ws;

}
```

```
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 96
        lea     rax, [rbp-96]
        mov     rdi, rax
        call    WorkingStudent::WorkingStudent()
        lea     rax, [rbp-96]
        add     rax, 48
        mov     QWORD PTR [rbp-8], rax
        lea     rax, [rbp-96]
        mov     QWORD PTR [rbp-16], rax
        lea     rax, [rbp-96]
        add     rax, 16
        mov     QWORD PTR [rbp-24], rax
        mov     eax, 0
        leave
        ret
```

https://godbolt.org/z/aMo43nEY1

Virtual Inheritance: static_cast

```
int main() {
    WorkingStudent ws;
    Person* p = &ws;
    Student* s = &ws;
    Employee* e = &ws;

    WorkingStudent* pws = static_cast<WorkingStudent*>(p);


}
```

https://godbolt.org/z/3eha6Te34

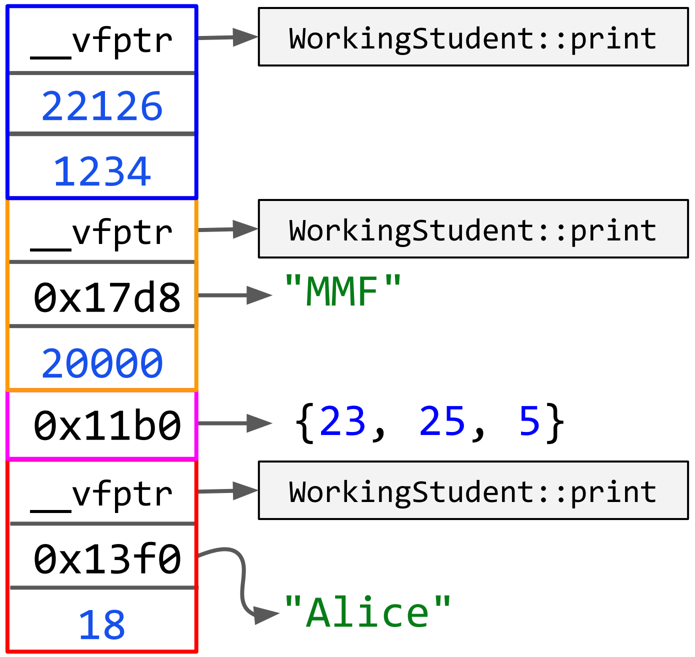
Virtual Inheritance: static_cast

```
int main() {
    WorkingStudent ws;
    Person* p = &ws;
    Student* s = &ws;
    Employee* e = &ws;

    WorkingStudent* pws = static_cast<WorkingStudent*>(p);
    // error: cannot convert from pointer to base class 'Person' to pointer to
    // derived class 'WorkingStudent' because the base is virtual
}
```

https://godbolt.org/z/3eha6Te34

Virtual Inheritance: static_cast

```
int main() {
    WorkingStudent ws;
    Person* p = &ws;
    Student* s = &ws;
    Employee* e = &ws;

    WorkingStudent* pws =
    static_cast<WorkingStudent*>(p);
    // error
}
```

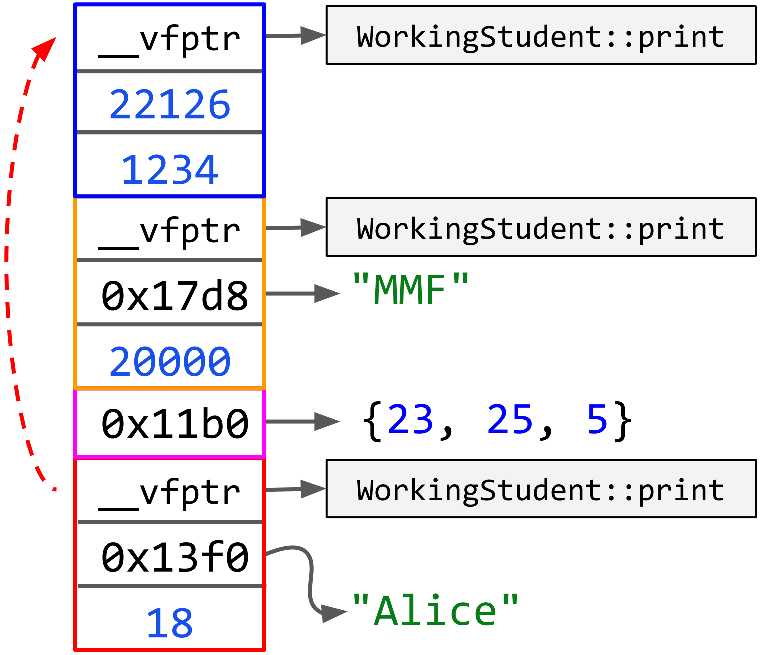


https://godbolt.org/z/3eha6Te34

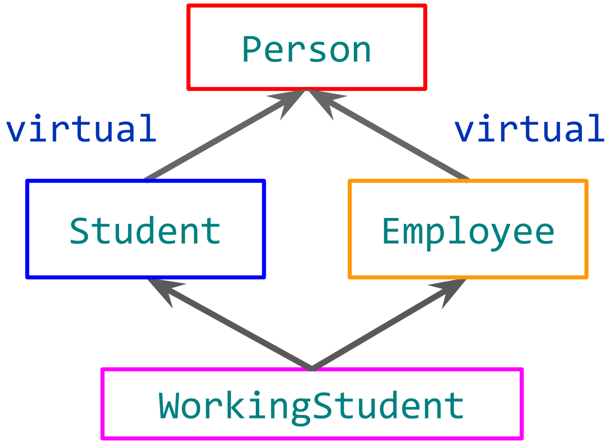
Virtual Inheritance: static_cast

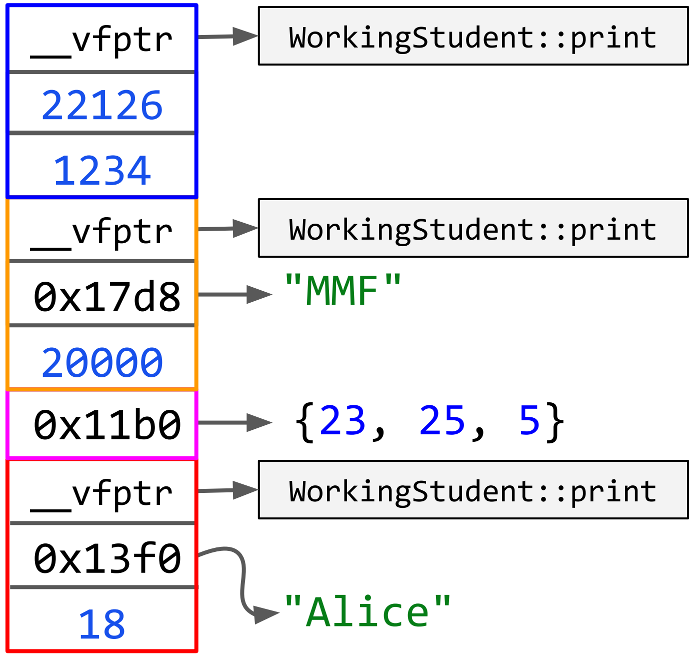
```
int main() {
    WorkingStudent ws;
    Person* p = &ws;
    Student* s = &ws;
    Employee* e = &ws;

    WorkingStudent* pws =
    static_cast<WorkingStudent*>(p);
    // error
}
```

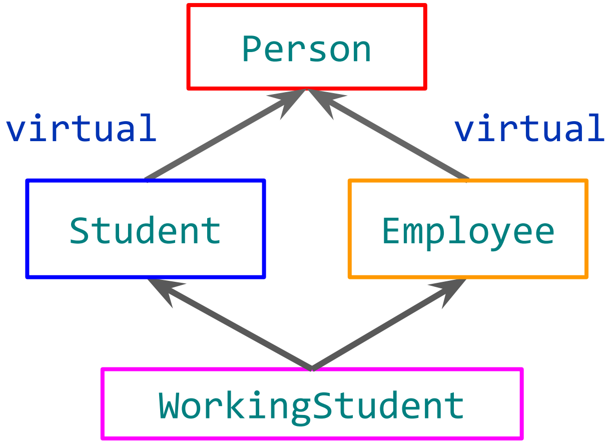
Do we know this distance statically?

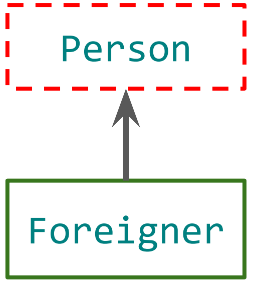


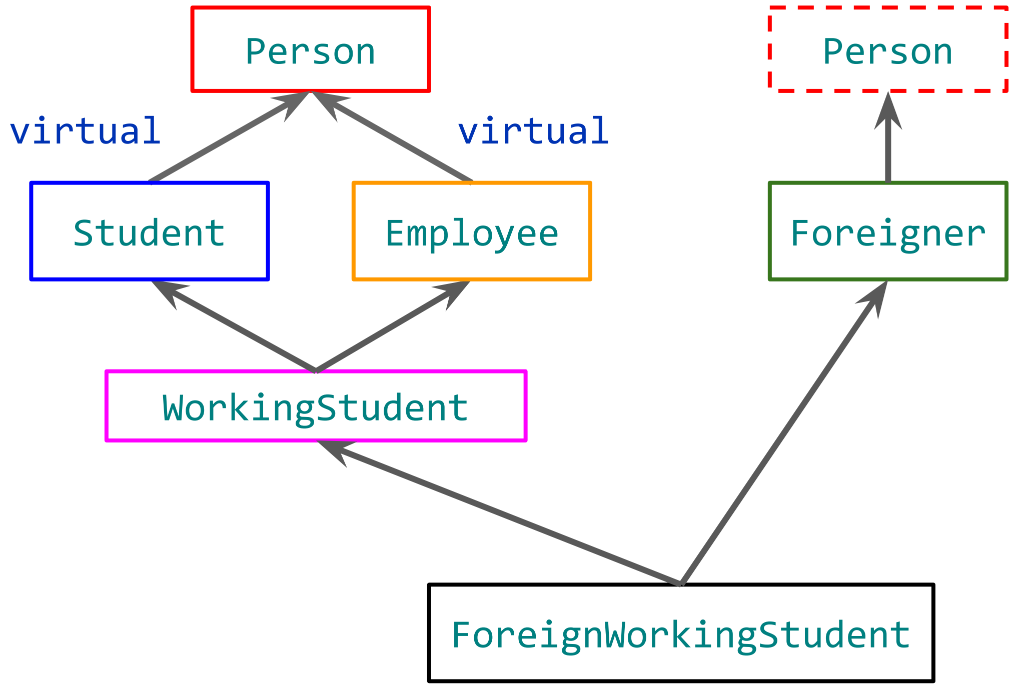
https://godbolt.org/z/3eha6Te34

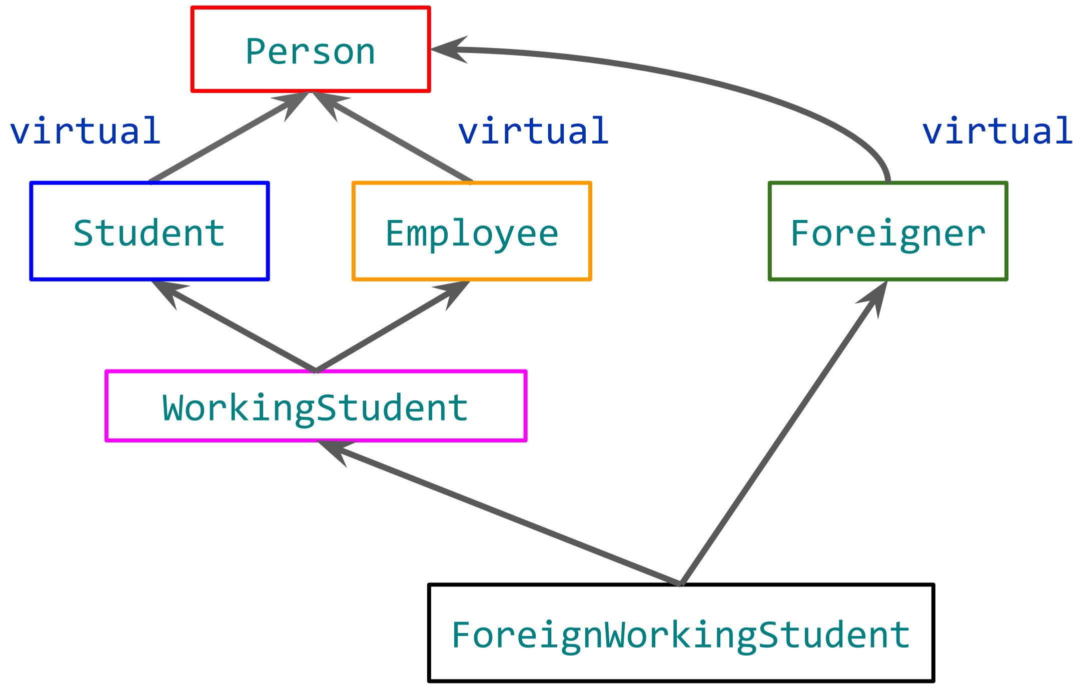

Person
virtual
virtual
Student
Employee
WorkingStudent

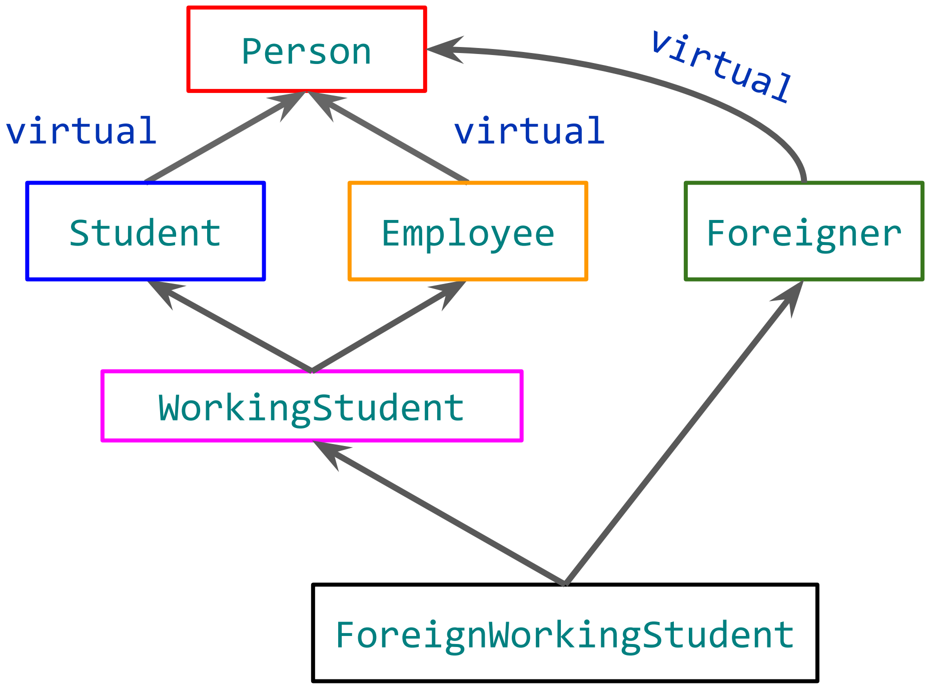
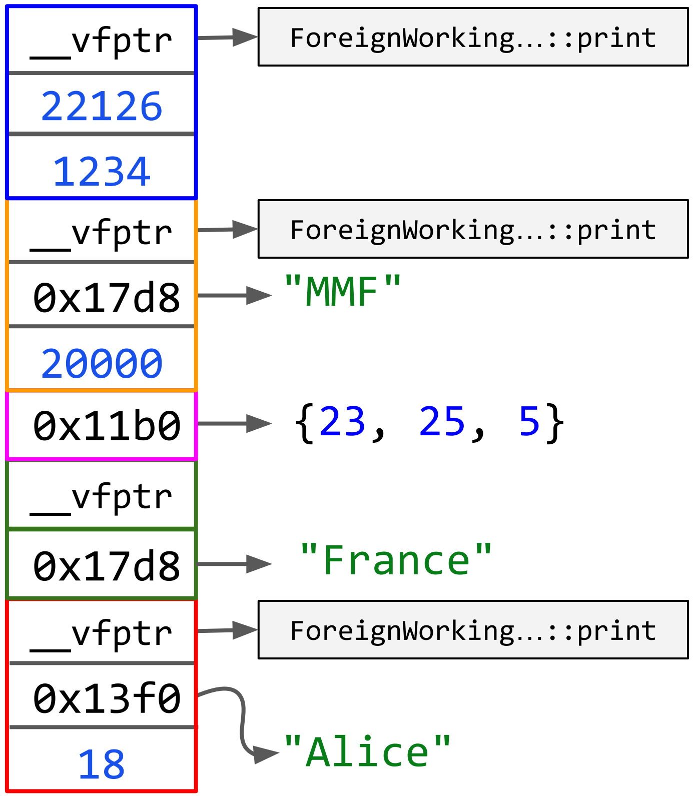

__vfptr
WorkingStudent::print
22126
1234
__vfptr
WorkingStudent::print
0x17d8
"MMF"
20000
0x11b0
{23, 25, 5}
__vfptr
WorkingStudent::print
0x13f0
"Alice"
18

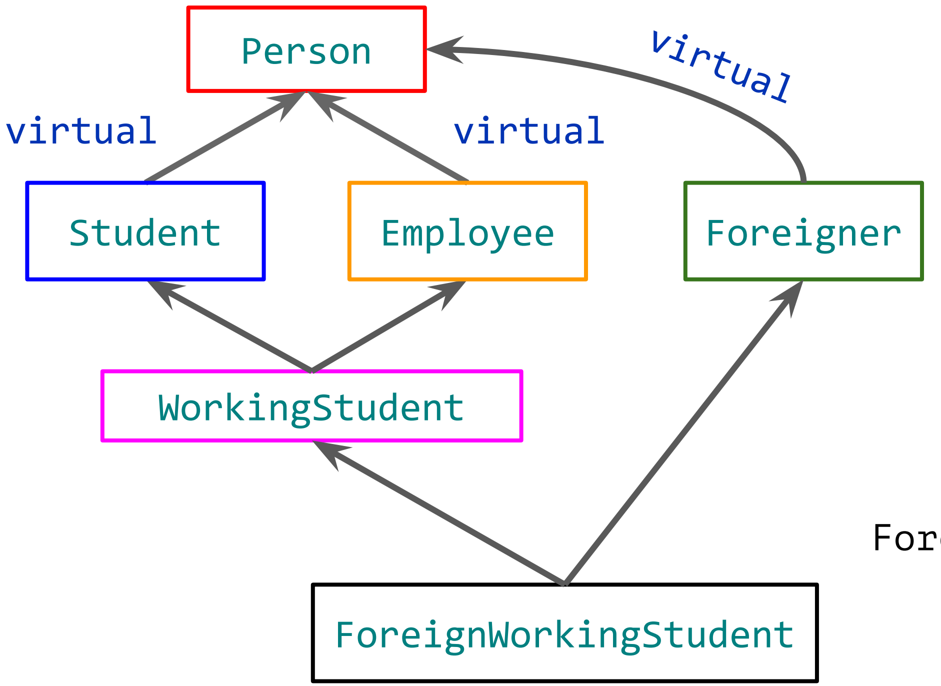
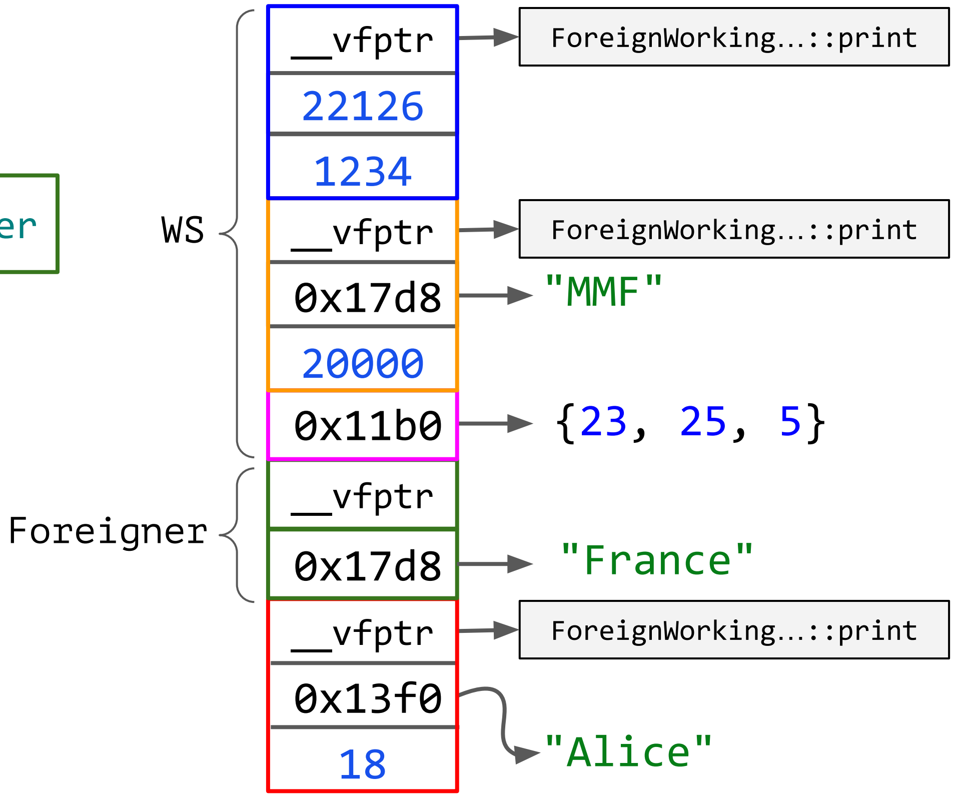

Person
virtual
virtual
Student
Employee
WorkingStudent


Person
Foreigner

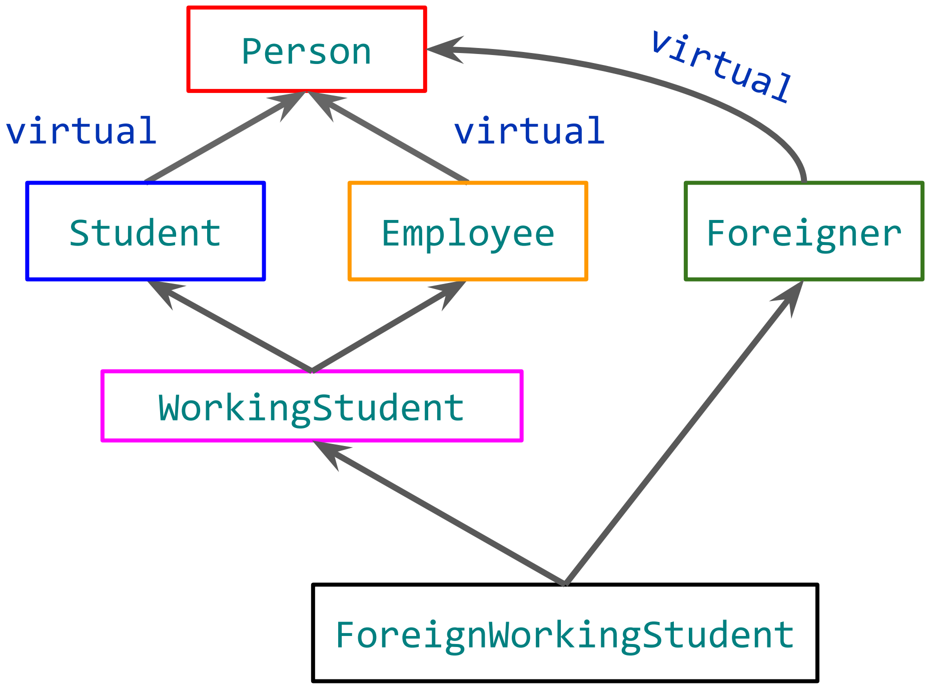

Person
virtual
virtual
Person
Student
Employee
Foreigner
WorkingStudent
ForeignWorkingStudent

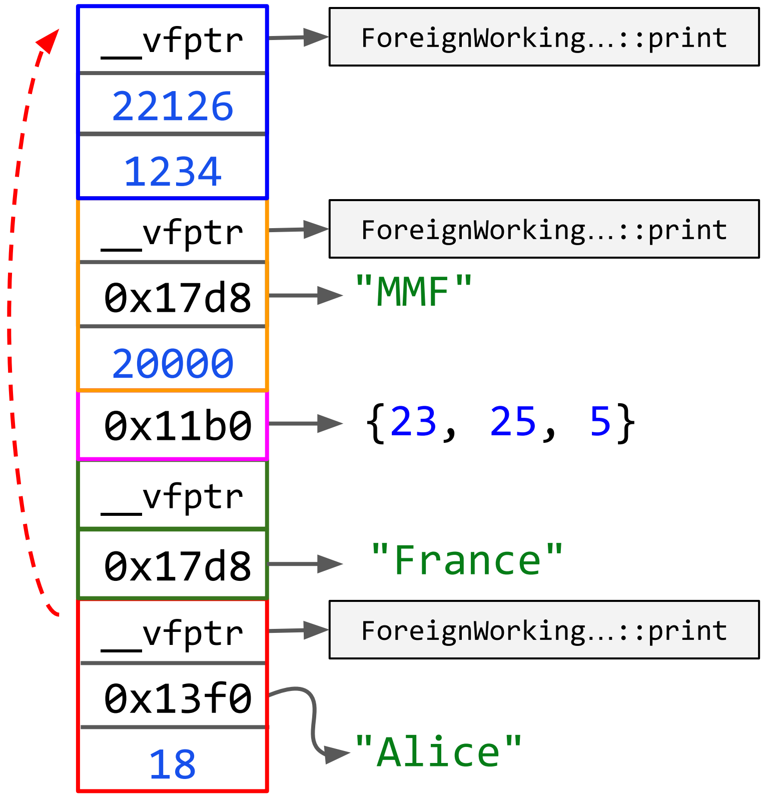

Person
virtual
virtual
virtual
Student
Employee
Foreigner
WorkingStudent
ForeignWorkingStudent


Person
virtual
virtual
virtual
Student
Employee
Foreigner
WorkingStudent
ForeignWorkingStudent

__vfptr
22126
1234
ForeignWorking…::print
__vfptr
ForeignWorking…::print
0x17d8
"MMF"
20000
0x11b0
{23, 25, 5}
__vfptr
0x17d8
"France"
__vfptr
ForeignWorking…::print
0x13f0
"Alice"
18


Person
virtual
virtual
virtual
Student
Employee
Foreigner
WorkingStudent
ForeignWorkingStudent

WS
Foreigner
__vfptr
22126
1234
__vfptr
0x17d8
20000
0x11b0
__vfptr
0x17d8
__vfptr
0x13f0
18
ForeignWorking…::print
ForeignWorking…::print
ForeignWorking…::print
"MMF"
{23, 25, 5}
"France"
"Alice"

In such case distance from Person to WorkingStudent is 72 bytes.

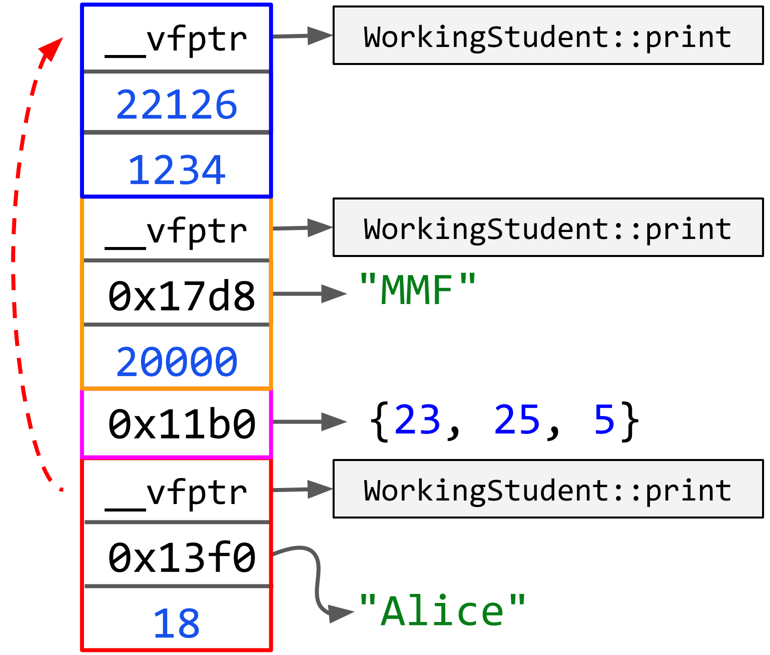




But in such case distance from Person to WorkingStudent is 56 bytes!

```
int main() {
    WorkingStudent ws;
    Person* p = &ws;
    Student* s = &ws;
    Employee* e = &ws;

    WorkingStudent* pws =
   static_cast<WorkingStudent*>(p);
   // error
}
```

Do we know this distance statically?



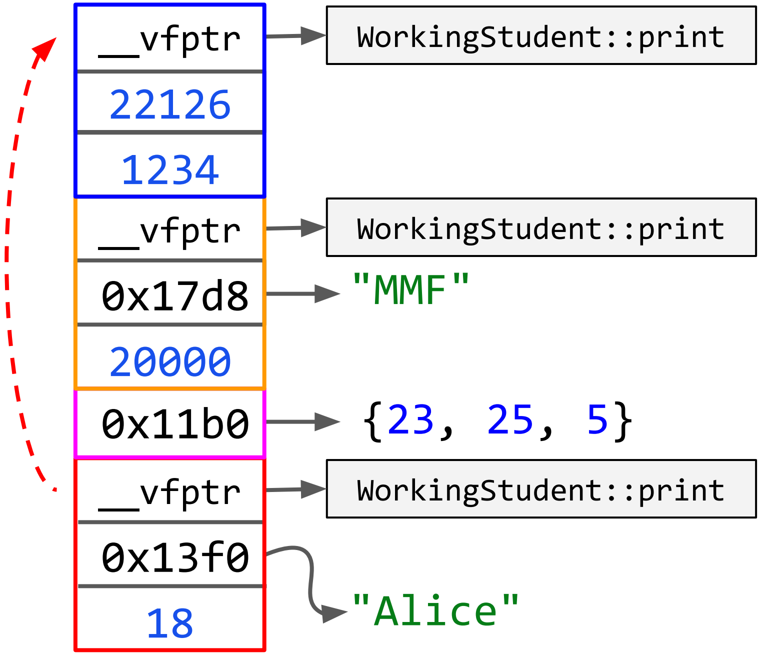
https://godbolt.org/z/3eha6Te34

So, by looking at some Person at compile-time it is not clear where to jump for the cast!

But in such case distance from Person to WorkingStudent is 56 bytes!

```
int main() {
    WorkingStudent ws;
    Person* p = &ws;
    Student* s = &ws;
    Employee* e = &ws;

    WorkingStudent* pws =
    static_cast<WorkingStudent*>(p);
    // error
}
```

Do we know this distance statically?



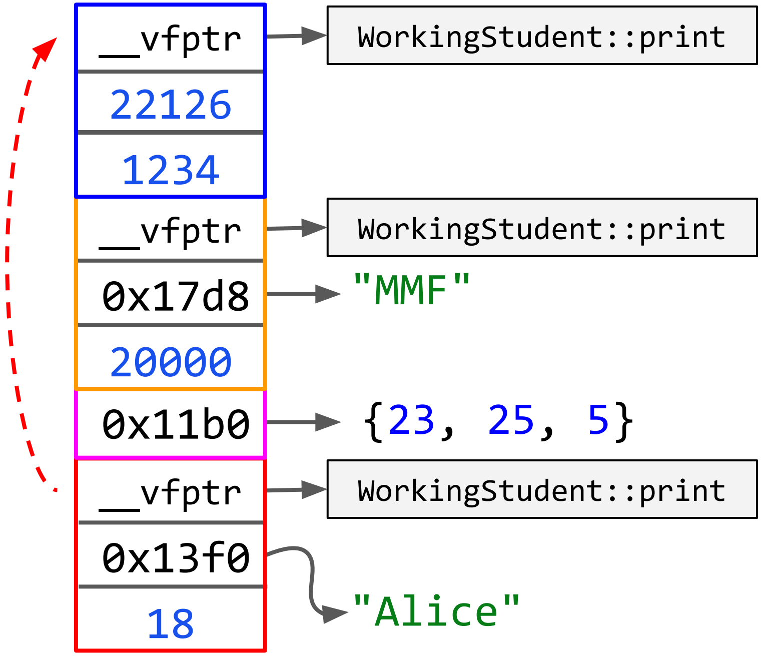
https://godbolt.org/z/3eha6Te34

So, by looking at some Person at compile-time it is not clear where to jump for the cast!

```
int main() {
    WorkingStudent ws;
    Person* p = &ws;
    Student* s = &ws;
    Employee* e = &ws;

    WorkingStudent* pws =
   static_cast<WorkingStudent*>(p);
   // error
}
```

Do we know this distance statically?

No!
You need to consult with VMT



https://godbolt.org/z/3eha6Te34

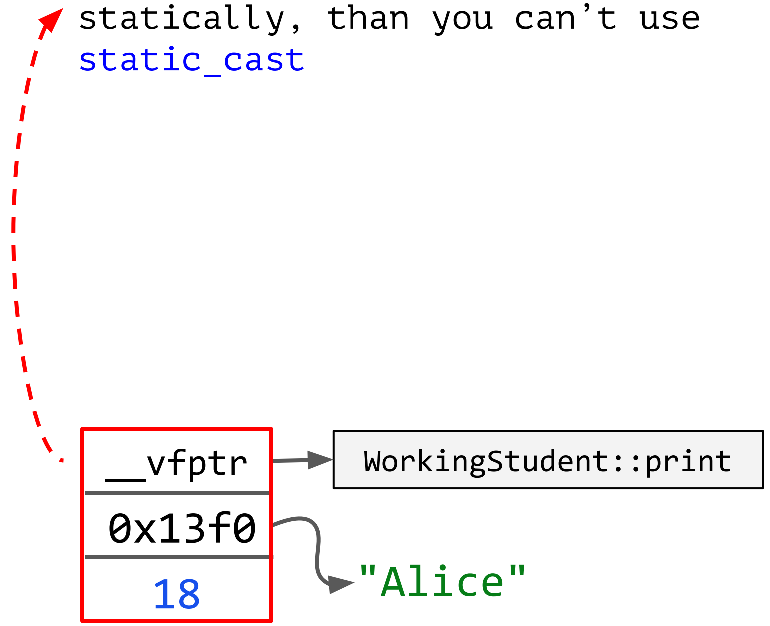
So, by looking at some Person at compile-time it is not clear where to jump for the cast!

```
int main() {
    WorkingStudent ws;
    Person* p = &ws;
    Student* s = &ws;
    Employee* e = &ws;

    WorkingStudent* pws =
   static_cast<WorkingStudent*>(p);
   // error
}
```

Do we know this distance statically?

No!
You need to consult with VMT

If cast can't be proceed statically, than you can't use static_cast



https://godbolt.org/z/3eha6Te34

So, by looking at some
Person at compile-time
it is not clear where to
jump for the cast!

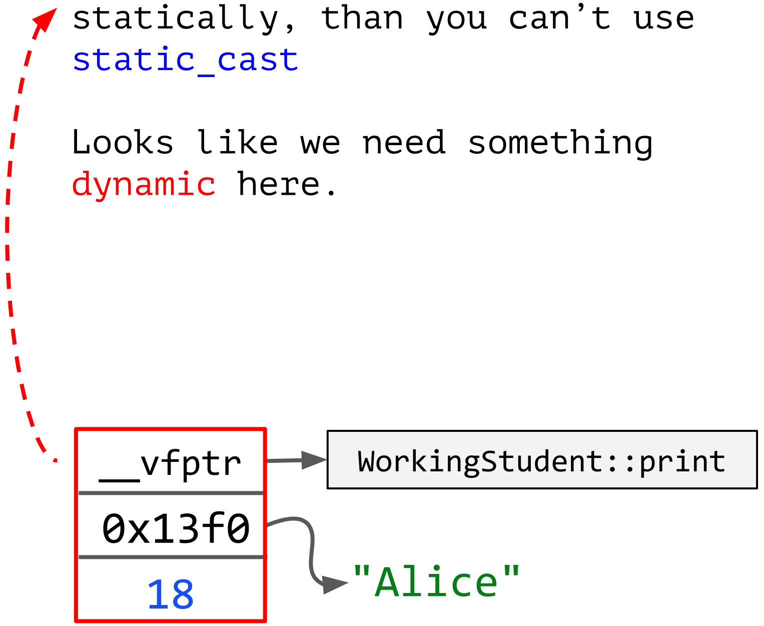
```
int main() {
    WorkingStudent ws;
    Person* p = &ws;
    Student* s = &ws;
    Employee* e = &ws;

    WorkingStudent* pws =
   static_cast<WorkingStudent*>(p);
   // error
}
```

Do we know
this distance
statically?

No!
You need to
consult with
VMT

If cast can't be proceed
statically, than you can't use
static_cast

Looks like we need something
dynamic here.



https://godbolt.org/z/3eha6Te34

RTTI

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
```

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
```

wtf is this? → `.quad   typeinfo for Person`

RTTI

```
int main() {
    WorkingStudent ws;

    Student& s = ws;
    Employee& e = ws;

    std::cout << typeid(s).name() << std::endl;
}
```

```
vtable for WorkingStudent:
        .quad   48
        .quad   0
        .quad   typeinfo for WorkingStudent
        .quad   WorkingStudent::print()
        .quad   32
        .quad   -16
        .quad   typeinfo for WorkingStudent
        .quad   non-virtual thunk to WorkingStudent::print()
        .quad   -48
        .quad   -48
        .quad   typeinfo for WorkingStudent
        .quad   virtual thunk to WorkingStudent::print()
```

RTTI

```
int main() {
    WorkingStudent ws;

    Student& s = ws;
    Employee& e = ws;

    std::cout << typeid(s).name() << std::endl;
    // 14WorkingStudent
}
```

RTTI

```
vtable for WorkingStudent:
        .quad   48
        .quad   0
        .quad   typeinfo for WorkingStudent
        .quad   WorkingStudent::print()
        .quad   32
        .quad   -16
        .quad   typeinfo for WorkingStudent
        .quad   non-virtual thunk to WorkingStudent::print()
        .quad   -48
        .quad   -48
        .quad   typeinfo for WorkingStudent
        .quad   virtual thunk to WorkingStudent::print()
```

```
int main() {
    WorkingStudent ws;

    Student& s = ws;
    Employee& e = ws;

    std::cout << typeid(s).name() << std::endl;
    // 14WorkingStudent
}
```

Compiler can’t know that s is of type WS&, for him it is Student&.

typeid(T) takes this field of VMT to get REAL type of an object.

RTTI

```
vtable for WorkingStudent:
        .quad   48
        .quad   0
        .quad   typeinfo for WorkingStudent
        .quad   WorkingStudent::print()
        .quad   32
        .quad   -16
        .quad   typeinfo for WorkingStudent
        .quad   non-virtual thunk to WorkingStudent::print()
        .quad   -48
        .quad   -48
        .quad   typeinfo for WorkingStudent
        .quad   virtual thunk to WorkingStudent::print()
```

```
int main() {
    WorkingStudent ws;

    Student& s = ws;
    Employee& e = ws;

    std::cout << typeid(s).name() << std::endl;
    // 14WorkingStudent
}
```

typeid(T) takes this field of VMT to get REAL type of an object.

Works only for classes with VMT. Otherwise, it will return static type (that compiler sees).

RTTI -> Run-Time Type Information

```
int main() {
    WorkingStudent ws;

    Student& s = ws;
    Employee& e = ws;

    std::cout << typeid(s).name() << std::endl;
    // 14WorkingStudent
}
```

typeid(T) takes this field of VMT to get REAL type of an object.

Works only for classes with VMT. Otherwise, it will return static type (that compiler sees).

RTTI -> Run-Time Type Information

A bit strange feature for C++: by default you pay for something you may not need!

```
int main() {
    WorkingStudent ws;

    Student& s = ws;
    Employee& e = ws;

    std::cout << typeid(s).name() << std::endl;
    // 14WorkingStudent
}
```

`typeid(T) takes this field of VMT to get REAL type of an object.`

`Works only for classes with VMT. Otherwise, it will return static type (that compiler sees).`

RTTI -> Run-Time Type Information

```
int main() {
    WorkingStudent ws;

    Student& s = ws;
    Employee& e = ws;

    std::cout << typeid(s).name() << std::endl;
    // 14WorkingStudent
}
```

A bit strange feature for C++: by default you pay for something you may not need!

It is often disabled to reduce code and data size.

`typeid(T)` takes this field of VMT to get REAL type of an object.

Works only for classes with VMT. Otherwise, it will return static type (that compiler sees).

dynamic_cast

dynamic_cast

```
void check_if_student(Person* p) {
    p->print();

    Student* st = dynamic_cast<Student*>(p);
    if (st == nullptr) {
        std::cout << "cast failed" << std::endl;
    } else {
        std::cout << "successfully casted" << std::endl;
    }
}
```

dynamic_cast

uses RTTI to cast to some class from hierarchy

```
void check_if_student(Person* p) {
    p->print();

    Student* st = dynamic_cast<Student*>(p);
    if (st == nullptr) {
        std::cout << "cast failed" << std::endl;
    } else {
        std::cout << "successfully casted" << std::endl;
    }
}
```

dynamic_cast

uses RTTI to cast to some class from hierarchy

```
void check_if_student(Person* p) {
    p->print();

    Student* st = dynamic_cast<Student*>(p);
    if (st == nullptr) {
        std::cout << "cast failed" << std::endl;
    } else {
        std::cout << "successfully casted" << std::endl;
    }
}
```

if there is no such class in hierarchy, returns nullptr

```cpp
void check_if_student(Person* p) {
    p->print();

    Student* st = dynamic_cast<Student*>(p);
    if (st == nullptr) {
        std::cout << "cast failed" << std::endl;
    } else {
        std::cout << "successfully casted" << std::endl;
    }
}

int main() {
    WorkingStudent ws;
    Student s;
    Employee e;
    ForeignWorkingStudent fws;

    check_if_student(&ws);
    check_if_student(&s);
    check_if_student(&e);
    check_if_student(&fws);
```

```cpp
void check_if_student(Person* p) {
    p->print();

    Student* st = dynamic_cast<Student*>(p);
    if (st == nullptr) {
        std::cout << "cast failed" << std::endl;
    } else {
        std::cout << "successfully casted" << std::endl;
    }
}

int main() {
    WorkingStudent ws;
    Student s;
    Employee e;
    ForeignWorkingStudent fws;

    check_if_student(&ws);  ──────▶ "successfully casted"
    check_if_student(&s);   ──────▶ "successfully casted"
    check_if_student(&e);
    check_if_student(&fws); ──────▶ "successfully casted"
```

```cpp
void check_if_student(Person* p) {
    p->print();

    Student* st = dynamic_cast<Student*>(p);
    if (st == nullptr) {
        std::cout << "cast failed" << std::endl;
    } else {
        std::cout << "successfully casted" << std::endl;
    }
}

int main() {
    WorkingStudent ws;
    Student s;
    Employee e;
    ForeignWorkingStudent fws;

    check_if_student(&ws);  → "successfully casted"
    check_if_student(&s);   → "successfully casted"
    check_if_student(&e);   → "cast failed"
    check_if_student(&fws); → "successfully casted"
```

```cpp
void check_if_student(Person* p) {
    p->print();

    Student* st = dynamic_cast<Student*>(p);
    if (st == nullptr) {
        std::cout << "cast failed"
                  << std::endl;
    } else {
        std::cout << "successfully casted"
                  << std::endl;
    }
}
```

```asm
check_if_student(Person*):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     QWORD PTR [rbp-24], rdi
        mov     rax, QWORD PTR [rbp-24]
        test    rax, rax
        je      .L8
        mov     rcx, -1
        mov     edx, OFFSET FLAT:typeinfo for Student
        mov     esi, OFFSET FLAT:typeinfo for Person
        mov     rdi, rax
        call    __dynamic_cast
        jmp     .L9
```

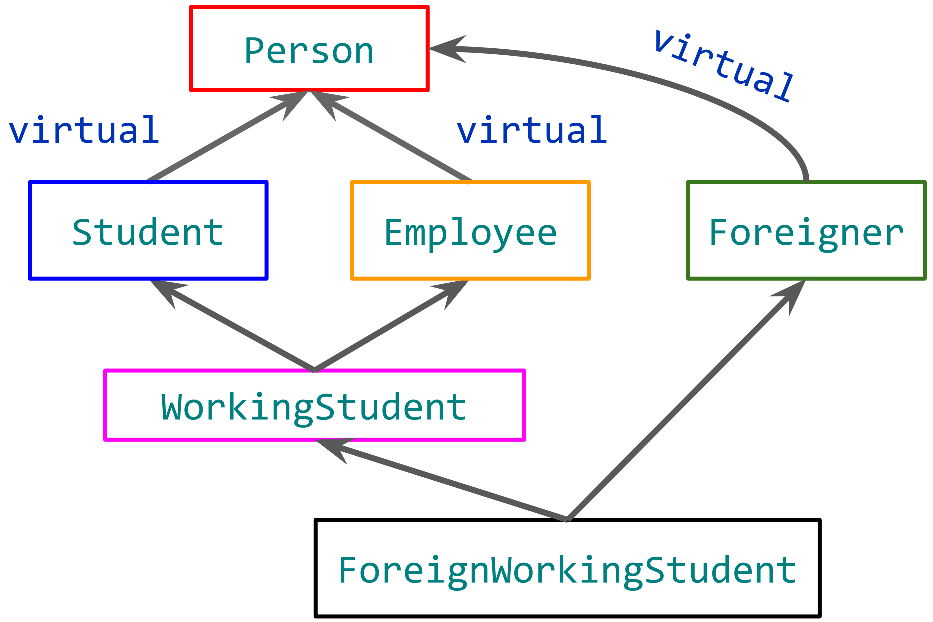
https://godbolt.org/z/zaYc9dY6o

dynamic_cast

1. Can cast to any type from hierarchy (for example, works as side cast!)

dynamic_cast

1. Can cast to any type from hierarchy (for example, works as side cast!)



We can cast Foreigner* to Student* for example!

dynamic_cast

1. Can cast to any type from hierarchy (for example, works as side cast!)

2. Really heavy operation as it iterates over the hierarchy of classes.

dynamic_cast

1. Can cast to any type from hierarchy (for example, works as side cast!)

2. Really heavy operation as it iterates over the hierarchy of classes.

3. For classes without VMT or when RTTI is disabled, works as static_cast (or can be just prohibited by the compiler).

dynamic_cast

1. Can cast to any type from hierarchy (for example, works as side cast!)

2. Really heavy operation as it iterates over the hierarchy of classes.

3. For classes without VMT or when RTTI is disabled, works as static_cast (or can be just prohibited by the compiler).

4. If used in constructors/destructors for down casting causes UB.

dynamic_cast -> very controversial feature in C++

1. Can cast to any type from hierarchy (for example, works as side cast!)

2. Really heavy operation as it iterates over the hierarchy of classes.

3. For classes without VMT or when RTTI is disabled, works as static_cast (or can be just prohibited by the compi

4. If used in constructors/dest casting causes UB.

Multiple Inheritance

1. Such a pain for compiler/runtime developers,

Multiple Inheritance

1. Such a pain for compiler/runtime developers,
2. Not such a big pain for users though (but has its pitfalls again),

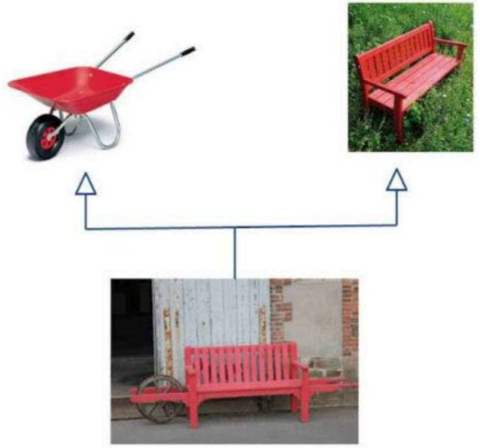
Multiple Inheritance

1. Such a pain for compiler/runtime developers,
2. Not such a big pain for users though (but has its pitfalls again),
3. Many languages somehow limit multiple inheritance or just prohibit it. Not C++.

Multiple Inheritance

1. Such a pain for compiler/runtime developers,
2. Not such a big pain for users though (but has its pitfalls again),
3. Many languages somehow limit multiple inheritance or just prohibit it. Not C++.
4. With great power comes great responsibility:
    a. Performance/memory costs
    b. Bad architecture can be built on it (as well as very beautiful one)

Multiple Inheritance



1. Such a pain for compiler/runtime
2. Not such a big pain for users,
3. Many languages somehow limit multiple inheritance or just prohibit it. Not C++.
4. With great power comes great responsibility:
   a. Performance/memory costs
   b. Bad architecture can be built on it (as well as very beautiful one)

```cpp
void check_if_student(Person* p) {
    p->print();

    Student* st = dynamic_cast<Student*>(p);
    if (st == nullptr) {
        std::cout << "cast failed"
                  << std::endl;
    } else {
        std::cout << "successfully casted"
                  << std::endl;
    }
}
```

```cpp
void check_if_student(Person& p) {
    p->print();

    Student& st = dynamic_cast<Student&>(p);
    // how should we understand whether cast failed or not???
}
```

Not So Tiny Task №8 (2 + 1 points)

Implement a hierarchy for reading/writing data from/to some source.

2 points

- Base class: IO; Should provide some basic information: if source is still open or not (can be closed manually by close() method), was eof reached or not.
- 1st level of derived classes: Reader and Writer; They provide functions for reading/writing primitive types (and std::strings).
- 2nd level of derived classes: ReaderWriter. It provides functions for reading and writing at the same time.
- 3rd level: specific implementation for different sources 1) std::string as a source, 2) FILE* as a source.

Not So Tiny Task №8 (2 + 1 points)

Implement a hierarchy for reading/writing data from/to some source.

…

- 3rd level: specific implementation for different sources 1) std::string as a source, 2) FILE* as a source.

- 4th level: implementation for both string and FILE* sources with buffer.

+1 point

  - Operations firstly read/write from/to the preallocated buffer of fixed size.

  - If buffer is empty/full, classes should read/write to the real source (string or file).