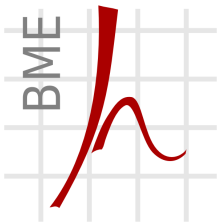
Származtatatás

C++



Híradástechnikai Tanszék

Izsó Tamás

2014. március 19.

Dinamikus tagváltozó az osztályban

```
class RVektor {
int n;
Racionalis *p;
public:
RVektor( int n=10 )
      : n(n), p( new Racionalis[n] )
{}

~RVektor() {delete [] p; }
   . . .
};
```

Vigyázz a copy konstruktorra és értékadó operátorra

Most nem jó a fordító által generált copy konstruktor és értékadó operátor.

- Kivülről elérhetetlenné kell tenni (nem mindig a legjobb megoldás)
- Meg kell írni.

copy ctor és operator=()

```
class RVektor {
    int n;
    Racionalis *p;
public:
    . . .
    RVektor( const RVektor& other ) {
        n = other.n;
        p = new Racionalis[n];
        for( int i=0; i<n; i++) p[i] = other.p[i];
    }

    RVektor& operator=( const RVektor& other )
    {
        if( this != &other ) {
            delete [] p;
            n= other.n;
            p = new Racionalis[n];
            for( int i=0; i<n; i++) p[i] = other.p[i];
        }
        return *this;
    }
    . . .
};
```

copy ctor írás operator=() segítségével

```
class RVektor {
    int n;
    Racionalis *p;
public:
    . . .
    RVektor( const RVektor& other ) {
        p = 0;
        *this = other;
    }

    RVektor& operator=( const RVektor& other )
    {
        if( this != &other ) {
            delete [] p;
            n= other.n;
            p = new Racionalis[n];
            for( int i=0; i<n; i++) p[i] = other.p[i];
        }
        return *this;
    }
    . . .
};
```

operator=() megírása copy ctor felhasználásával

```
class RVektor {
    int n;
    Racionalis *p;
public:
    . . .

    void swap( RVektor& other )
    {
        int s = other.n; other.n = n; n = s;
        Racionalis* v = other.p; other.p = p; p = v;
    }

    RVektor( const RVektor& other ) { . . . }

    RVektor& operator=( const RVektor& other )
    {
        RVektor v( other );
        swap( v );
        return *this;
    }
    . . .
};
```

operator=() megírása copy ctor felhasználásával egyszerűbben

```
class RVektor {
    int n;
    Racionalis *p;
public:
    . . .
    void swap( RVektor& other )
    {
        int s = other.n; other.n = n; n = s;
        Racionalis* v = other.p; other.p = p; p = v;
    }

    RVektor( const RVektor& other ) { . . . }

    RVektor& operator=( RVektor other )
    {
        swap( other );
        return *this;
    }
    . . .
};
```

Index operátorok

```
class RVektor {
    int n;
    Racionalis *p;
public:
    . . .

    Racionalis& operator[](int i)
    {
        if( i<0 || i>=n ) throw "Index hiba";
        return p[i];
    }

    const Racionalis operator[](int i) const
    {
        if( i<0 || i>=n ) throw "Index hiba";
        return p[i];
    }


    . . .
```

Mit ír ki?

```
class A {
    int k;
public:
    A(const int i = 0) :k(i){ cout << 'k'; }
    A(const A& a) { k = a.k; cout << 'c'; }
    void operator=(A& a) { k = a.k; f(a); cout << 'e'; }
    A& operator*(int i) { cout << i*100; return *this; }
    void f(A a) { k++; cout << 'f'; }
    ~A() { cout << 'd'; }
};

A& operator*( int i, A& a ) { cout << i; return a; }

int main(int argc, char* argv[]){
    A a = A(1);    cout << '\n';
```

Mit ír ki?

```
class A {
    int k;
public:
    A(const int i = 0) :k(i){ cout << 'k'; }
    A(const A& a) { k = a.k; cout << 'c'; }
    void operator=(A& a) { k = a.k; f(a); cout << 'e'; }
    A& operator*(int i) { cout << i*100; return *this; }
    void f(A a) { k++; cout << 'f'; }
    ~A() { cout << 'd'; }
};

A& operator*( int i, A& a ) { cout << i; return a; }

int main(int argc, char* argv[]){
    A a = A(1);    cout << '\n'; //kcd
    a = a * 2; cout << '\n';
```

Mit ír ki?

```
class A {
    int k;
public:
    A(const int i = 0) :k(i){ cout << 'k'; }
    A(const A& a) { k = a.k; cout << 'c'; }
    void operator=(A& a) { k = a.k; f(a); cout << 'e'; }
    A& operator*(int i) { cout << i*100; return *this; }
    void f(A a) { k++; cout << 'f'; }
    ~A() { cout << 'd'; }
};

A& operator*( int i, A& a ) { cout << i; return a; }

int main(int argc, char* argv[]){
    A a = A(1);    cout << '\n'; //kcd
    a = a * 2; cout << '\n';     //200cfde
    a = 3 * a; cout << '\n';
```

Mit ír ki?

```
class A {
    int k;
public:
    A(const int i = 0) :k(i){ cout << 'k'; }
    A(const A& a) { k = a.k; cout << 'c'; }
    void operator=(A& a) { k = a.k; f(a); cout << 'e'; }
    A& operator*(int i) { cout << i*100; return *this; }
    void f(A a) { k++; cout << 'f'; }
    ~A() { cout << 'd'; }
};

A& operator*( int i, A& a ) { cout << i; return a; }

int main(int argc, char* argv[]){
    A a = A(1);    cout << '\n'; //kcd
    a = a * 2; cout << '\n';     //200cfde
    a = 3 * a; cout << '\n';     //3cfde
    return 0;
}
```

Mit ír ki?

```
class A {
    int k;
public:
    A(const int i = 0) :k(i){ cout << 'k'; }
    A(const A& a) { k = a.k; cout << 'c'; }
    void operator=(A& a) { k = a.k; f(a); cout << 'e'; }
    A& operator*(int i) { cout << i*100; return *this; }
    void f(A a) { k++; cout << 'f'; }
    ~A() { cout << 'd'; }
};

A& operator*( int i, A& a ) { cout << i; return a; }

int main(int argc, char* argv[]){
    A a = A(1);    cout << '\n'; //kcd
    a = a * 2; cout << '\n';     //200cfde
    a = 3 * a; cout << '\n';     //3cfde
    return 0;
}                                //d
```

Mit ír ki?

```
class String {
  char *p; int size;
public:
  String( ):p(new char[2]) { cout << 1 ; }
  String(const char *):p(new char[2]){ cout << 2; }
  String(String&s):p(new char[2]){ cout << 3; }
  ~String( ){delete [] p;  cout << 4; }
  String operator+(String&s1){
     String s; cout << 5; return s;}
  char& operator[](int){ cout << 6 ; return p[0]; }
  String& operator=(const String s) {
     cout << 7 ; return *this; }
};

int main(int argc, char* argv[]){
  String s1("rejtvény"); cout << endl;
  String s2; cout << endl;
  String s3=s2; cout << endl;
  char c = s3[3]; cout << endl;
  s2 = s3; cout << endl;
  s2 = s3 + s2 + s1; cout << endl;
  return 0;
}
```

Section 1

osztályok kapcsolata

Rétegelés

```
class Address { ..... };
class Name { ..... };
class Person {
    Name name;
    Address addr;
    .....
};
```

Reláció

A rétegelés *"vanegy"*, vagy *keresztül implementált* relációt modellez.

Analítikus (nyilvános) öröklés

```
class Person { ..... };

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

Reláció

Az analítikus öröklődés *(isa) "azegy"* relációt jelent. Még helyesebben az *"úgy működik mint"* kapcsolatot jelenti.

A fenti példában a diák `(Student)` *"azegy"* személy `(Person)` reláció igaz, tehát használhatjuk a publikus öröklődést.

Származtatott osztály használata

```
// Minden személy táncolhat
void dance( const Person& p);

// Csak a diákok tanulnak.
void study( const Student& s );

Person p;
Student s;

dance(p);   // OK. p egy személy
dance(s);   // rendben s egy diák, "azegy"  személy

study(s);   // OK.
study(p);   // Hiba p nem diák.
```

Korlátozó öröklés

```
class Rectangle { ..... };
class Square : private Rectangle {
    .....
};
```

Reláció

A korlátozó öröklés nem *(isa) "azegy"* relációt, mivel a származtotott objektum nem *"úgy működik mint"* kapcsolatban áll az alapobjektummal.

Alaposztály létrehozás

```
class Allat {
private:
    char nev[20];
public:
    Allat(const char *nev)  {
        strcpy(this->nev, nev);
    }

    virtual void bemutatkozik() {
        cout <<  nev << "␣vagyok␣";
    }
    virtual ~Allat() {}
};
```

Jótanács

C++-ban kerüljük a világobjetumokat! Azaz ne származtassunk olyan típusokat egy alaposztályból, akiknek semmi közük nincs egymáshoz.

Származtatás

```
class Kutya : public Allat {
  char gazdi[20];
public:
  Kutya(const char *nev, const char* gazdi) : Allat(nev)
  {
    strcpy(this->gazdi,gazdi);
  }

  void bemutatkozik() {
    Allat::bemutatkozik();
    cout << "␣gazdam␣" << gazdi;
  }
};

void hogy_hivnak( Allat* allat) {
   allat->bemutatkozik();
   cout << endl;
}
```

Származtatás egyik tulajdonsága

Alapelv

Mindenhol, ahol az alaposztály használható, ott a származtatott osztályt is használható. Fordítva nem igaz!

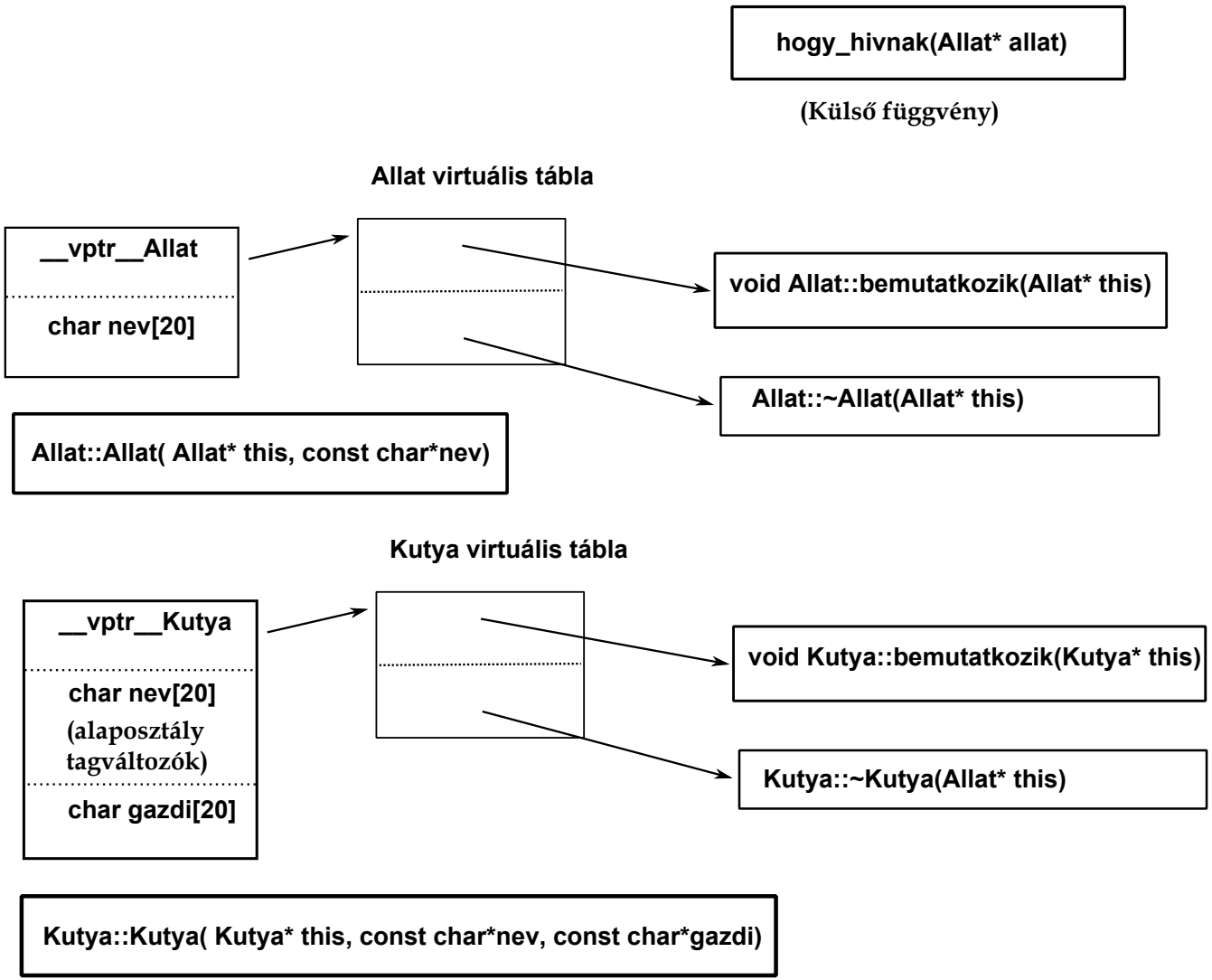
Többalakú viselkedés

```
void hogy_hivnak( Allat* allat) {
    allat->bemutatkozik();
    cout << endl;
}

int main() {

   Allat* frakk = new Kutya("Frakk","Karoly bacsi");
   Allat* pok = new Allat("Odon");

   hogy_hivnak(frakk);
   hogy_hivnak(pok);
   return 0;
}
```

Output:

```
Frakk vagyok  gazdam Karoly bacsi
Odon vagyok
```

Virtuális függvény működése



Objektumorientált programozás

Motto

Aki nem használt még virtuális függvényt, az nem írt még objektumorientált programot

Virtuális függvény felüldefiniálás

```
class B {
public:
    virtual void f() const { cout << "B::f␣␣"; }
    void g() const { cout << "B::g␣␣"; }
};

class D : public B {
public:
    void f() const { cout << "D::f␣␣"; }
    void g() { cout << "D::g␣␣"; }
};

class DD : public D {
public:
    void f() { cout << "DD::f␣␣"; }
    void g() const { cout << "DD::g␣␣"; }
};
```

Virtuális függvény felüldefiniálás

```
void call(const B& b) {
    b.f();
    b.g();
}
int main() {
    B b;      D d;      DD dd;

    call(b);          // B::f   B::g
    call(d);          // D::f   B::g
    call(dd);         // D::f   B::g
    b.f();            // B::f
    b.g();            // B::g
    d.f();            // D::f
    d.g();            // D::g
    dd.f();           // DD::f
    dd.g();           // DD::g
    return 0;
}
```

Szeletelődés

```
void hogy_hivnak( Allat allat) {
    allat.bemutatkozik();
    cout << endl;
}
```

Output:

```
Frakk vagyok
Odon vagyok
```

Komplex példa

```
class Allat {
    char *nev;
    void operator=(const Allat&);
public:
    Allat(const char *nev)  {
        this->nev = new char[strlen(nev) + 1];
        strcpy(this->nev, nev);
    }
    Allat(const Allat& other ) {
        nev = new char[strlen(other.nev)+1];
        strcpy( nev, other.nev);
    }
    virtual void hangot_ad() = 0;
    virtual void bemutatkozik() = 0 {
        cout <<  nev << "␣vagyok␣";
    }
    virtual ~Allat() {
        cout<< "~" << nev << endl;
        delete [] nev;
    }
};
```

Kérdések

- Miért virtuális a hangot_ad(), és a bemutatkozik() függvény?
- Miért kell destruktor?
- Miért kell virtuális destruktor?
- Mit jelent a bemutatkozik() függvény utáni kód.

Származtatott osztály

```
class Eger : public Allat {
public:
    Eger(const char *nev) : Allat(nev) { }
    void hangot_ad() { cout << "cin-cin " << endl; }
    void bemutatkozik() {
         Allat::bemutatkozik(); cout <<  endl;
    }
    . . .
};
```

- Miért kell meghívni az alaposztály konstruktorát.
- Mért kell a bemutatkozik() függvényben az alaposztály bemutatkozik() függvényét meghívni.
- Mi történt volna, ha a bemutatkozik() függvényben az alaposztály névterét nem írtuk volna ki.

Kérdések

- Miért elég az alaposztályban eldugni az értékadó operátort, és miért nem kellett a származtatott osztályokban?
- Mi történik, ha nem dugjuk el az értékadó operátort, és a főprogramban a következőt írjuk:
  $*($ farm[1] $)=$ $*($ farm[3] $)$;

Klónozás

Hogyan hozhatunk létre alaposztályra mutató pointer alapján egy hasonló származtatott objektumot?

```
Allat* dolly = new Barany("Dolly");

Barany* dolly2 = ???
```

Klónozás virtuális függvény felhasználásával

```
class Allat {
public:
    Allat(const Allat& other ){. . .}
    virtual Allat* clone() = 0;
};
class Eger : public Allat {
public:
   . . .
   Eger* clone() { return new Eger(*this); }
};
class Barany : public Allat {
public:
    . . .
    Barany* clone() { return new Barany(*this); }
};

int main() {
   . . .
   Allat* dolly = new Barany("Dolly");
   . . .
   Barany* dolly2 = dolly->clone();
   . . .
}
```

Qt platformfüggetlen toolkit

Fejlesztők:

- Trolltech (1991-2008)
- Nokia (2008-2011)
- Digia (2010-)
- Qt Project (2011-)

Alakzat osztály

```
class Alakzat {
protected:
     Pont p0;      /// alakzat origója
     QPen pen;     /// rajzceruza
public:
    Alakzat(const Pont& p0, const QPen& pen) :p0(p0), pen(pen) {}

    const Pont& origo() const { return p0; }

    virtual void rajzol( QPainter& painter ) const = 0 // tisztán virtuális
   {
      painter.setPen(pen);
   }

   void mozgat(const Pont& d );
};

class Kor : public Alakzat {
    int r;  /// sugár
public:
    Kor(const Pont& p0, int r, const QPen& pen)
       : Alakzat(p0, pen), r(r)   /// ős osztály inic
    {}

    /// kört rajzol
    void rajzol( QPainter& ) const;
};
```

Főprogram

```
int main( int argc, char **argv )
{
    Alakzat* rajz[3];
    rajz[0]=new Kor( Pont(100,100), 20, QPen(Qt::green, 2) );
    rajz[1]=new Kor( Pont(300,500), 30, QPen(Qt::red, 5) );
    rajz[2]=new Szakasz(Pont(200,400),100,200,QPen(Qt::black, 5) );

    QApplication app( argc, argv );

    MyWidget myWidget;
    myWidget.setAlakzat( rajz, 3 );
    myWidget.show();

    return app.exec();
}
```

Származtatott widget osztály

```
class MyWidget : public QWidget {
    Q_OBJECT
    bool moving;
    Alakzat **alakzat;
    int nalakzat;
    int x, y;
public:
    MyWidget()
      : moving(false), nalakzat(0)
    {}

    void setAlakzat( Alakzat** alakzat, int n ){
       this->alakzat = alakzat, nalakzat=n;
    };
protected:
    void paintEvent(QPaintEvent *event);
    void mousePressEvent(QMouseEvent * event);
    void mouseReleaseEvent(QMouseEvent * event);
    void mouseMoveEvent(QMouseEvent * event);
};
```

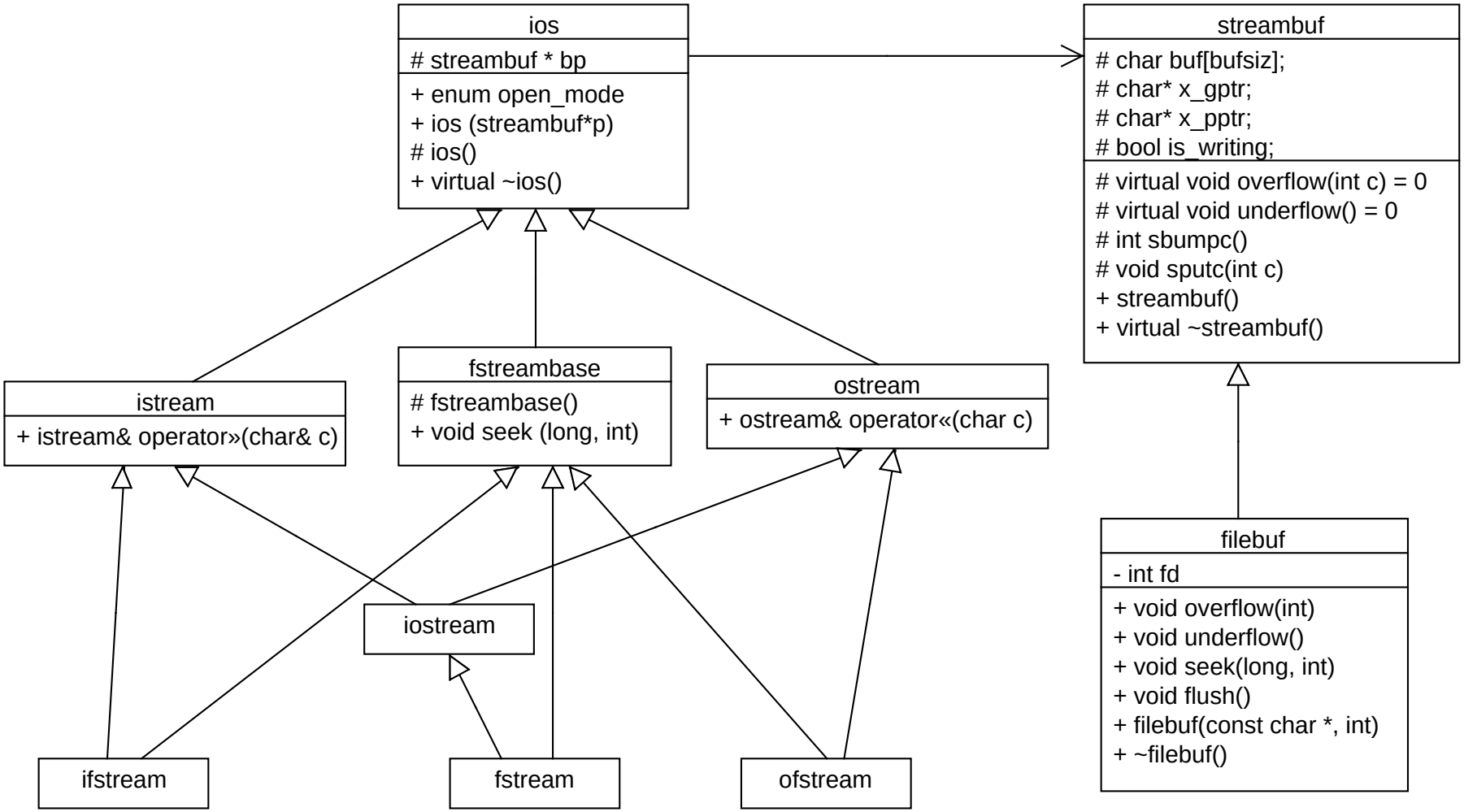
virtuális függvények átdefiniálása

```
void MyWidget::mousePressEvent(QMouseEvent * event) {
  if (event->button() == Qt::LeftButton) {
    moving = true;
    this->setCursor(Qt::OpenHandCursor);
    x =  event->x(); y = event->y();
  }
  event->accept();
}
void MyWidget::mouseReleaseEvent(QMouseEvent * event) {
  if (event->button() == Qt::LeftButton) {
    moving = false;
    setCursor(Qt::ArrowCursor);
  }
  event->accept();
}
void MyWidget::mouseMoveEvent(QMouseEvent * event) {
  if (moving) {
    int dx =  event->x()-x;
    int dy =  event->y()-y;
    x =  event->x(); y = event->y();
    for(int i=0; i<nalakzat; i++)
      alakzat[i]->mozgat( Pont(dx, dy) );
    update(); event->accept();
  }
}
```

Rajzolás

```
void MyWidget::paintEvent(QPaintEvent *event)
{
  QPainter painter(this);

  for(int i=0; i<nalakzat; i++)
    alakzat[i]->rajzol( painter );
}
```

Mini iostream



Jonathan E. Shopiro An Example of Multiple Inheritance in C++: A Model of the Iostream Library

class streambuf

```
class streambuf {
protected:
  // encapsulates buffer management
  char buf[bufsiz]; // the character buffer
  char* x_gptr; // get pointer
  char* x_pptr; // put pointer
  bool is_writing; // buffer must be written
  virtual void overflow(int c) = 0; // send full buffer
  virtual void underflow() = 0; // refill empty buffer
  int sbumpc() { // get a character
    if (x_gptr >= &buf[bufsiz]) underflow() ;
    return *x_gptr++;
  }

  void sputc(int c) { // put a character
    if (x_pptr >= &buf[bufsiz]) overflow(c) ;
    else *x_pptr++ = c;
  }
public:
  streambuf() : is_writing(false), x_gptr(&buf[bufsiz]),
                x_pptr(&buf[bufsiz])
  {}
  virtual ~streambuf() {}
};
```

class ios, istream, ostream, iostream

```
class ios { // a connection to a streambuf
public :
  enum open_mode { in=1, out=2 } ;
  ios (streambuf*p) : bp(p) {}
  virtual ~ios() { delete bp; }
protected:
  streambuf * bp;
  ios() {} // never called
};

class istream : public virtual ios { // input stream class
public:
  istream& operator>>(char& c) {
    c = bp->sbumpc(); return *this;
  }
};

class ostream : public virtual ios { // output stream class
public:
  ostream& operator<<(char c) {
    bp->sputc(c); return *this;
  }
};

class iostream : public istream, public ostream {};
```

class filebuf, fstreambase

```
class filebuf : public streambuf {
  // a streambuf for files
  int fd;
  void overflow(int) ;
  void underflow() ;
  void seek(long offset, int whence);
  void flush() { if (is_writing) write(fd, buf, x_pptr-buf); }
  filebuf(const char *name, int om);
  ~filebuf() ;
};

class fstreambase : virtual public ios {
protected:
   fstreambase() { }
public:
   // the member streambuf: :bp is here known to point to a filebu
   void seek (long offset, int whence) {
      ( (filebuf*)bp)->seek (offset, whence) ;
  }
};
```

file stream-ek

```
class ifstream : public fstreambase, public istream {
public:
  ifstream(const char* name)
    : ios(new filebuf(name, ios::in))
  {}
};

class ofstream : public fstreambase, public ostream {
public:
  ofstream(const char* name)
    : ios(new filebuf(name, ios::out))
  {}
};

class fstream : public fstreambase, public iostream {
public:
  fstream(const char* name)
    : ios(new filebuf(name,ios::in | ios::out))
  {}
};
```

Pufferelt írás

```
ostream& ostream::operator<<(char c) {
  bp->sputc(c);
  return *this;
}

void streambuf::sputc(int c) { // put a character
  if (x_pptr >= &buf[bufsiz]) overflow(c) ;
  else *x_pptr++ = c;
}

void filebuf::overflow (int c) {
  flush () ;
  is_writing = true;
  x_pptr = buf;
  x_gptr = &buf[bufsiz];
  sputc (c) ;
}

void filebuf::flush(){ if (is_writing) write(fd, buf, x_pptr-buf); }
```

filebuf konstruktor és destruktor

```
filebuf::filebuf(const char *name, int om) {
    switch (om) {
    case ios::in:
        fd = open (name, O_RDONLY) ; break;
    case ios::out:
        fd = creat (name, 0666) ; break;
    case ios::in | ios::out:
        fd = open(name, O_RDWR | O_CREAT, 0666); break;
    }
}


filebuf:: ~filebuf () {
  flush() ;
  close(fd) ;
}
```

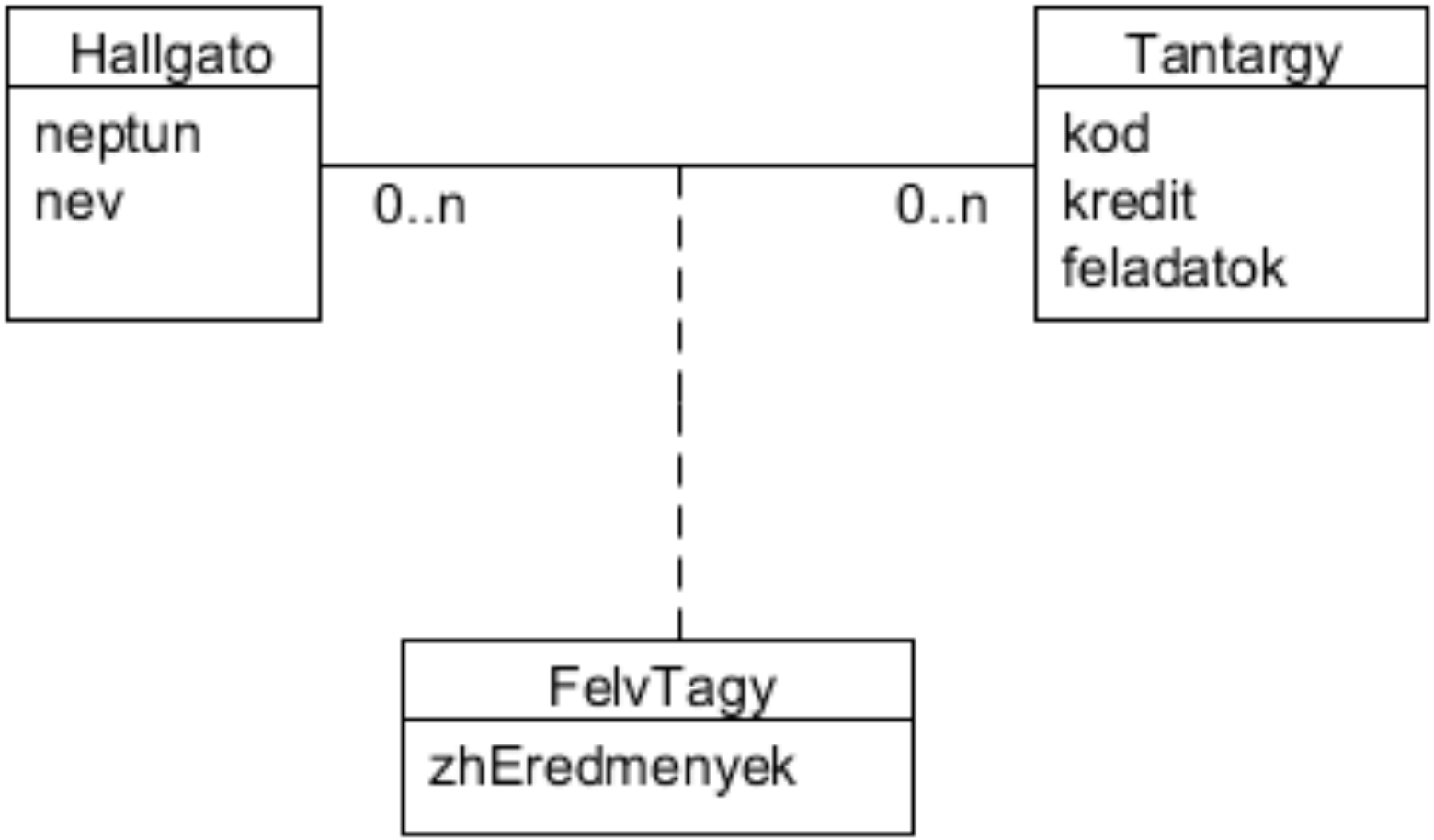
Hallgatói C++ modell

- Az évfolyamot hallgatók alkotják.
- A hallgatók különböző tantárgyakat vesznek fel.
- A hallgatók kiszh-kat írnak, és az év végén ezek alapján kapják az érdemjegyet.

Funkciók:

- tárgy felvétele
- óra látogatása
- következő kiszh megírása
- ZH-n elért eredmény lekérdezése
- hallgatói átlag lekérdezése

Első modell


Hallgato
neptun
nev
0..n
0..n
Tantargy
kod
kredit
feladatok
FelvTagy
zhEredmenyek

Hallgatói modell finomítása

Szereplők:

- évfolyam
- hallgató
- tantárgy
- felvett tantárgy
- kiszh eredmények
- kiszh feladatok

Évfolyam

Attribútum: Hallgatók halmaza

Metódusok:

- felvesz egy hallgatót
- töröl egy hallgatót
- listáz

Hallgató

Attribútum:

- név
- neptun kód
- felvett tantárgyak

Metódusok:

- név lekérdezés
- neptun kód lekérdezés
- tantárgy felvétel
- orára jár
- kiszh-t ír
- egy tantárgyból írt kzh-k számának a lekérdezés
- kiszh eredmények lekérdezése
- átlagszámítás

Tantárgy

Attribútum:

- kód
- kredit
- kiszh feladatok

Metódusok:

- kód lekérdezés
- kredit lekérdezés
- követelmények közlése
- kiszh íratás
- jegy kiszámítás

Felvett tantárgy

Attribútum:

- hallgató
- tantárgy
- jelenlét
- megírt kiszh-k

Metódusok:

- tantárgy lekérdezés
- hallgató lekérdezés
- jelenlét lekérdezés
- kiszh eredmény lekérdezése
- kiszh íratás
- jegy kiszámítás

Kiszh eredmény

Attribútum:

- kiszh sorszáma
- pont
- dátum

Metódusok:

- pont lekérdezés
- sorszám lekérdezés
- dátum lekérdezés

Kiszh feladat

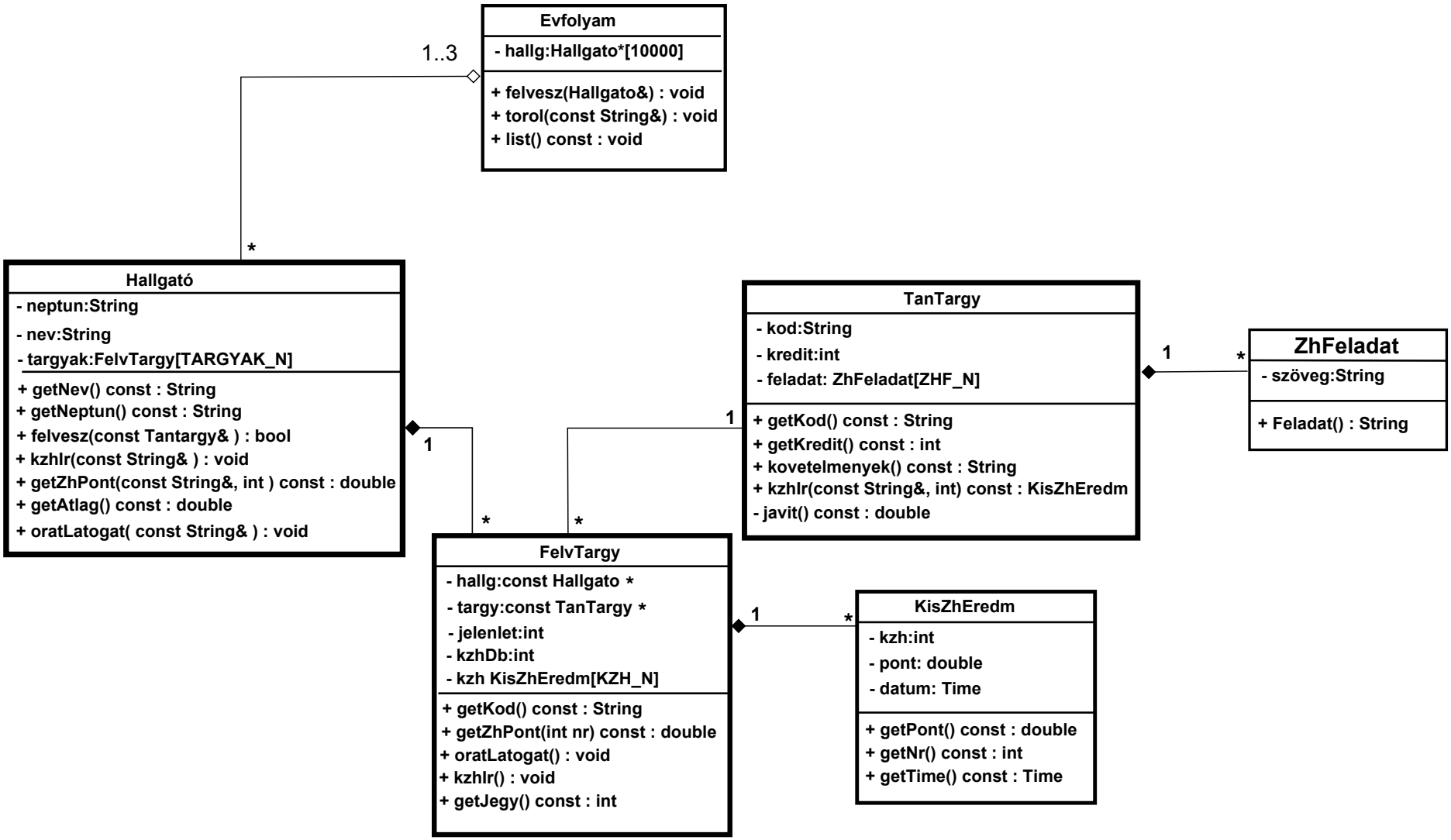
Attribútum:

- megoldandó feladat

Metódusok:

- feladatok lekérése

Második modell



Kiszh írás

```
void Hallgato::kzhIr(const String& targykod) {
    // megkeressuk a felvett targyak kozul:
    int i;
    for (i=0; i<targyDb && targyak[i].getKod()!=targykod; i++);
    // ha nincs meg, hiba:
    if (i==targyDb) throw "Nincs ilyen tantargy!";
    // ha megvan, ZH-t ir a targybol:
    targyak[i].kzhIr();
}
void FelvTargy::kzhIr() {
    if (kzhDb>=KZH_N) throw "Nem tud tobb ZH-t irni!";
    KisZhEredm zh = targy->kzhIr( hallg->getNeptun(), kzhDb+1 );
    kzh[kzhDb] = zh;
    kzhDb++;
}
KisZhEredm Tantargy::kzhIr(const String& neptun, int nr) const {
   double pont = javit(nr);
    time_t t; time(&t);    // datum:
    String datum(ctime(&t));
    KisZhEredm zh(nr, pont, datum);
    return zh;
}
```