Programozás alapjai II.
(4. ea) **C++**

konstruktor és értékadás, dinamikus szerkezetek

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

Hol tartunk ?

- C → C++ javítások
- OO paradigmák, objektum fogalma

A C++ csupán eszköz:

- objektum megvalósítása
  - osztály (egységbe zár, és elszigetel),
  - konstruktor, destruktor, tagfüggvények
  - alapértelmezett operátorok, és tagfüggvények
  - operátor átdefiniálás (függvény átdefiniálás)
- Elegendő eszköz van már a kezünkben?

Konstr: létrehoz+inic. (ism.)

```
class Komplex {
   double re, im;
 public:
   Komplex() { re = 0; im = 0; }
   Komplex(double r) { re = r; im = 0; }
   Komplex(double r, double i) { re = r; im = i; }
...
};
Komplex k;              // default
Komplex k1(1);          // 1 paraméteres
Komplex k2(1, 1);       // 2 paraméteres
```


Inicializáló lista

```
class Valami {
   const double c1 = 3.14;  // inicializálni kell, de hogyan?
   Komplex k1;
 public:
   Valami(double c) { c1 = c }
   Valami(double c) :c1(c) { }
   Valami(double c, Komplex k) :c1(c), k1(k) { }
};
```

Konstans tag, és referencia tag, csak inicializáló listával inicializálható. Célszerű a tagváltozókat is inicializáló listával inicializálni (felesleges műveletek elkerülése).

Destruktor: megszüntet (ism.)

```
class String {
        char *p;
        int    size;
public:
        String(const char *s = "") {
            p = new char[(size = strlen(s)) + 1]; // terület foglalás
            strcpy(p, s);
        }
        ~String() { delete [] p; } // foglalt terület felsz.
};
```

A p és a size megszüntetése automatikus, ahogy egy lokális változó is megszűnik. A new-val foglalt dinamikus terület felszabadítása azonban a mi feladatunk.

Konstans tagfüggvények

```
class Komplex {
   double re, im;
 public:
   Komplex(double r = 0, double i = 0) : re(r), im(i) { }
   double Re() const { return(re); }
   double Im() const { return(im); }
   double Abs() const;
};
double Komplex::Abs() const
{
        return(sqrt(re*re + im*im));
}
```

Nem változtathatja meg az állapotot (adatokat)

Alapértelmezett tagfüggvények

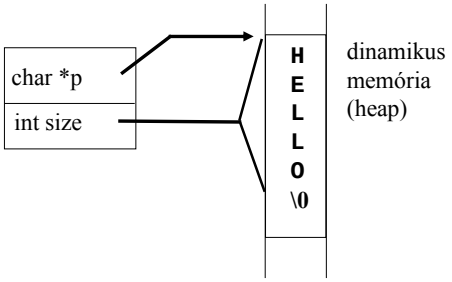
- Konstruktor
  - default: X() // nincs paramétere
  - másoló: X(const X&)// referencia paraméter
- operator=(const X&) // értékadó
- operator&() // címképző
- operator,(const X&) // vessző

A másoló konstruktor és az értékadó operátor alapértelmezés szerint meghívja az adattagok megfelelő tagfüggvényét. Alaptípus esetén bitenként másol!

Példa: Intelligens string

- String tárolására alkalmas objektum, ami csak annyi helyet foglal a memóriában, amennyi feltétlenül szükséges. → dinamikus adatszerkezet
- Műveletei:
  - létrehozás, megszüntetés
  - indexelés: []
  - másolás: =
  - összehasonlítás: ==
  - összefűzés: (String + String), (String + char) (char + String)
  - kiírás: cout <<
  - beolvasás: cin >>

String adatszekezete



A String osztály

```
class String {
        char *p;
        int    size;
public:
        String(const char *s = "") {
            p = new char[(size = strlen(s)) + 1];
            strcpy(p, s);
        }
        ~String( ) { delete[] p; }
        char& operator[] (int i) { return p[i]; }
        ....
};
```

Ez a default konstruktor is

Itt nem fontos, de ...

Függvényhívás mint balérték

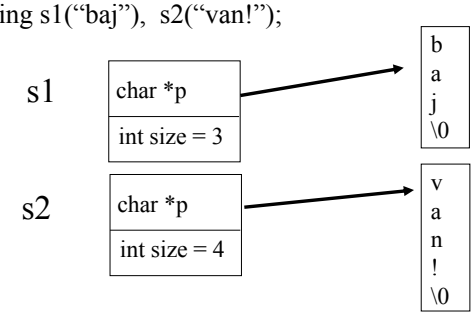
```
main ( ) {
        char c;
        String s( "Üdvözöllek dicső lovag" );

        c = s[3];       // c = s.operator[](3); ➔ c=p[3];

        s[2]='a';       // s.operator[](2) =' a'; ➔ p[2]='a';
}                       // destruktor: delete[] p
```

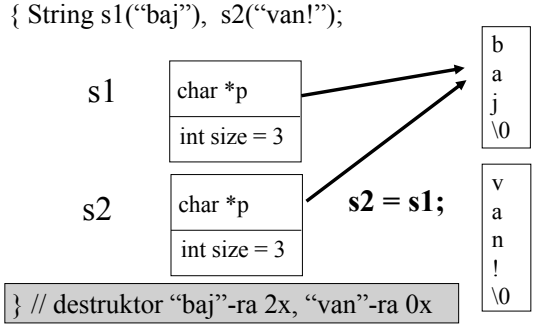

Értékadás problémája

{ String s1(“baj”), s2(“van!”);



Értékadás problémája/2

{ String s1(“baj”), s2(“van!”);



} // destruktor “baj”-ra 2x, “van”-ra 0x

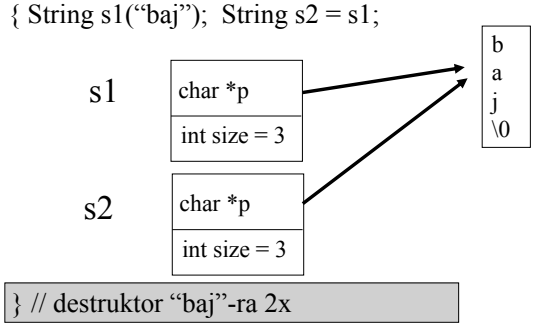
Megoldás: operátor= átdefiniálása

Paraméterként kapja azt, amit értékül kell adni egy **létező** objektumnak.

```
class String {
  ....
  String& operator=(const String& s) { // s1=s2=s3 miatt
       if (this != &s ) {              // s = s miatt
            delete[] p;
            p = new char[(size = s.size) + 1];
            strcpy(p, s.p);
       }
       return *this;                   // visszaadja saját magát
  }
};
```


Kezdeti értékadás problémája

{ String s1(“baj”); String s2 = s1;



} // destruktor “baj”-ra 2x

Megoldás: másoló konstruktor

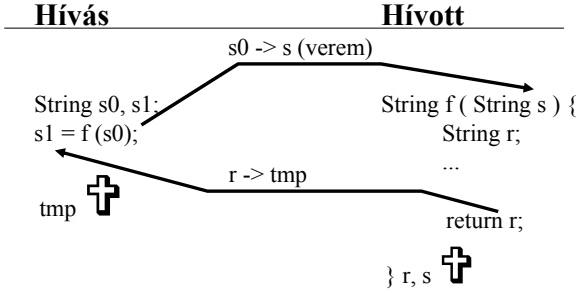
Referenciaként kapja azt a példányt, amit lemásolva létre kell hoznia **egy új** objektumot.

```
class String {
  ....
     String(const String& s) {
          p = new char[(size = s.size) + 1];
          strcpy(p, s.p);
     }
}
```


Miért más mint az értékadás?

- A kezdeti értékadáskor még inicializálatlan a változó (nem létezik), ezért nem lehet a másolással azonos módon kezelni.
- Mikor hívódik a másoló konstruktor?
  - inicializáláskor (azonos típussal inicializálunk)
  - függvény paraméterének átadásakor
  - függvény visszatérési értékének átvételekor
  - ideiglenes változók összetett kifejezésekben
  - kivétel átadásakor

Függvényhívás és visszatérés



Összetett algebrai kifejezés

String s, s0, s1, s2;
s = s0 + s1 + s2;

1. lépés: tmp1=s0+s1

2. lépés: tmp2=tmp1+s2

3. lépés: s = tmp2

4. lépés: tmp1, tmp2 megszüntetése destruktor hívással

String rejtvény

```
class String {
  char *p;
  int  size;
public:
  String( );                    // 1
  String(char *);               // 2
  String(String&);              // 3
  ~String( );                   // 4
  String operator+(String&);    // 5
  char& operator[](int);        // 6
  String& operator=(String&);   // 7
};
```

```
main( ) {
    String s1("rejtvény");   2
    String s2;               1
    String s3 = s2;          3

    char c = s3[3];          6
    s2 = s3;                 7
    s2 = s3 + s2 + s1        5,3,5,3,
                             (3),7,4,4,(4)
} // destr.                  4,4,4
```


String rejtvény/2

```
class String {
  char *p;
  int  size;
public:
  String( );                    // 1
  String(char *);               // 2
  String(String&);              // 3
  ~String( );                   // 4
  String operator+(String&);    // 5
  char& operator[](int);        // 6
  String& operator=(String);    // 7
};
```

```
main( ) {
    String s1("rejtvény");   2
    String s2;               1
    String s3 = s2;          3

    char c = s3[3];          6
    s2 = s3;                 3,7,4
    s2 = s3 + s2 + s1        5,3,5,3,
                             (3),7,4,4,(4)
} // destr.                  4,4,4
```

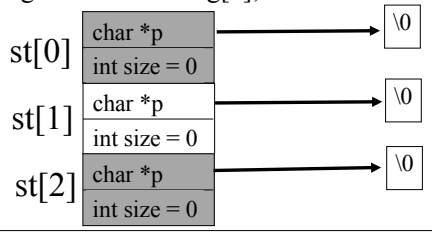

Miért referencia ?

Miért kell referencia a másoló konstruktorhoz?

- A paraméterátadás definíció szerint másoló konstruktort hív.
- Ha a másoló konstruktor nem referenciát, hanem értéket kapna, akkor végtelen ciklus lenne.

Miért fontos a delete[] ?

String *st = new String[3];



A delete st hatására csak a *st, azaz az st[0] destruktora hívódik meg! Az st[1] és az st[2] által foglalt memória nem szabadul fel! A delete[] meghívja minden elem destruktorát.

String +

```
class String {
    ....
    String operator+(String& s);
    String operator+(char c);
    friend String operator+(char c, String s);
};
String operator+(char c, String s) {
    char *p = new char[s.size + 2];
    *p = c; strcpy (p+1, s.p);
    String sr(p); delete[] p;
    return(sr);
}
```

Védelem enyhítése

Nem tagfüggvény!

Keletkezett-e += ?

- Az alaptípusokra meghatározott műveletek közötti logikai összefüggések nem érvényesek a származtatott típusokra.
- Azaz az operator= és az operator+ meglétéből nem következik az operator +=
- Ha szükség van rá, definiálni kell.

Változtatható viselkedés

- Feladat: "Varázsütésre" az összes String csupa nagybetűvel írjon ki!
- Megoldás: viselkedést befolyásoló jelző, de hol?
  - objektum állapota (adata) – csak az adott példányra van hatása.
  - globális változó – elég ronda megoldás !
  - az osztályhoz rendelt állapot: statikus tag ill. tagfüggvény.

Statikus tag

- Az osztályban statikusan deklarált tag nem példányosodik.
- Pontosan egy példány létezik, amit explicit módon definiálni kell (létre kell hozni).
- Minden objektum ugyanazt a tagot éri el.
- Nem szükséges objetummal hivatkozni rá.
  pl: String**::**SetUcase(true);
- Statikus tagként az osztály tartalmazhatja önmagát.
- Felhasználás: globális változók elrejtése

String statikus taggal

```
class String {
        char *p;   int   size;
   static bool ucase;          // statikus tag deklarálása
public:
   ....
   static bool SetUcase(bool b = true) {
        bool prev = ucase; ucase = b; return (prev);
   }
   friend ostream& operator<<(ostream& os, String& s);
};
bool String::ucase = false; // FONTOS: létre kell hozni !!
```


String statikus taggal /2

```
ostream& operator<<(ostream& os, String& s) {
      for (i = 0; i < s.size; i++) {
         char ch  = s.ucase ? toupper(s.p[i]) : s.p[i];
         os << ch;              // miért kell ch ?
       } return os;
}
```

Osztályhoz tartozik, nem a példányhoz

Komplex példa újból

- Olvassunk be adott/tetszőleges számú komplex számot és írjuk ki a számokat és abszolút értéküket fordított sorrendben!
- Objektumok:
  - Komplex,
  - KomplexTar
    - konstruktorban adott méret (a, változat)
    - igény szerint változtatja a méretét (b, változat)
  - Mindkét megoldás dinamikus memóriakezelést igényel. Ügyelni kell a helyes felszabadításra, foglalásra.

KomplexTar osztály

```
class KomplexTar {
   Komplex *t;          // pointer a dinamikusan foglalt tömbre
   int db;              // tömb mérete (elemek száma)
public:
   class Tar_Hiba {}; // osztály az osztályban a hibakezeléshez
   KomplexTar(int m = 10) :db(m) {
        t = new Komplex[m]; }        // konstruktor (def = 10)
   KomplexTar(const KomplexTar& kt);// másoló konstruktor
   Komplex& operator[](int i);      // indexelés
   KomplexTar& operator=(const KomplexTar& kt); // értékadás
   ~KomplexTar() { delete[] t;}     // felszabadítás
};
```


KomplexTar osztály/2

```
KomplexTar::KomplexTar(const KomplexTar& kt) {//másoló konst.
   t = new Komplex[db = kt.db];
   for (int i = 0; i < db; i++)  t[i] = kt.t[i]; // miért nem memcpy ?
}
```

A memcpy nem hívná meg a konstruktort

```
KomplexTar& KomplexTar::operator=(const KomplexTar& kt) {// =
   if (this != &kt) {
      delete[] t; t = new Komplex[db = kt.db];
      for (int i = 0; i < db; i++)  t[i] = kt.t[i]; // miért nem memcpy ?
   }
   return *this;
}
```

Visszavezettük értékadásra

```
KomplexTar::KomplexTar(const KomplexTar& kt){//másoló 2.vált.
        t = NULL;  *this = kt;  // trükkös, de rendben van !
}
```


Indexelés és a főprogram (a,)

```
Komplex& KomplexTar::operator[](int i) {
   if (i >= db || i < 0) throw Tar_Hiba();  return t[i];
}

int main() {
   KomplexTar t(5);              // a tárolóban 5 elemünk van
   try {
      for (int i = 0; i < 20; i++)   cin >> t[i];   // beolvasás
      KomplexTar t2 = t1;       // másoló konstruktor
      for (i = 19; i >= 0; i--)
         cout << t[i] ' ' << (double)t[i] << endl; // kiírás
   } catch (KomplexTar::Tar_hiba) {
        cerr << "Indexelesi hiba\n";   // hibakezelés
    }
    return(0);
}
```


Változó méretű KomplexTar (b,)

```
// Indexelés hatására növekszik a méret, ha kell
Komplex& KomplexTar::operator[](int i)
{
  if (i < 0) throw Tar_Hiba();   // hibás indexelés
  if (i >= db) {  // növekednie kell, célszerű kvantumokban
     Komplex *tmp = new Komplex[i+10]; // legyen nagyobb
     for (int j = 0; j < db; j++)  tmp[j] = t[j]; // átmásol
     delete[] t;    // régi törlése
     t = tmp;       // pointer az új területre
     db = i + 10; // megnövelt méret
  }
  return t[i];      // referencia vissza
}
```


Összefoglalás /1

- INICIALIZÁLÁS != ÉRTÉKADÁS
- Inicializáló lista szerepe.
- Alapértelmezett tagfüggvények.
- Dinamikus szerkezeteknél nagyon fontos a másoló konstruktor és az értékadás felüldefiniálása. (nem maradhat alapért.)
- Default konstruktornak fontos szerepe van a tömböknél.
- Egyparaméterű konstruktor → automatikus konverzió.

Összefoglalás /2

- Konstans tagfüggvények nem változtatják az objektum állapotát.
- Statikus tag és tagfüggvény az osztályhoz tartozik.
- Védelem enyhítése: friend
- Létrehozás, megsemmisítés feladatait a konstruktor és destruktor látja el.

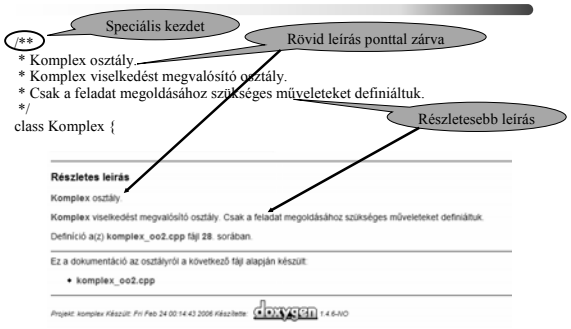
Létrehozás, megsemmisítés

- Konstruktor
  - default: X() // nincs paramétere automatikusan létrejön, ha nincs másik konst.
  - másoló: X(const X&) // referencia paramétere van, automatikusan létrejön: meghívja az adattagok másoló konst.-rát, ha objektumok, egyébként bitenként másol.
- Destruktor
  - delete[ ] // [ ] nélkül csak a 0. tömbelemre!!
  - automatikusan létrejön: meghívja az adattagok destr.
- operator=(const X&) // értékadó operátor automatikusan létrejön: meghívja az adattagok értékadó operátorát, ha objektumok, egyébként bitenként másol.

Milyen furcsa kommentek!

- A kommentekből automatikusan generál dokumentációt a Doxygen program. (html, latex, rtf, man, ... formátumban)
- Csak jó kommentből lesz jó dokumentáció!

Milyen furcsa kommentek! /2



Milyen furcsa kommentek! /3