Bahasa C++: Keyword "virtual"

IF2210 – Semester II 2022/2023

Keyword `virtual` di C++

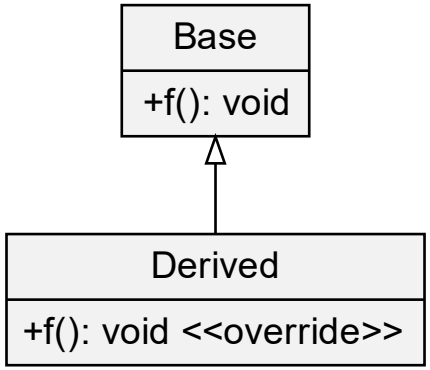
- Keyword `virtual` memiliki dua kegunaan:
  - *Specifier* untuk method
  - *Specifier* untuk kelas dasar (*base class*)

Keyword `virtual` sebagai *specifier* untuk method

Virtual method

- Virtual method adalah method yang perilakunya dapat di-*override* oleh *derived class*.
- Jika objek dari *derived class* ditangani dengan ptr atau ref, maka pemanggilan method akan menginvokasi perilaku hasil *override*. (ingat kembali *polymorphism*)
- Invokasi perilaku method asli (milik *base class*) dapat dilakukan dengan menggunakan *scope* (`::`) *base class*.
- Jika method tidak virtual, pemanggilan method pada *derived class* akan menginvokasi perilaku milik *base class*.

```cpp
#include <iostream>

class Base {
   virtual void f() {
       std::cout << "base\n";
   }
};

class Derived: Base {
    void f() override { // keyword 'override' tidak wajib
        std::cout << "derived\n";
    }
};
```



Notasi pada kuliah ini: diagram kelas (UML) yang dimodifikasi untuk memudahkan ilustrasi (jangan ditiru)

```
int main() {
    Base b;
    Derived d;

    // non-ptr, non-ref: non-virtual function call
    Base b1 = b;   // the type of b1 is Base
    Base d1 = d;   // the type of d1 is Base as well
    b1.f();        // prints "base"
    d1.f();        // prints "base" as well

    // virtual function call through reference
    Base& br = b;  // the type of br is Base&
    Base& dr = d;  // the type of dr is Base& as well
    br.f();        // prints "base"
    dr.f();        // prints "derived“

    // virtual function call through pointer
    Base* bp = &b; // the type of bp is Base*
    Base* dp = &d; // the type of dp is Base* as well
    bp->f();       // prints "base"
    dp->f();       // prints "derived"

    // non-virtual function call
    br.Base::f();  // prints "base"
    dr.Base::f();  // prints "base"
}
```

In detail... (1)

- Jika:
  - method `f` dideklarasikan sebagai virtual di kelas `Base`,
  - terdapat kelas `Derived`, yang diturunkan (langsung maupun tidak) dari kelas `Base`, memiliki deklarasi method dengan kesamaan: **nama**, **signature**, **cv-qualifiers**, dan **ref-qualifiers**,
- maka method tsb. di kelas `Derived`:
  - juga bersifat virtual (meski tanpa *keyword* `virtual`), dan
  - meng-*override* `Base::f` (meski tanpa *keyword* `override`)

In detail... (2)

- `Base::f` tidak perlu *visible* (dapat dideklarasikan `private`, atau diwariskan dengan mode akses `private`) untuk bisa di-*override*.

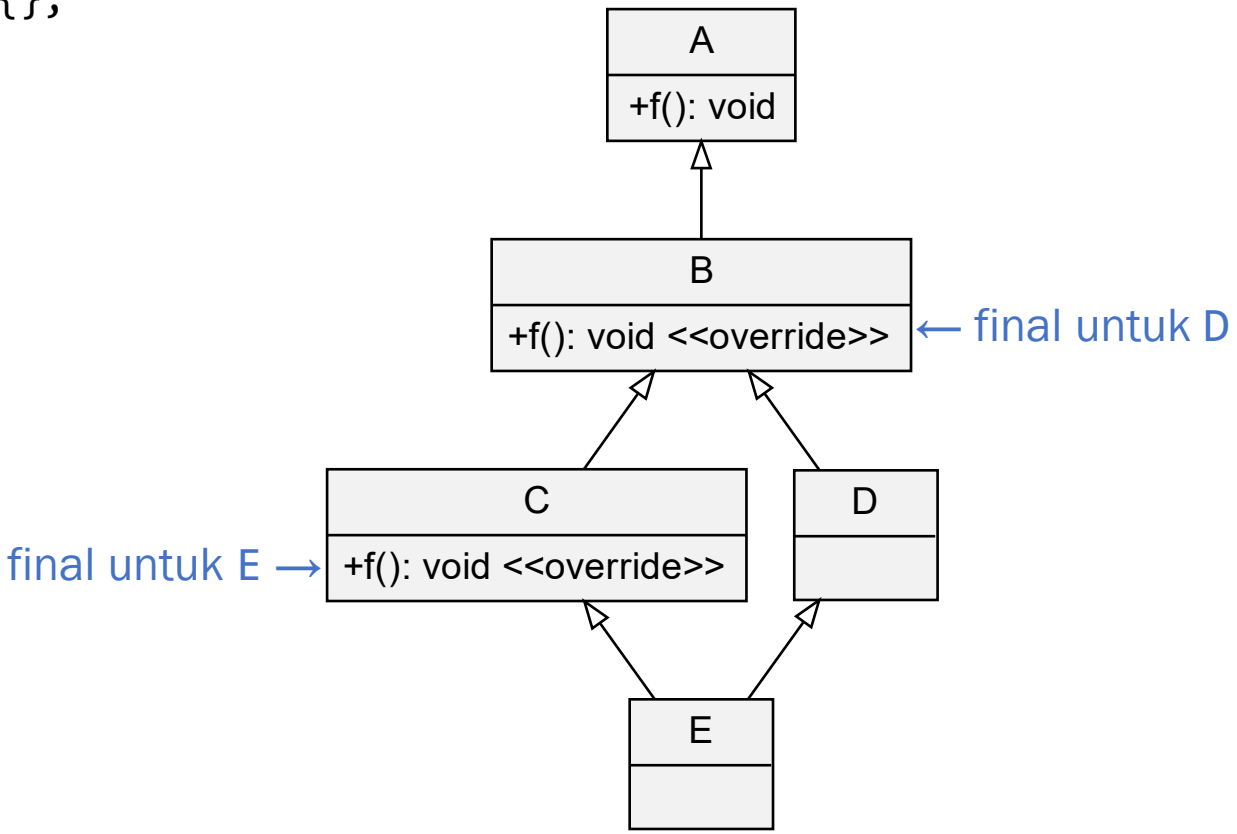
```
class B {
  private:
    virtual void do_f();  // private member
  public:
    void f() { do_f(); }  // public interface
};

class D: public B {
  public:
    void do_f() override; // overrides B::do_f
};

int main() {
    D d;
    B* bp = &d;
    bp->f(); // internally calls D::do_f();
}
```

Final overrider

- *Final overrider* adalah method **sebenarnya** yang dieksekusi saat *run-time* ketika method virtual dipanggil.
  - Saat *compile-time*, *compiler* tidak tahu implementasi mana yang akan dieksekusi
- Method virtual `f` milik kelas `Base` adalah *final overrider* kecuali kelas turunannya mendeklarasikan atau mewariskan method lain yang meng-*override* `f`.

```
struct A { virtual void f(); };     /* A::f is virtual          */
struct B: A { void f(); };          /* B::f overrides A::f in B */
struct C: virtual B { void f(); }; /* C::f overrides A::f in C */
struct D: virtual B {};             /* D does not introduce an overrider,
                                       B::f is final in D */
struct E: C, D {                    /* E does not introduce an overrider,
                                       C::f is final in E */
    using A::f;   /* not a function declaration,
                     just makes A::f visible to lookup */
};

int main() {
   E e;
   e.f();    /* virtual call calls C::f, the final overrider in e  */
   e.E::f(); /* non-virtual call calls A::f, which is visible in E */
}


// struct = class dengan hak akses member default adalah public
```

```
struct A { virtual void f(); };
struct B: A { void f(); };
struct C: virtual B { void f(); };
struct D: virtual B {};

struct E: C, D {

    using A::f;

};

int main() {
   E e;
   e.f();
   e.E::f();
}
```

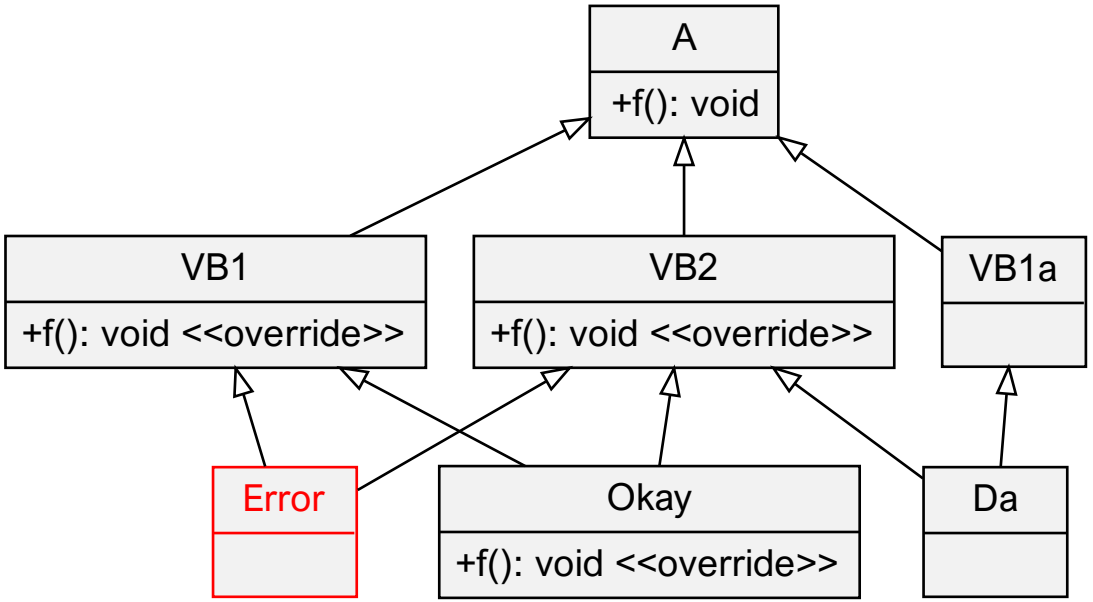


Final overrider: conflict

```
struct A {
    virtual void f();
};
struct VB1: virtual A {
    void f(); // overrides A::f
};
struct VB2: virtual A {
    void f(); // overrides A::f
};
// struct Error: VB1, VB2 {
//             // Error: A::f has two final overriders in Error
// };
struct Okay: VB1, VB2 {
    void f(); // OK: this is the final overrider for A::f
};
struct VB1a: virtual A {}; // does not declare an overrider
struct Da: VB1a, VB2 {
    // in Da, the final overrider of A::f is VB2::f
};
```

Final overrider: conflict

```
struct A {
    virtual void f();
};
struct VB1: virtual A {
    void f();
};
struct VB2: virtual A {
    void f();
};
// struct Error: VB1, VB2 {
//
// };
struct Okay: VB1, VB2 {
    void f();
};
struct VB1a: virtual A {};
struct Da: VB1a, VB2 {

};
```

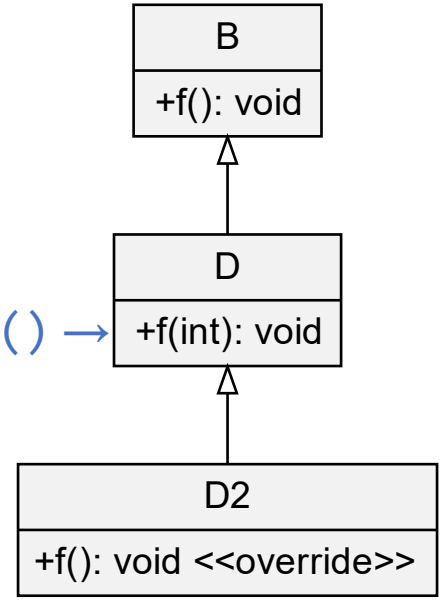


Hiding virtual function

- Method dengan nama yang sama tapi memiliki signature yang berbeda tidak meng-*override* method kelas dasar, tapi menyembunyikannya.

```
struct B {
    virtual void f();
};
struct D: B {
    void f(int); // D::f hides B::f (wrong parameter list)
};
struct D2: D {
    void f();    // D2::f overrides B::f (doesn't matter that it's not visible)
};

int main() {

    B b;   B& b_as_b   = b;
    D d;   B& d_as_b   = d;    D& d_as_d = d;
    D2 d2; B& d2_as_b  = d2;   D& d2_as_d = d2;

    b_as_b.f();  // calls B::f()
    d_as_b.f();  // calls B::f()
    d2_as_b.f(); // calls D2::f()

    d_as_d.f();  // Error: lookup in D finds only f(int)
    d2_as_d.f(); // Error: lookup in D finds only f(int)
}
```

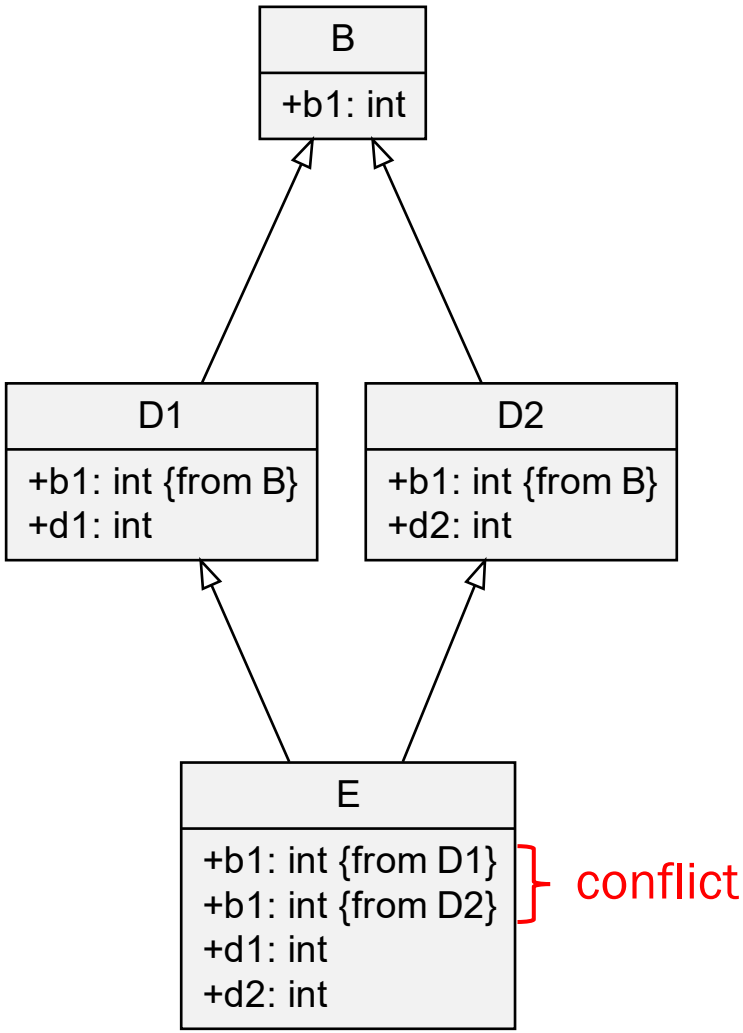
Menyembunyikan `B::f()` →



Keyword `virtual` sebagai *specifier* kelas dasar

Mengingat kembali: *inheritance* & DDD

- Jika D turunan dari B, maka D mempunyai seluruh anggota B ditambah anggota D sendiri. Contoh:
  - Jika B memiliki anggota b1,
    - ... dan D1 adalah turunan B dengan tambahan anggota d1, maka D1 memiliki anggota: b1 dan d1.
    - ... dan D2 adalah turunan B dengan tambahan anggota d2, maka D2 memiliki anggota: b1 dan d2.
      - Jika E adalah turunan dari D1 dan D2, maka E memiliki anggota: (dari D1) b1, d1, (dari D2) b1, dan d2 → DDD.

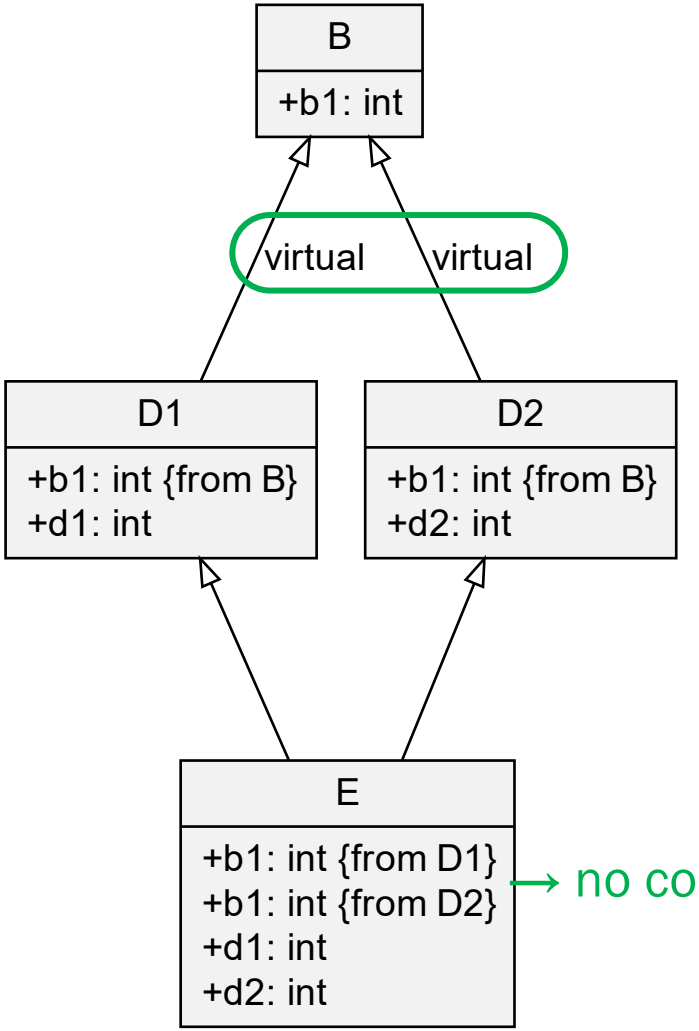


DDD

```
E e;
e.b1 = 1;      // error
e.B::b1 = 2;   // error
e.D1::b1 = 3; // ok
e.D2::b1 = 4; // ok
```

Kelas dasar virtual

- Kelas dasar virtual menghindari terjadinya DDD:
- Untuk setiap *base class* yang diturunkan secara virtual, kelas turunan level berikutnya hanya memiliki satu subobjek dari kelas tersebut.
- Pada contoh sebelumnya, D1 dan D2 harus diturunkan secara virtual dari B
- Sintaks: `class D1: virtual public B { ... };`
  atau `class D1: public virtual B { ... };`
  (sama saja)



→ no conflict

No DDD

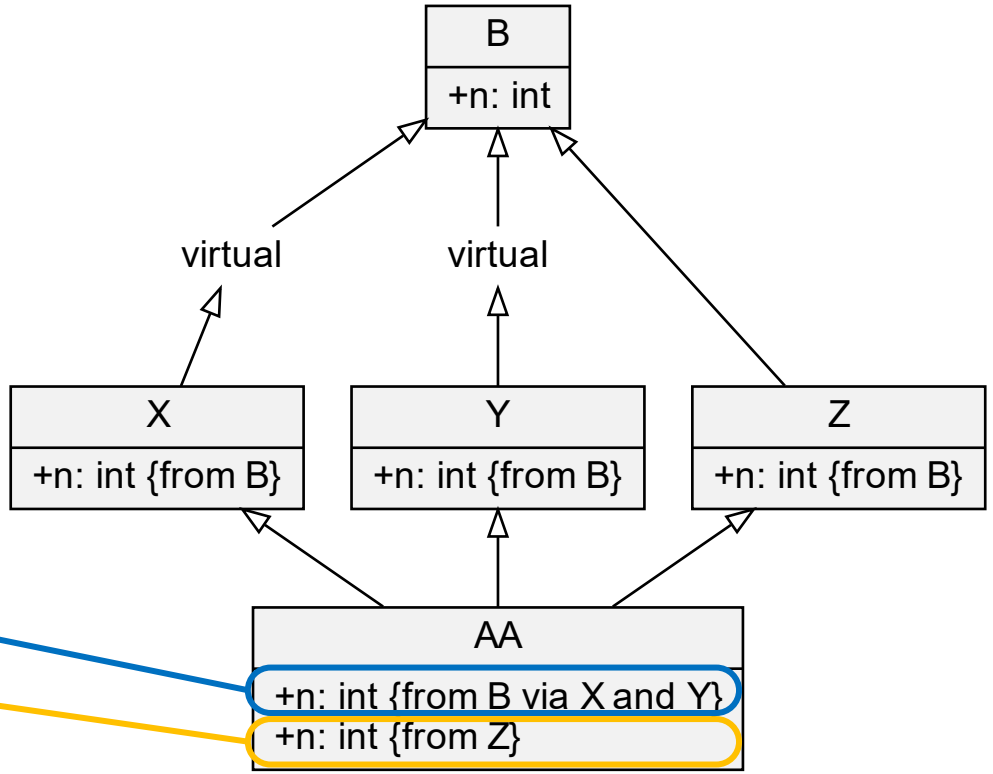
```
E2 e;
e.b1 = 1;      // ok
e.B::b1 = 2;   // ok
e.D1::b1 = 3; // ok
e.D2::b1 = 4; // ok
// all calls access the same b1
```

Contoh lain...

```
struct B { int n; };
class X: public virtual B {};
class Y: virtual public B {};
class Z: public B {};

struct AA: X, Y, Z {
  // every object of type AA has one X, one Y, one Z, and two B's:
  // one that is the base of Z and one that is shared by X and Y
    AA() {
        X::n = 1; // modifies the virtual B subobject's member
        Y::n = 2; // modifies the same virtual B subobject's member
        Z::n = 3; // modifies the non-virtual B subobject's member



        std::cout << X::n << Y::n << Z::n << '\n'; // prints 223
    }
};
```

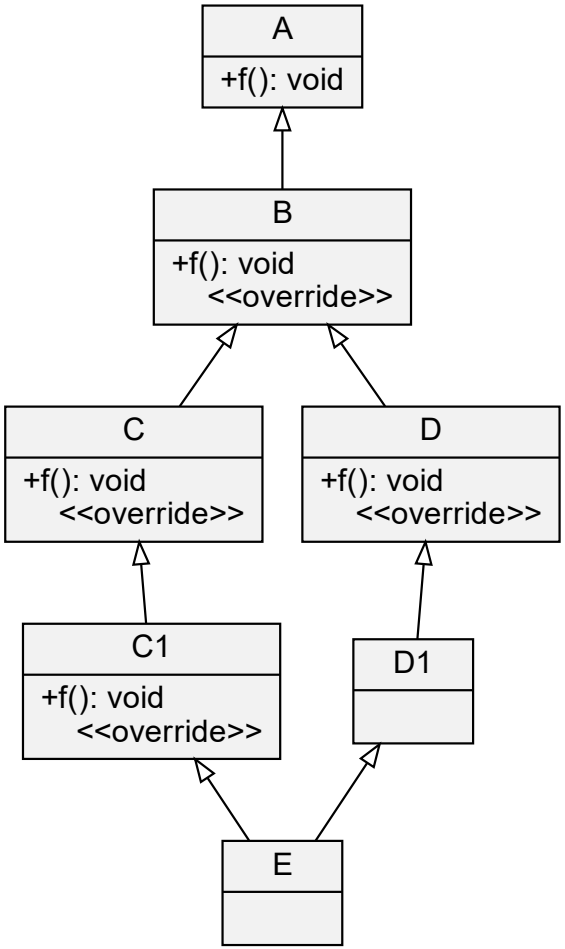
Contoh lain...

```
struct B { int n; };
class X: public virtual B {};
class Y: virtual public B {};
class Z: public B {};

struct AA: X, Y, Z {

    AA() {
        X::n = 1;
        Y::n = 2;
        Z::n = 3;

        std::cout << X::n << Y::n << Z::n << '\n';
    }
};
```

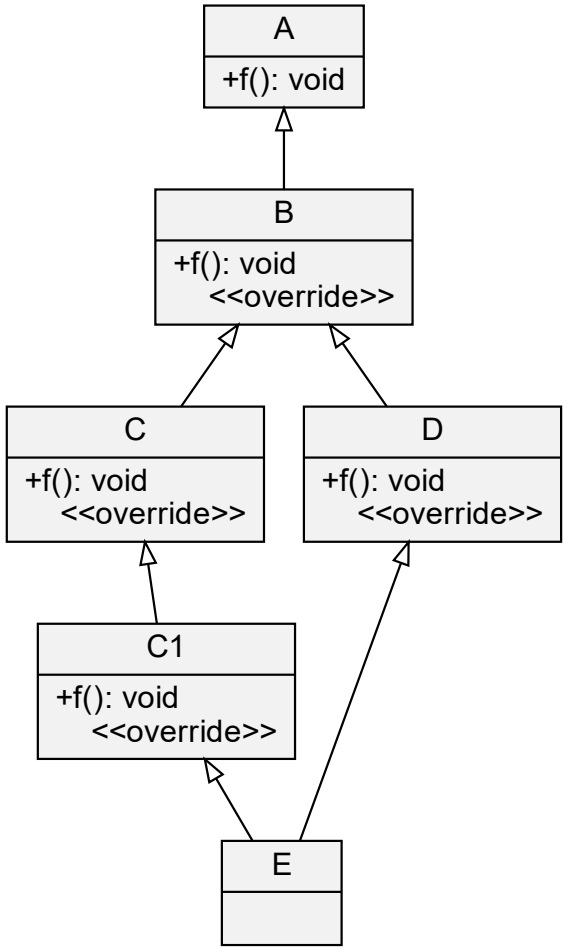


Latihan

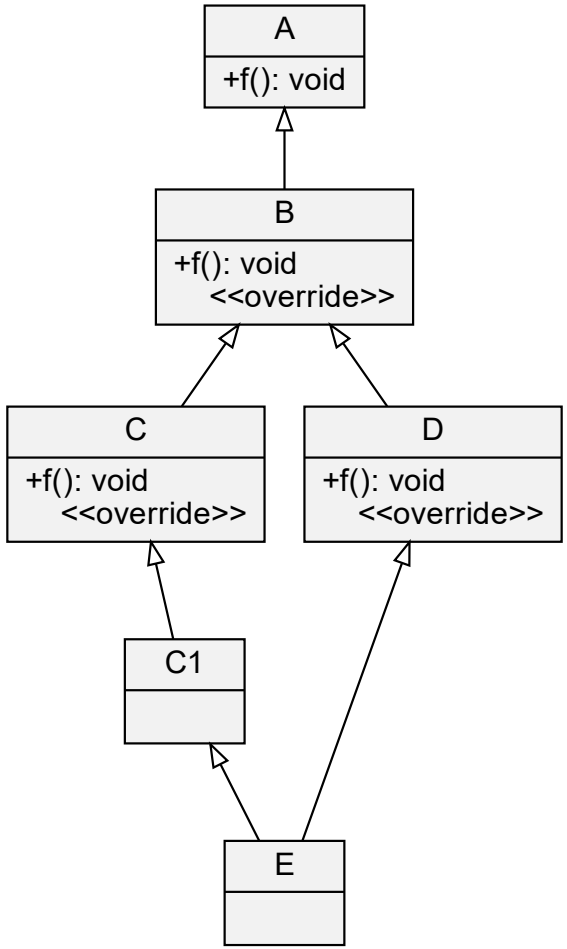
Method `f()` mana yang menjadi *final overrider* di kelas `E` pada hierarki-hierarki berikut? Apakah terjadi konflik?
[asumsi `A::f()` adalah virtual]



(a)



(b)



(c)

Sumber

- www.cppreference.com