Programozás alapjai II.
(6. ea) C++

mutatókonverziók, heterogén kollekció

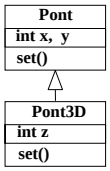
Szeberényi Imre
BME IIT
<szebi@iit.bme.hu>

Öröklés (ismétlés)

- Egy osztályból olyan újabb osztályokat származtatunk, amelyek rendelkeznek az eredeti osztályban már definiált tulajdonságokkal és viselkedéssel.
- Analitikus – Korlátozó
- A tagfüggvények átdefiniálhatók (overload)
- virtuális függvény: hogy a tagfüggvény alaposztály felől is elérhető legyen

Analitikus öröklés példa (ism.)

```
class  Pont {
  int x, y;
public:
  Pont(int x1, int y1) :x(x1), y(y1) {}
  void set(int x1, int y1) {x = x1; y = y1;}
};
class Pont3D :public Pont {
  int z;
public:
  Pont3D(int x1, int y1, int z1)
                          :Pont(x1, y1), z(y1) {}
  void set(int x1, int y1, int z1) {
                        Pont::set(x1, y2); z = z1; }
};
```



Bővült

Korlátozó öröklés példa/1 (ism.)

Szeretnénk egy stack és egy queue osztályt:

- mindkettő tároló
- nagyon hasonlítanak, de
- eltér az interfészük:
  - o put, get <> push, pop
- önállóak vagy örökléssel ?

```
class Queue {  ....
public:
   void   put( int e );
   int      get( );
};
```

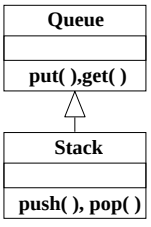
```
class Stack {  ....
public:
    void   push( int e );
    int      pop( );
};
```


Korlátozó öröklés példa/2 (ism.)

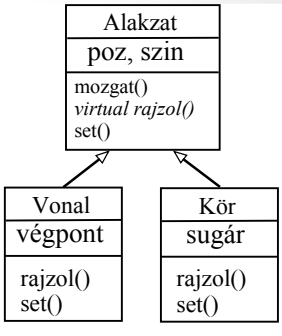
```
class Stack : private Queue { // privát: eltakar a külvilág felé
  int nelem;
public:
  Stack( ) : nelem(0) { }
  int pop( ) { nelem--; return(get()); }
  void push(int e) {
        put(e); // betesz
        for( int i = 0; i < nelem; i++ )
              put(get( )); // átforgat
        nelem++;
  }
};

Stack s1; s1.pop();  s1.get()
```

Továbbhívjuk a Get()-et

Nem hatékony, csak példa!



Virtuális tagfüggvény (ism.)



rajzol() "átdefiniálása" virtuális függvénnyel

késői kötés

set() átdefiniálása: fv overload

Alakzat::rajzol();
Mit hív ?

Fontos C++ sajátságok

- Konstruktor nem lehet virtuális
- Destruktor igen, és érdemes odafigyelni rá
  - Alaposztályból dinamikus adattagot tartalmazó osztályt hozunk létre, majd ezt az alaposztály "felöl" használjuk (töröljük).
- A konstruktorból hívott (saját) virtuális függvény még nincs átdefiniálva! A virt. táblát maga konstruktor tölti ki! (kötés)
  - absztrakt osztály estén NULL pointer!

Inicializálás /1 (ism.)

```
class  Pont {
protected:
  int x, y;
public:
  Pont(int x1 = 0, int y1 = 0) { x = x1; y = y1;}
};
class Pont3D :public Pont {
  int z;
public:
  Pont3D(int x1, int y1, int z1)
                        {x = x1; y = y1; z = z1;}
};
```

Legyen default, mert ....

Mindig lehet így?

Alaposztály konstruktora mikor hívódik?

Inicializálás /2 (ism.)

```
class  Pont {
protected:
  int x, y;
public:
  Pont(int x1, int y1)  : x(x1), y(y1) { }
};
class Pont3D :public Pont {
  int z;
public:
  Pont3D(int x1, int y1, int z1) : Pont(x1, y1),  z(z1)
                                   { }
};
```

Nem fontos a default ....

Alaposztály konstruktora

Inicializálás /3 (ism.)

```
class  FixKor :public Pont {
  double& r;
  const double PI;
public:
  Kor(int x, int y, double& r;) :x(x), y(y), r(r), PI(3.14) { }
};
```

Kötelező inicializálni!

Melyik y? Van már this?

```
class  FixKor :public Pont {
  double& r;
  static const double PI;
...
};
const double Kor::PI = 3.14;    // statikus tag, létre kell hozni
```


Öröklés és polimorfizmus

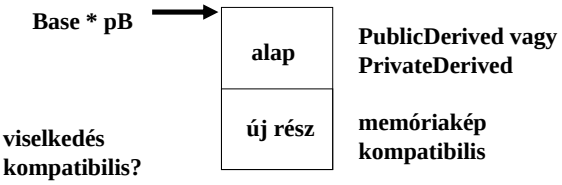
```
struct A {
  void valami() { cout << "A valami" << endl; }
  void semmi() { cout << "A semmi" << endl; }
};
struct B: public  A{
  void valami() { cout << "B valami" << endl; }
  void valami(int) { cout << "B valami int" << endl; }
};
...
  B b;
  b.valami();          // B valami
  b.valami(1);         // B valami(int)
  b.semmi();           // A semmi
  b.A::valami();       // A valami
  b.A::valami(int)     // HIBA
```


Mutatókonverzió öröklésкor

- Mutatókonverzió = rejtett objektumkonverzió
- Kompatibilitás: öröklés
  - kompatibilis memóriakép
  - kompatibilis viselkedés (tagfüggvények)

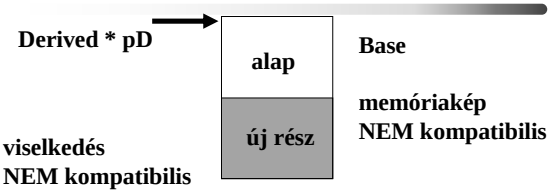
class Base { .... };
class PublicDerived : public Base { .... };
class PrivateDerived: private Base { .... };

Konverzió alaposztályra




```
PublicDerived pubD; // pubD kaphatja a Base üzeneteit
Base * pB = &pubD; // nem kell explicit típuskonverzió

PrivateDerived priD;// priD nem érti a Base üzeneteit
pB = (Base *)&priD;// mégis érti explicit konverzió!
```


Konverzió származtatott osztályra




```
Base base;
Derived *pD = (Derived *) &base;
        // nem létező adatmezőket és üzeneteket el lehet érni
        // veszély: explicit konverzió
```


Mutatókonverzió és virtuális fv.

```
class Alakzat { ... virtual void rajzol() = 0; };
class Vonal : public Alakzat { ... };
class Kor : public Alakzat { ... };
class Teglalap : public Alakzat { ... };
Alakzat* tar[100];

tar[0] = new Vonal(....); // konverzió
tar[1] = new Kor(...); // konverzió
....
```

```
for (int i = 0; i < 100; i++ )
      tar[i] ->rajzol();
```

Származtatott osztály függvénye

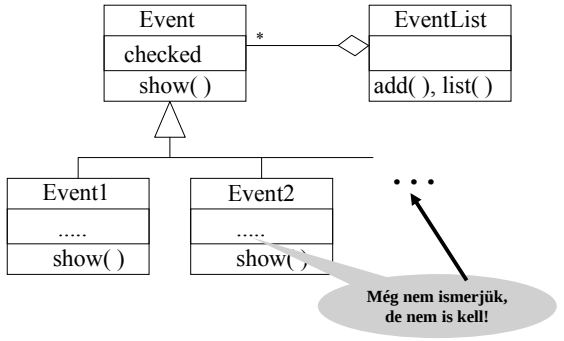
Heterogén gyűjtemények

- Különböző típusú objektumokat egy közös gyűjteménybe tesszük
- Egységes kezelés: valamilyen viselkedési kompatibilitás
  - egy öröklési hierarchiából definiált objektumokat tehetünk heterogén szerkezetbe
  - kompatibilitásból származó előnyök (pl. alaposztály pointere) kihasználása

Heterogén kollekció példa

- Egy rendszer eseményeit kell naplózni.
- Az események egymástól eltérő adattartamúak, és esetleg új események is lesznek, amit még nem ismerünk.
- Események sorrendje fontos, ezért célszerűen egy tárolóban kell lenniük.
- Az eseménynapló megnézésekor meg kell mutatni azt is, hogy mely eseményeket néztük meg már korábban.

Eseménykezelő megvalósítása



Eseménykezelő megvalósítása/2

```
class Event {
  bool  checked;
public:
  Event ( ) :checked(false) {}
  virtual  void show( ) { cout<<" Checked: ";
                          cout<<checked<<endl; checked = true; }
};

class Event1 :public Event {
  ...
public:
   Event1();
   void show ( ) { cout << ..... ;
           Event::show();
  }
};
```

Hurok ?

Eseménykezelő megvalósítása/3

```
class EventList {
  int           nevent;
  Event         *events[100];
public:
  EventList( ) { nevent = 0; }
  void add(Event& e) { events[nevent++] = &e; }
  void add(Event *e) { events[nevent++] = e; }
  void list( ) {
      for (int i = 0; i < nevent; i++)
              events[i]->show();
  }
};
```

Alaposztály mutatója

Nem célszerű mind a két fajta add()!

Származtatott osztály függvénye

Eseménykezelő használata

```
class Event {...};
class Event1 :public Event {...};
class Event2 :public Event {...};
class EventList {...};
...
EventList list;
        list.add(new Event1(....));
        list.add(new Event2(....));
...
...     list.add(new Event9(....));

        Event1 e;
        list.add(e);
        list.list();
```

Új esemény: csupán definiálni kell az új osztályt

Ez másik add()! (ref)

Heterogén kollekció összefoglalás

- Különböző típusú objektumokat egy közös gyűjteménybe tesszük.
- Kihasználjuk az öröklésből adódó kompatibilitást.
- Nagyon gyakran alkalmazzuk
  - könnyen bővíthető, módosítható, karbantartható

Tipikus hiba

```
class EventList {
  int           nevent;
  Event         events[100];
public:
  EventList( ) { nevent = 0; }
  void add(Event& e) { events[nevent++] = e; }
  void add(Event* e) { events[nevent++] = *e; }
  void list( ) {
      for (int i = 0; i < nevent; i++)
              events[i].show();
  }
};
```

Nem pointert tárol!

Event::show()

Adavesztés!!
A származtatott rész elveszik!

Ki szabadít fel?

```
class EventList {
  int           nevent;
  Event         *events[100];
public:
  ....
  void add(Event* e) { events[nevent++] = e; }
  void list( ) {
      for (int i = 0; i < nevent; i++)  events[i]->show();
  }
  ~EventList() {
      for (int i = 0; i < nevent; i++)
              delete events[i];  // events[i]->~Event();
  }
}; ....
```

list.add(new Event1(....));

Virtuális kell!

Virtuális destruktor újból

```
class Event {
  ...
public:
  virtual  void show( ) {} ;
  virtual ~Event() {} ;
};
```

2

```
class Event1 :public Event {
   int *p;
public:
   Event1(int s) { p = new int[s]; }
   ....
   ~Event1() { delete[] p; }
};
```

```
Event *ep = new Event1(120);
delete ep;
```

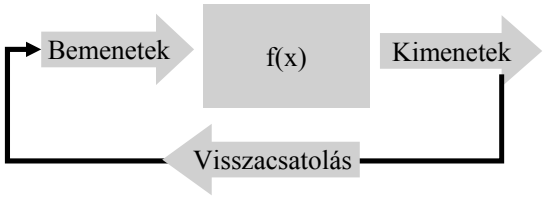
1

Más, mint a többi virt. fv.

Digitális áramkör modellezése

- Digitális jel: üzenet
- Áramköri elemek: objektumok
  - bemenet, kimenet, viselkedés (f(x))
  - kapcsoló, kapu, drót, forrás, csomópont
- Objektumok a valós jelterjedésnek megfelelően egymáshoz kapcsolódnak. (üzennek egymásnak)
- Visszacsatolás megengedett.

Modell

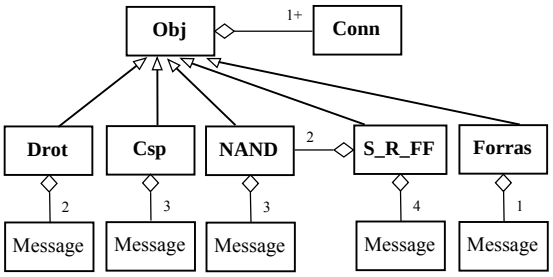


A változásokat üzenetek továbbítják. Ha nincs változás, nem küldünk újabb üzenetet.
Csak véges számú iterációt engedünk meg.

Áramköri elemek felelőssége

- Kapcsolatok (bemenet/kimenet) kialakítása, nyilvántartása.
- Bejövő üzenetek tárolása összehasonlítás céljából.
- Válaszüzenetek előállítása és továbbítása a bejövő üzeneteknek és a működésnek megfelelően.

Osztályhierarchia



Obj: alaposztály

- A különböző objektumok ősosztálya.
  - minden áramköri elem ebből származik
- Lehetővé teszi ill. támogatja:
  - az objektumok közötti kapcsolatok kialakítását, (a *Conn* osztályból felépített dinamikus tömbben tároljuk a kapcsolatokat)
  - kapcsolatokon keresztül az üzenetek (*Message* objektum) továbbítását,
  - a működést (viselkedést) megvalósító függvény elérését (a Set virtuális függvényen keresztül).

Obj: absztrakt alaposztály

```
class Obj {
   char nev[10];                  // objektum neve
   Pin        nk;                 // kapcsolódási pontok száma
   Conn    *konn;                 // kapcsolatok leírása
   Obj(const Obj&)  {}            // hogy ne lehessen használni
   Obj& operator=(const Obj&) {} // hogy ne lehessen haszn.
public:
   Obj(char *n, Pin k) {setNev(n); konn = new Conn[nk = k]; }
   ~Obj() { delete[] konn; }                    // tömb felszab.
   void setNev(char *n) { strcpy(nev, n); }     // név beállítása
   void setConn(Pin k, Obj& o, Pin on);         // összekapcs.
   void uzen(Pin k, Message& msg);              // üzen
   virtual void set(Pin n, Message& msg) = 0;  //működtet
};
```


Conn: kapcsolatok tárolása

- Egy objektumkapcsolatot leíró osztály
- Példányaiból felépített dinamikus tömb *(Obj::konn)* írja le egy objektum összes kapcsolatát

Miért nem referencia ?

```
class Conn {
   Obj    *obj;          // ezen objektumhoz kapcsolódik
   Pin     n;            // erre a pontra
public:
   Conn() :obj(NULL) {}
   void setConn(Pin k, Obj& o)  { n = k; obj = &o; }  // beállít
   Obj *getConn(Pin& k)  { k = n; return(obj); }      // lekérdez
};
```


Message: jel mint üzenet

- Digitális jelet reprezentáló osztály
  - undef, jel 0 és jel 1 értéke van.
- A végtelen iteráció elkerülése végett a jelszint mellet egy iterációs számláló is van.
- Megvalósítása struktúrával, mivel az adattakarás csak nehezítene.
- Műveletei:
  msg1 == msg2
  msg1 != msg2
  --msg

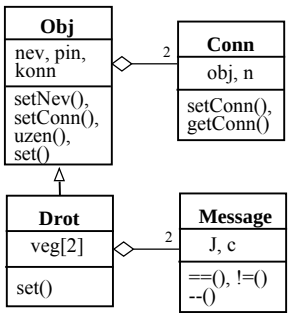
Message: jel mint üzenet /2

```
struct Message {
   enum msgt { undef, jel} typ;   // típus
   bool J;                        // jelszint 0 v. 1
   int  c;                        // iterációs számláló
   // default konstruktor is
   Message(msgt t = undef, bool j = false, int n = 20)
                :typ(t), J(j), c(n) {}
   // két üzenet egyenlő, ha az típusuk és jelszintjük is azonos
   bool operator==(Message& m) {
                return(typ == m.typ && J == m.J); }
   bool operator!=(Message& m) { return(!operator==(m)); }
   Message& operator--() {
         if (--c <= 0)   // iterációs számláló csökkentése
               throw "Sok Iteracio!"; return(*this); }
};
```


Üzenet továbbítása

```
/**
 * Üzenet (msg) küldése a k. pontra kapcsolódó obj. felé
 */
void Obj::uzen(Pin k, Message& msg) {
   Pin n;          // kapcsolodó objektum kapcs. pontja
   if (k >= nk)
      throw "Uzenet hiba";  // hiba, nincs ilyen végpont
   if (Obj *o = konn[k].GetConn(n)) {
         o->set(n, --msg);    // szomszéd működtető függvénye
   }
}
```


Drót obj. modellje



Drót

```
class Drot :public Obj {
protected:              // megengedjük a származtatottnak
  Message veg[2];  // két vége van, itt tároljuk az üzeneteket
public:
   Drot(char *n = "") : Obj(n, 2) {} // 2 végű obj. létrehozása
   void set(Pin n, Message& msg); // működtet
};

void Drot::set(Pin n, Message& msg) {
   if (veg[n] != msg) {          // ha változott
      veg[n] = msg;              // megjegyezzük és
      uzen(n^1, msg);            // elküldjük a másik végére (vezet)
   }
}
```


Csomópont

```
class Csp :public Obj {
protected:              // megengedjük a származtatottnak
  Message veg[3];  // három vége van, itt tároljuk az üzeneteket
public:
   Csp(char *n = "") : Obj(n, 3) {} // 3 végű obj. létrehozása
   void set(Pin n, Message& msg); // működtet
};

void Csp::set(Pin n, Message& msg) {
   if (veg[n] != msg) {          // ha változott
      veg[n] = msg;              // megjegyezzük és
      uzen((n+1)%3, msg);   // elküldjük a másik 2 végére
      uzen((n+2)%3, msg);
   }
}
```


Kapcsoló

```
class Kapcsolo :public Drot { // Drótból
   int be;                    // állapot
public:
   Kapcsolo(char *n = "") : Drot(n), be(0) {}
   void set(Pin n, Message& msg);
   void kikap() { be = 0; uzen(0, Message(Message::jel));
                          uzen(1, Message(Message::jel)); }
   void bekap() { be = 1; uzen(0, veg[1]); uzen(1, veg[0]);}
};
void Kapcsolo::set(Pin n, Message& msg) {
   if (be)  Drot::set(n, msg); // be van kapcsolva, drótként viselk.
   else  veg[n] = msg; // ki van kapcsolva, csak megjegyezzük
}
```

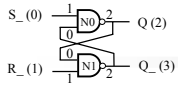
jel, false, lehetne undef

NAND kapu

```
class NAND :public Obj {
   Message veg[3];          // három "vége" van
public:
   NAND(char *n = "") : Obj(n, 3) {} // 3 végű obj. létreh.
   void set(Pin n, Message& msg);      // működtet
   const Message& get() { return(veg[2]); } // kim. lekérdezése
};
void NAND::set(Pin n, Message& msg) {
   if (n != 2 && veg[n] != msg) {   // ha változott bemenet
      veg[n] = msg;                       // megjegyezzük
      uzen(2, veg[2] = Message(Message::jel,
        !(veg[0].J * veg[1].J), msg.c)); // üzenünk a kimeneten
   }
}
```

kimenet előállítása

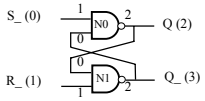
ciklusszám marad



S_-R_ *tároló*

```
class S_R_FF :public Obj {
protected:
   Message veg[4];            // négy "vége" van
   NAND N[2];                 // két db NAND kapu, komponens
public:
   S_R_FF(char *n) : Obj(n, 4) {
      N[0].setConn(2, N[1], 0); // összekötések létrehozása
      N[1].setConn(2, N[0], 0); }
   void set(Pin n, Message &msg); // működtet
   const Message& get(int i) {       // kimenet lekérdezése
      if (i >= 2) i = 0; return(veg[i+2]);}
};
```

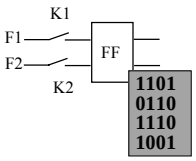

S_-R_ *tároló /2*




```
void S_R_FF::set(Pin n, Message &msg) {
   if (n < 2 && veg[n] != msg) {   // ha input és változott,
      veg[n] = msg;                      // letárolja
      N[n].set(1, msg);                  // megfelelő bemenetre küldi
      uzen(2, veg[2] = N[0].get());  // üzen a kimeneten
      uzen(3, veg[3] = N[1].get());  // üzen a kimeneten
   }
}
```

Szimulátorunk próbája

```
Kapcsolo K1("K1"), K2("K2");
Forras F1("F1"), F2("F2"); S_R_FF FF("FF");

try {
  F1.setConn(0, K1, 0);  FF.setConn(0, K1, 1);
  F2.setConn(0, K2, 0);  FF.setConn(1, K2, 1);

  F1.init(); F2.init();
  K1.bekap(); K2.bekap();
  cerr << FF.get(0).J << FF.get(1).J << FF.get(2).J << FF.get(3).J << endl;
  K1.kikap();
  cerr << FF.get(0).J << FF.get(1).J << FF.get(2).J << FF.get(3).J << endl;
  K1.bekap();
  cerr << FF.get(0).J << FF.get(1).J << FF.get(2).J << FF.get(3).J << endl;
  K2.kikap();
  cerr << FF.get(0).J << FF.get(1).J << FF.get(2).J << FF.get(3).J << endl;
} catch (const char *s) {  cerr << s << endl; }
```



1101
0110
1110
1001