Objektno orijentirano programiranje C++

Predavanje 07 - višestruko i virtualno nasljeđivanje

Matej Mihelčić

Prirodoslovno-matematički fakultet
Matematički odsjek

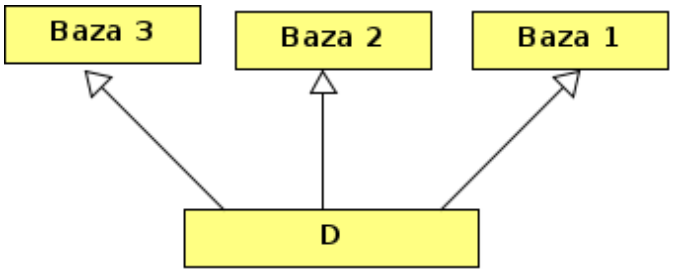
14. travnja 2024.

Višestruko nasljeđivanje

Klasa može proširivati više baznih klasa. U tom slučaju imamo višestruko nasljeđivanje.

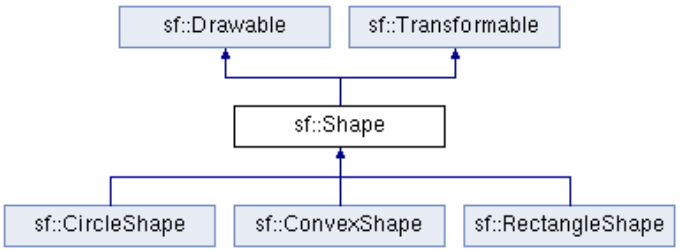
- Objekt izvedene klase sadrži podobjekte svih svojih baznih klasa. Ako klasa proširuje tri bazne klase, onda će instanca te klase sadržavati tri podobjekta baznih klasa. Ta činjenica prestaje vrijediti kada je proširivanje *virtualno*.
- Konstruktor proširene klase mora pozvati konstruktore svih svojih baznih klasa. Pri tome će defaultni konstruktori biti implicitno pozvani ukoliko nisu navedeni u inicijalizacijskoj listi. Poredak poziva baznih konstruktora jednak je poretku baznih klasa u derivacijskoj listi.
- Destruktori se pozivaju u obrnutom poretku od onog kojim su pozvani konstruktori.

Višestruko nasljeđivanje



Primjer poziva destruktora se može vidjeti na slici gore.

U biblioteci `SFML`, klasa `Shape` proširuje dvije bazne klase: apstraktnu bazu `Drawable` i implementacijsku bazu `Transformable`.



Ilustracija konstrukcije izvedenog objekta

```
1  class A1 {
2    public:
3      A1(){ std::cout << "konstruktor: A1"<<
4                                          std::endl;}
5      void f() {}
6  };
7
8  class A2 {
9     public:
10       A2(){ std::cout << "konstruktor: A2"<<
11                                        std::endl;}
12 };
```

Ilustracija konstrukcije izvedenog objekta

```
13 class B1 : public A1, public A2 {
14     public:
15     B1(){ std::cout << "konstruktor: B1"<<
16                                         std::endl;}
17     void g(){}
18 };
19
20 class B2 {
21         public:
22         B2(){ std::cout << "konstruktor: B2"<<
23                                         std::endl;}
24         void g() {}
25 };
```

Ilustracija konstrukcije izvedenog objekta

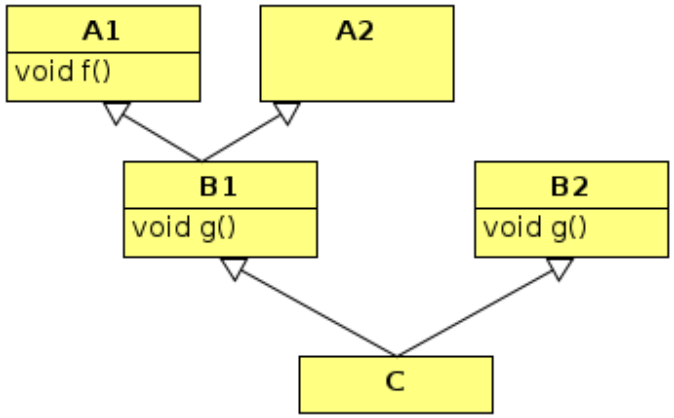
```
26 class C : public B1, public B2 {
27         public:
28         C(){ std::cout << "konstruktor: C"<<
29                                         std::endl;}
30 };
31
32 int main()
33 {
34         C c;
35         return 0; }
```

Program ispisuje:

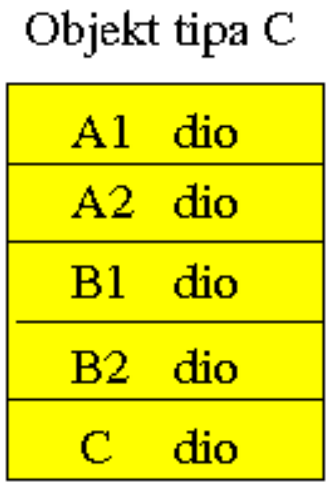
```
konstruktor: A1
konstruktor: A2
konstruktor: B1
konstruktor: B2
konstruktor: C
```

Primjer - nastavak

Stablo nasljeđivanja ima oblik:



Objekt tipa `C` ima formu:



Višestruko nasljeđivanje

- Kao i kod jednostrukog nasljeđivanja, ako proširena klasa definira konstruktor, njegova je dužnost da pozove konstruktore svih svojih baznih klasa. Isto vrijedi i za *konstruktor kopije*, te *operator pridruživanja* koji moraju brinuti o kopiranju svih podobjekata.
- Destruktor, kao i kod jednostrukog nasljeđivanja, ne poziva destruktor baznih klasa, o tome brine prevodilac.

Pravila za višestruko nasljeđivanje

- Uglavnom su ista kao i za jednostruko nasljeđivanje.
- Pokazivač ili referenca na instancu proširene klase može biti automatski konvertiran(a) u pokazivač ili referencu na svaku (javnu) baznu klasu. Višestruke konverzije (među kojima prevodilac ne pravi razlike) povećavaju mogućnost dvosmislenih funkcijskih poziva.

```
1 C c;
2 A1 * pa1 = &c;
3 B2 * pb2 = &c;
```

Pravila za višestruko nasljeđivanje

- Bazne klase su međusobno neovisne. Kada objekt koristimo kroz pokazivač/referencu na jednu od baznih klasa, onda nam metode iz drugih baznih klasa nisu dostupne.

```
1 c.f();      // o.k.
2 pa1->f();   // o.k
3 pb2->f();   // greska
```

Pravila za višestruko nasljeđivanje

- Kod višestrukog nasljeđivanja ime koje nije nađeno u proširenoj klasi traži se istovremeno u svim svojim baznim klasama. Ako je ime istovremeno nađeno u dva ili više bazna podstabla onda je poziv dvosmislen. Moguće je u takvoj situaciji precizno specificirati koja se funkcija zove pomoću operatora dosega. Za rješenje dvosmislenog poziva najbolje je implementirati funkciju u proširenoj klasi.

```
1 c.g();    // greska
2 c.B2::g(); // o.k.
```

Primjer

```
1  template <typename T>
2  class A{
3          public:
4          A(T x): p(new T(x)) {}
5          A(const A& a) : p(new T(a.get())) {
6                  std::cout << "A-CCtor: T = " <<
7                  typeid(T).name()<< ", val = " <<
8                  a.get() << std::endl; }
9          A& operator=(const A& a);
10         virtual ~A(){
11                 delete p;
12                 std::cout << "A-Dtor: T = " <<
13                   typeid(T).name()<< std::endl;}
```

Primjer

```
14              T get() const {return *p; }
15                    void set(T x) { *p = x; }
16                    private:
17                    T * p; };
18
19                    template <typename T>
20                    A<T>& A<T>::operator=(const A& a)
21                    {
22                    if( a.p == p ) return *this;
23
24                    if( p != 0) delete p;
25                    p = new T(*a.p);
26                    std::cout << "A-op=: T = " <<
27                    typeid(T).name()<< ", val = " <<
28                    a.get() << std::endl;
29                    return *this; }
```

Primjer

```
30 class C : public A<int>, public A<double>{
31  public:
32    C() : A<int>(0), A<double>(0){}
33    C(int x, double y) : A<int>(x), A<double>(y){}
34    // Poziv funkcijama get i set je dvosmislen.
35    // Stoga trebamo nove verzije tih metoda.
36    double get_d() { return A<double>::get(); }
37    int get_i() { return A<int>::get(); }
38
39    void set_d(double y) { A<double>::set(y); }
40    void set_i(int y) { A<int>::set(y); }};
```

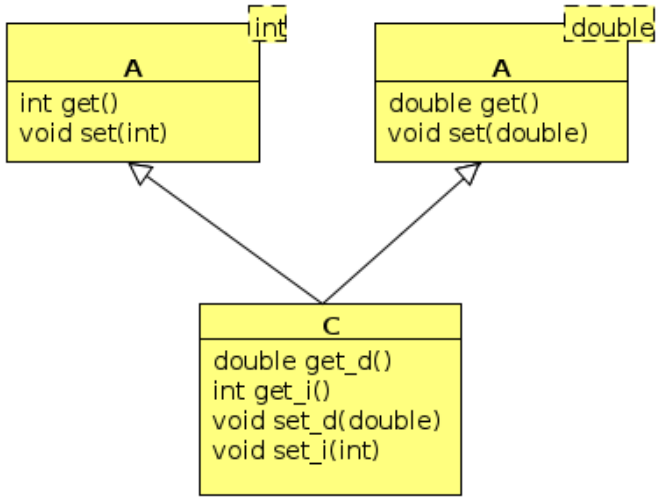
Primjer

```
41 int main() {
42     A<int> a(2);
43
44     C c; // defaultni konstruktor
45     c.set_i(12);
46     c.set_d(16.0);
47
48     A<int> b(c);    // o.k. b se "izrezuje" iz c
49     A<double> * p_a = &c; // o.k. c dohvacamo
50     //kroz A<double> sucelje
51     p_a->set(19.0);
52     cout  <<  p_a->get() << endl;  // o.k. Vidi
53     //samo funkciju get it A<double> klase
54     cout << c.get() << endl; // Greska!
55     //Dvosmislen poziv, get postoji u obje
56     //bazne klase
```

Primjer

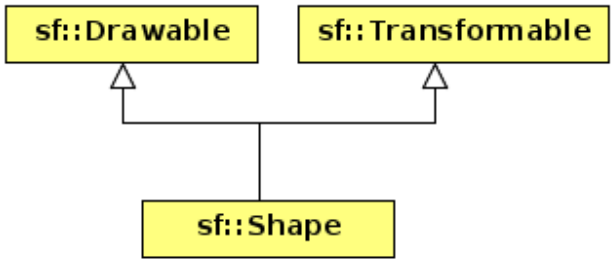
```
57      cout <<  c.A<double>::get() << endl; // o.k.
58      // Eksplicitno zadavanje bazne klase
59      // pri pozivu funkcije get.
60
61      C cc(c);  // Sintetizirani CCtor.
62      //Zove redom bazne CCtore
63
64      C d;
65      d = cc;   // Sintetizirani OP.
66      //Zove redom bazne OP
67      return 0;
68                }
```

Primjer

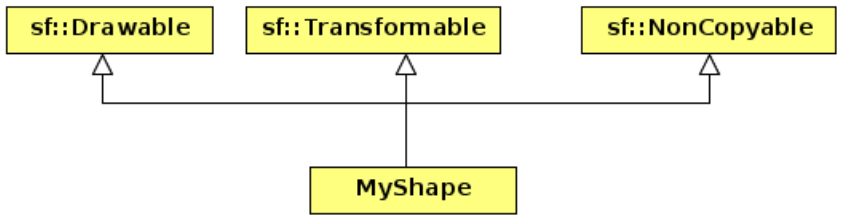

int
A
int get()
void set(int)
double
A
double get()
void set(double)
C
double get_d()
int get_i()
void set_d(double)
void set_i(int)

Kada koristimo višestruko nasljeđivanje?

Višestruko nasljeđivanje prirodno koristimo kada imamo objekt koji implementira više različitih sučelja.



Neke baze mijenjaju ponašanje izvedene klase, bez mijenjanja same izvedene klase:

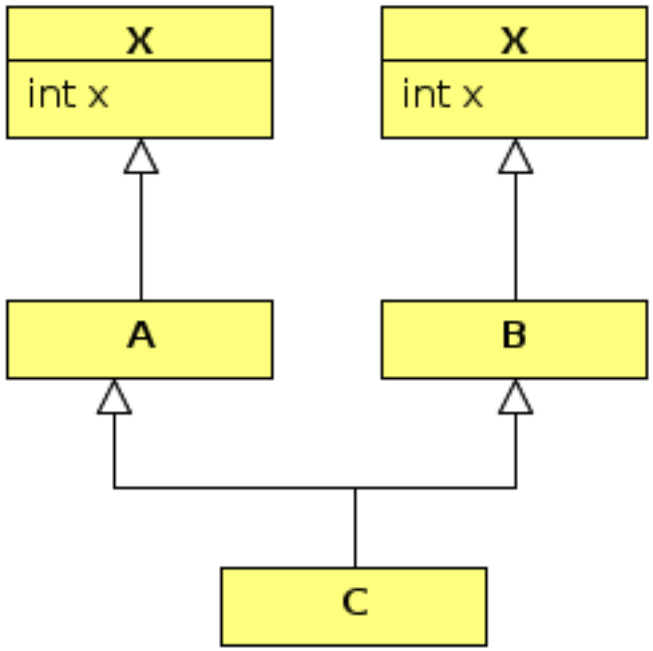


Obično (nevirtualno) nasljeđivanje

Kada nasljeđivanje nije *virtualno* (ključna riječ `virtual` nije navedena u derivacijskoj listi) onda u instanci klase svakoj baznoj klasi odgovara jedan podobjekt tipa bazne klase.

```
1  struct X{
2           int x;
3  };
4  struct A : X {
5           A(int x_) { x= x_; }
6  };
7  struct B : X {
8           B(int x_) { x= 2*x_; }
9  };
10 struct C : A, B {
11          C(int x_, int y_) : A(x_), B(y_)  {}
12 };
```

Obično (nevirtualno) nasljeđivanje




```
1 C c(1,2);
2 c.x = 3; // greska
```

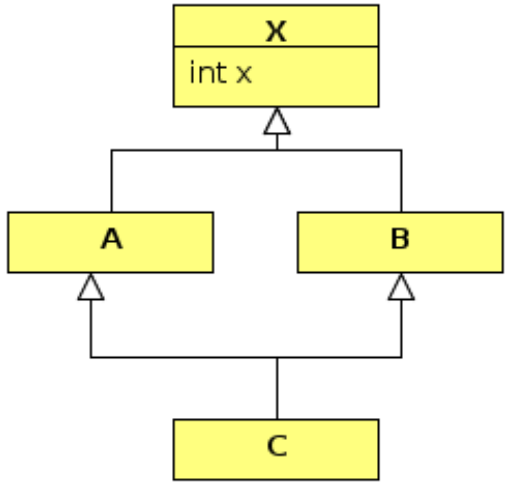
Klasa koja sadrži dva podobjekta istog tipa ima reduciranu funkcionalnost budući da se iz dosega izvedene klase ne može pozvati funkcija iz bazne klase čiji je objekt više puta uključen; Svaki takav poziv je dvosmislen.

Virtualno nasljeđivanje

Kada je u derivacijskoj listi bazna klasa X deklarirana `virtual` (virtualna baza) onda će svaka klasa izvedena iz nje sadržavati samo jedan podobjekt tipa X bez obzira koliko se puta u stablu nasljeđivanja tip X javlja kao virtualna baza. Prethodni primjer s virtualnim nasljeđivanjem daje:

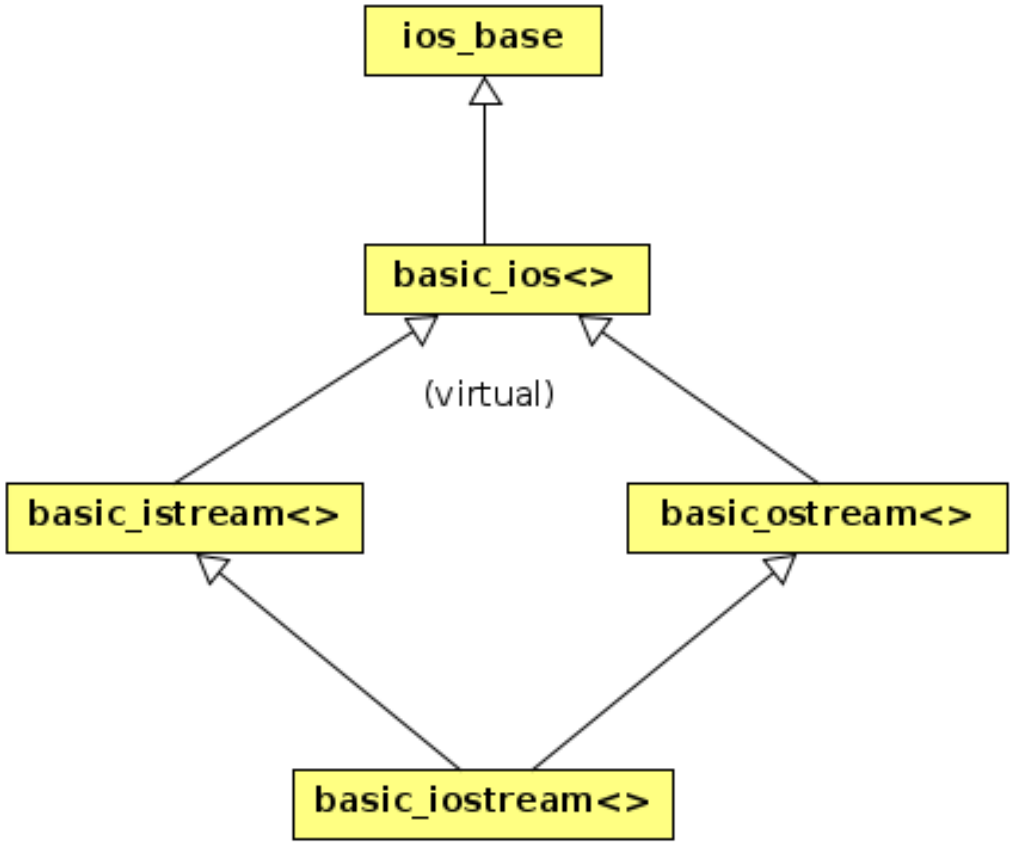
```
1 struct X{
2          int x; };
3 struct A : virtual X {
4          A(int x_) { x= x_; } };
5 struct B : virtual X {
6          B(int x_) { x= 2*x_; } };
7 struct C : A, B {
8          C(int x_, int y_) : A(x_), B(y_)  {} };
```

Virtualno nasljeđivanje




```
1  int main(){
2          C c(1,2);
3          c.x = 3;} // o.k. samo je jedan x
```

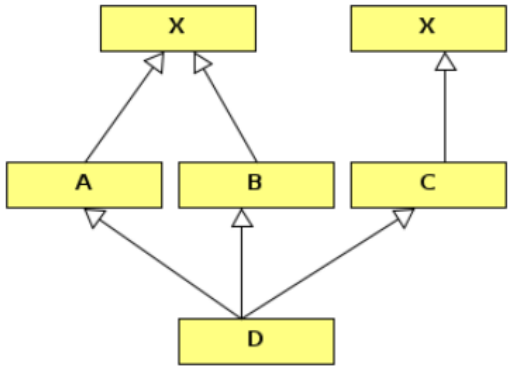
Primjer iz standardne biblioteke



Ista klasa može biti virtualna i nevirtualna baza

Svako virtualno pojavljivanje klase X u stablu nasljeđivanja odgovara jednom podobjektu, dok svako drugo, nevirtualno, pojavljivanje iste klase X rezultira zasebnim podobjektom tipa X.

```
1 class X { /* ... */ };
2 class A : public virtual X {/*...*/};
3 class B : public virtual X {/*...*/};
4 class C : public  X {/*...*/};
5 class D : public A, public B, public C {/*...*/};
```



Pravila o virtualnom nasljeđivanju

- Klasa koja ima virtualnu baznu klasu deklarira da je spremna dijeliti svoju virtualnu bazu u stablu nasljeđivanja. Svaki objekt nasljeđuje točno jedan podobjekt dane virtualne bazne klase bez obzira na to koliko se puta bazna klasa javlja kao *virtualna baza* u stablu nasljeđivanja.
- Objekt izvedene klase može se koristiti kroz referencu ili pokazivač tipa virtualne bazne klase na isti način kao što se koristi kroz referencu/pokazivač tipa regularne bazne klase.
- Svaki član virtualne baze može se dohvatiti bez dvosmislenosti poziva budući da postoji samo jedan podobjekt virtualne baze.

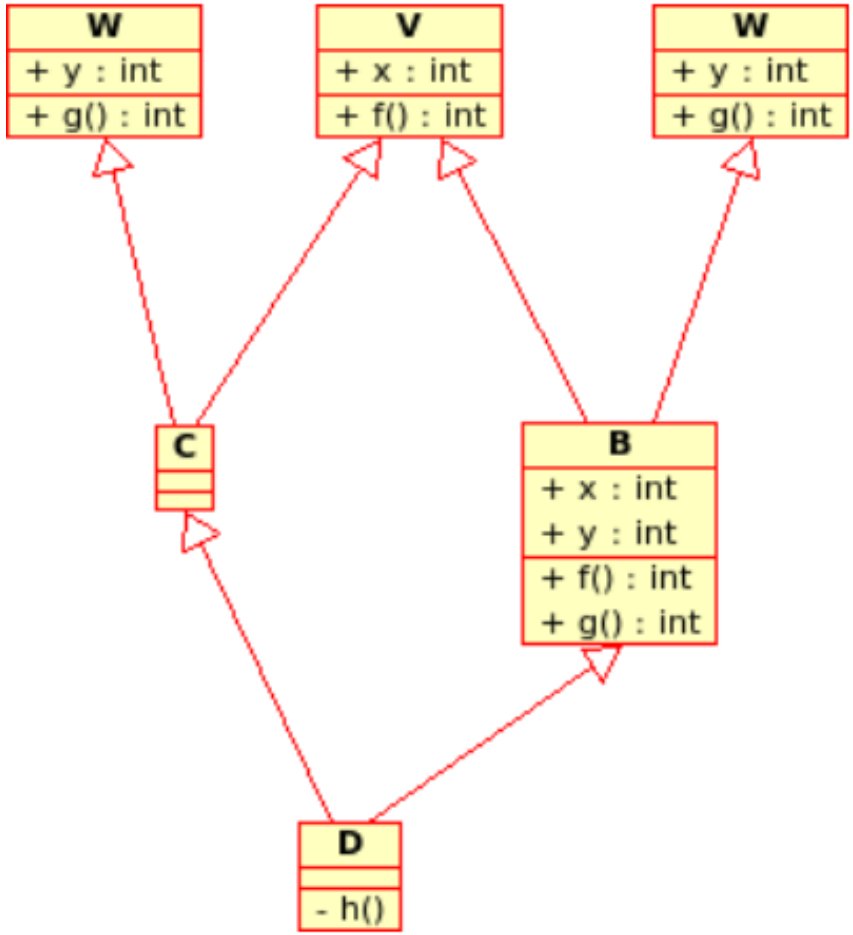
Pravila o virtualnom nasljeđivanju

- Kod virtualnog nasljeđivanja nastaje situacija u kojoj do članice virtualne bazne klase ima više staza kroz stablo nasljeđivanja. Pri tome su moguće tri situacije:
  - Ako svaka staza vodi do istog člana u virtualnoj bazi onda je poziv korektan.
  - Ako jedna staza vodi do člana u virtualnoj bazi, a druga do člana prerađenog u nekoj od klasa izvedenih iz virtualne baze, onda je poziv korektan i poziva se prerađena metoda koja *dominira*.
  - Ako kroz različite staze dolazimo do različitih prerađenih članova u klasama koje proširuju virtualnu bazu, onda je poziv *dvosmislen*.

Pravila o virtualnom nasljeđivanju

```
1  class V { public: virtual int f(){ return 2; }
2                    int x; };
3  class W { public: int g(); int y; };
4  class B : public virtual V, public W
5  {
6          public:
7          int f(){ return 5; }  int x;
8          int g();  int y;
9  };
10 class C : public virtual V, public W { };
11 class D : public B, public C { void h(); };
```

Pravila o virtualnom nasljeđivanju



Pravila o virtualnom nasljeđivanju

```
1  void D::h()
2    {
3        x++;   // OK: B::x skriva V::x
4        f();   // OK: B::f() skriva V::f()
5        y++;   // greska: B::y ili C::W::y ?
6        g();   // greska: B::g() ili C::W::g() ?
7           }
8
9  int main()
10    {
11       D d;
12       V * pv = &d;
13       std::cout << pv->f() << std::endl;
14       // 2 ili 5 ?
15       return 0;
16         }
```

Konstrukcija klase s virtualnom bazom

Kod virtualnih baza se javlja pitanje, kako provesti konstrukciju virtualne bazne klase. Kod regularnih baza je dužnost izvedene klase da pozove konstruktore svih svojih baznih klasa, međutim to nije primijenjivo na virtualne baze jer bismo tada dobili više podobjekata koji odgovaraju višestrukom pojavljivanju virtualne baze u stablu nasljeđivanja.

Problem konstrukcije virtualne baze rješava se sljedećim pravilom: **konstruktor virtualne baze u svojoj inicijalizacijskoj listi poziva konstruktor najizvedenije klase u stablu nasljeđivanja**.

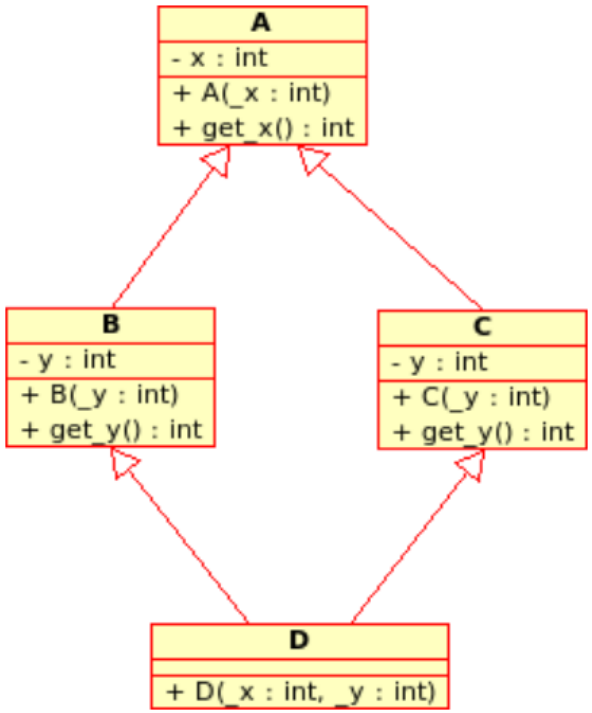
```
1  class A{
2    int x;
3    public:
4    A(int _x) : x(_x){cout << "A("<<_x<<")"<<endl;}
5    int get_x(){ return x; } };
```

Konstrukcija klase s virtualnom bazom

```
1  class B : public virtual A{
2      int y;
3      public:
4      B(int _y) : A(10 * _y), y(_y) {
5                      cout << "B("<<_y<<")"<<endl;}
6      int get_y(){ return y; } };
7
8      class C : public virtual A{
9            int y;
10           public:
11           C(int _y) : A(_y*_y), y(_y) {
12                         cout << "C("<<_y<<")"<<endl;}
13           int get_y(){ return y; } };
```

Konstrukcija klase s virtualnom bazom

```
1  class D : public B, public C{
2          public:
3          D(int _x, int _y) : A(_x), B(_y), C(_y) {
4      cout << "D("<<_x<<","<<_y<<")"<<endl;}
5  };
```



Konstrukcija klase s virtualnom bazom

- Klasa `D`, kao najizvedenija, poziva konstruktor svoje virtualne baze.
- Klase `B` i `C` također moraju u svojim konstruktorima inicijalizirati `A` kao svoju direktnu bazu, ali prevodilac te inicijalizacije ignorira.
- Ukoliko najizvedenija klasa ne pozove konstruktor virtualne baze prevodilac će ubaciti poziv defaultnom konstruktoru, ako takvog nema javlja se greška pri prevođenju.

Primjer:

```
1  int main(){
2          D d(2,6);
3          cout << d.get_x() << endl;
4          cout << d.C::get_y() << endl;
5
6          return 0; }
```

Ispis primjera:

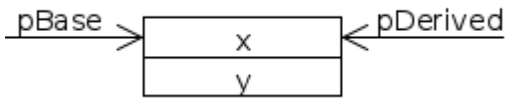
```
A(2)
B(6)
C(6)
D(2,6)
2
6
```

Prvo se konstruira virtualna baza, zatim ostali dijelovi objekta prema redosljedu pojavljivanja u derivacijskoj listi.
Općenito pravilo: virtualne baze se konstruiraju prije nevirtualnih baza, bez obzira na mjesto pojavljivanja u stablu nasljeđivanja.

O implementaciji višestrukog i virtualnog nasljeđivanja

```
1  class Base{
2          int x; };
3
4  class Derived : public Base{
5          int y; }
6
7  Derived * pDerived = new Derived();
8  Base *    pBase = pDerived;
```

Objekt klase `Derived` u memoriji ima oblik:



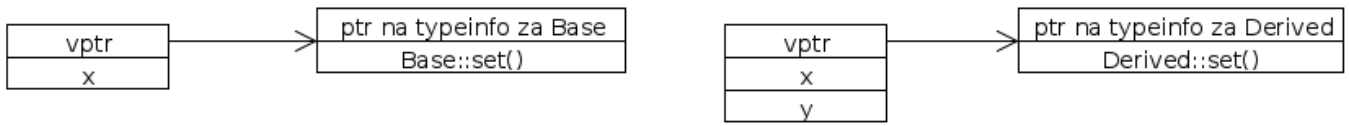
Pokazivači `pBase` i `pDerived` pokazuju na istu memorijsku lokaciju.

O implementaciji višestrukog i virtualnog nasljeđivanja

Ako su klase polimorfne onda se instanca klase povećava za pokazivač na tabelu virtualnih funkcija.

```
1  class Base{
2          public:
3          virtual void set(int x_){ x = x_; }
4          protected:
5          int x; };
6
7  class Derived : public Base{
8          public:
9          void set(int x_) { x= x_*x_; }
10         int y; };
```

Objekti u memoriji imaju oblik:



O implementaciji višestrukog i virtualnog nasljeđivanja

Svaki objekt sada ima pokazivač na virtualnu tabelu. Novi element u virtualnoj tablici, koji do sada nismo spominjali je pokazivač na `typeinfo` objekt. To je još jedna tablica pridružena klasi koja sadrži informacije o klasi za RTTI (run-time type information) sustav. Pokazivač na nju se smješta u tablicu virtualnih funkcija.

Višestruko nasljeđivanje je složenije.

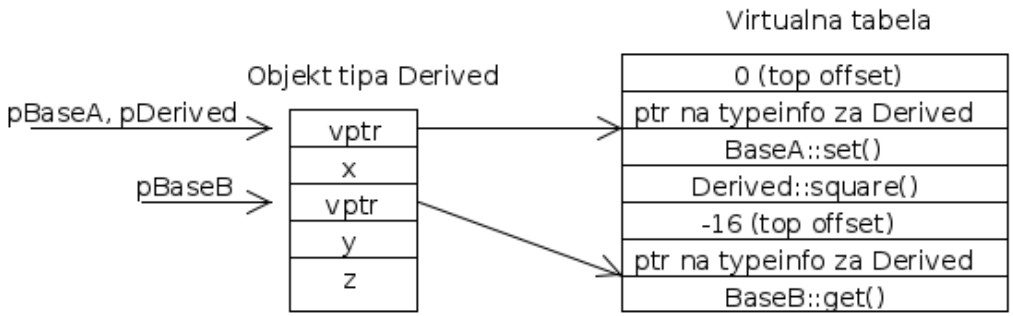
```
1  class BaseA{
2          public:
3          virtual void set(int x_){ x = x_; }
4          int x; };
5
6  class BaseB{
7          public:
8          virtual int get(){ return y; }
9          int y; };
```

O implementaciji višestrukog i virtualnog nasljeđivanja

```
10 class Derived : public BaseA, public BaseB{
11         public:
12         int z;
13         virtual void square(){ z = z*z; } };
```

```
1 Derived * pDerived = new Derived();
2 BaseA *     pBaseA = pDerived;
3 BaseB *     pBaseB = pDerived;
```

Objekt klase `Derived` u memoriji i pripadna tablica virtualnih funkcija imaju oblik:



O implementaciji višestrukog i virtualnog nasljeđivanja

Kao i do sada pokazivači `pDerived` i `pBaseA` drže istu adresu i pokazuju na početak objekta. S druge strane, pokazivač `pBaseB` (BaseB je desna baza u derivacijskoj listi) ima različitu adresu od one u `pDerived` - on pokazuje na podobjekt `BaseB` u objektu tipa `Derived`.

- Kao posljedicu imamo da kod višestrukog nasljeđivanja objekt ima *po jedan pokazivač na virtualnu tablicu za svaku svoju bazu*. Broj virtualnih tablica je jednak broju baza, dok se kod nekih implementacija one spoje u jednu tablicu (kao u prikazu).

O implementaciji višestrukog i virtualnog nasljeđivanja

- Druga posebnost višestrukog nasljeđivanja je potreba za korekcijom `this` pokazivača pri pozivu virtualnih funkcija kroz pokazivač na desnu baznu klasu. Iz pokazivača na podobjekt desne bazne klase potrebno je dobiti pokazivač na početak cijelog objekta. Tu korekciju ne može napraviti prevodilac za vrijeme kompilacije jer je dinamički tip pokazivača poznat tek za vrijeme izvršavanja programa. Stoga virtualna tablica može pored pokazivača na virtualne funkcije sadržavati i pomake (`offsets`) potrebne da se dođe do početka virtualne tablice (odnosno virtualne tablice pridružene lijevoj bazi).
- Situacija postaje puno složenija kod `virtualnog nasljeđivanja`. Prije svega, podobjekt virtualne bazne klase dolazi nakon svih ostalih dijelova objekta. To znači da se njegova pozicija u klasama nastalim nasljeđivanjem klase s virtualnom bazom mijenja, pa mu se mora pristupati indirektno.

O implementaciji višestrukog i virtualnog nasljeđivanja

- Virtualnom baznom dijelu može se pristupiti putem pokazivača na bilo koju njenu podklasu te stoga na osnovi tog pokazivača nije jasno gdje se virtualna baza nalazi - ta se informacija mora pročitati iz virtualne tablice. Nadalje, postupak konstrukcije i destrukcije objekta s virtualnom bazom postaje složeniji jer ne smije svaki konstruktor podklase konstruirati svoju virtualnu bazu. To mora napraviti samo jedan od njih, te isto tako, pri destrukciji samo jedan destruktor uništava virtualnu bazu. Zbog svih tih komplikacija prevodioc generira više različitih virtualnih tablica koje mu omogućavaju da izvrši svoju zadaću.