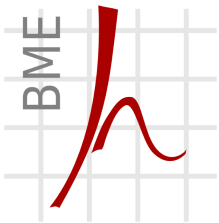
Felhasználó által definiált adattípus

C++

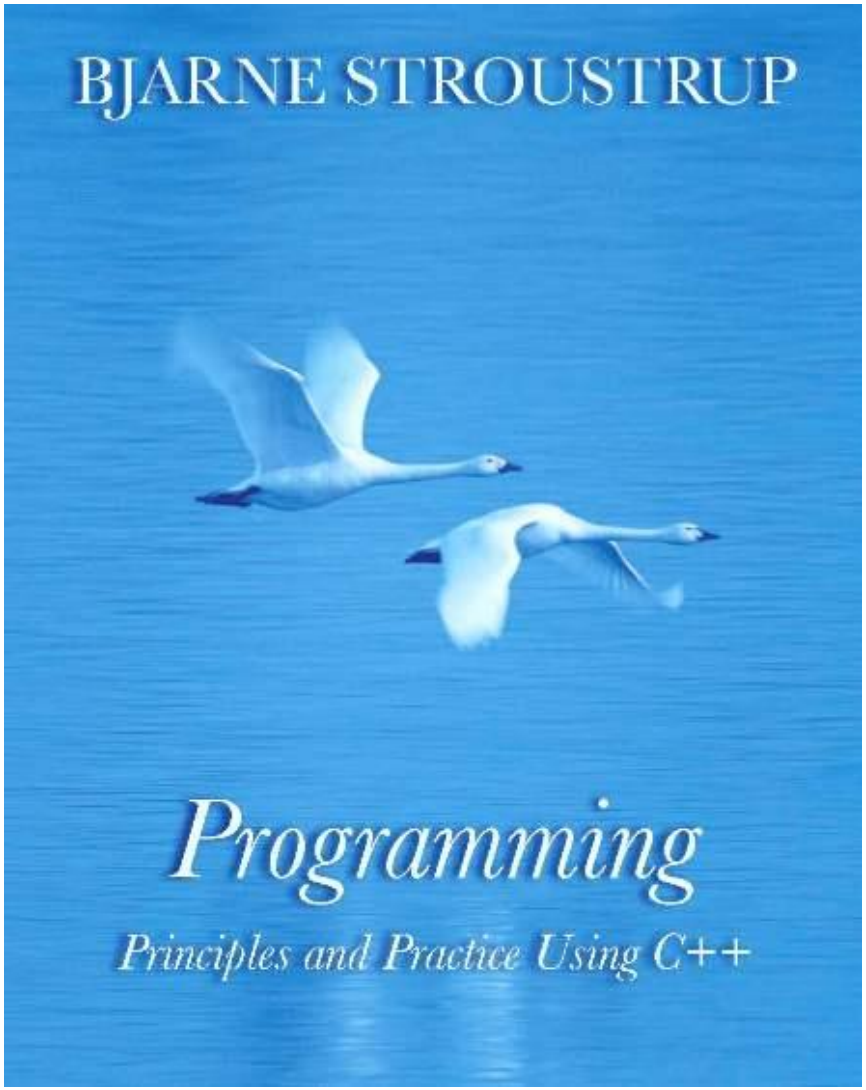


Híradástechnikai Tanszék

Izsó Tamás

2017. február 24.

Irodalom


BJARNE STROUSTRUP
Programming
Principles and Practice Using C++

Programtervezési szempontok

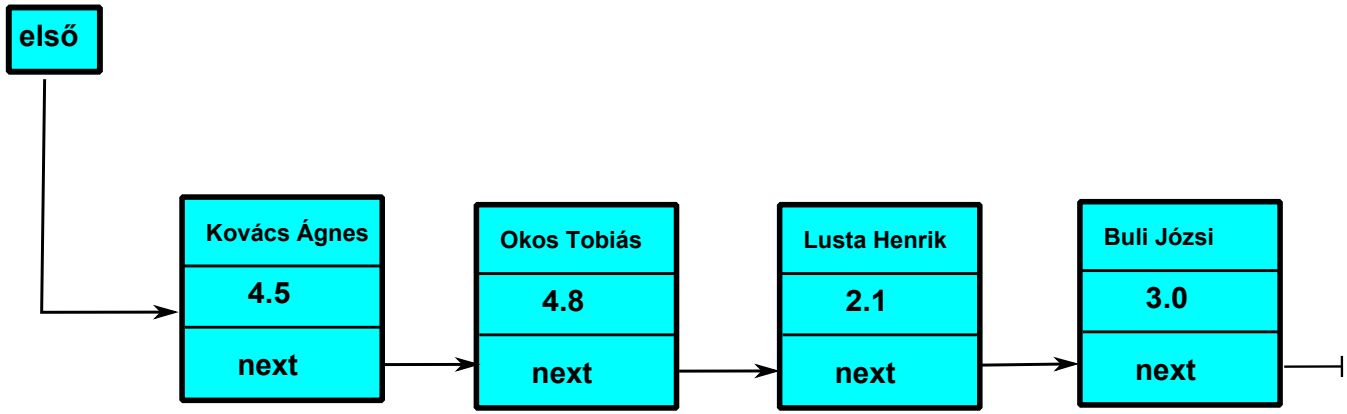
- Teljesítse a specifikációt.
- Legyen robosztus, extrém adatokra se szálljon el.
- Rugalmasan lehessen változtatni.
- A megírt részeket máshol is lehessen használni.
- Részek egymással tudjanak együttműködni.
- Legyen hatékony (futási idő, memória)
- Legyen hordozható.
- Könnyen lehessen használni.
- Forrás legyen könnyen érthető és jól dokumentált.

Funkcionális dekompozíció

Olvassuk be és írjuk ki azoknak a diákoknak az adatait, akiknek a tanulmányi átlaga az összátlag felett van. (megvalósítás nélkül)

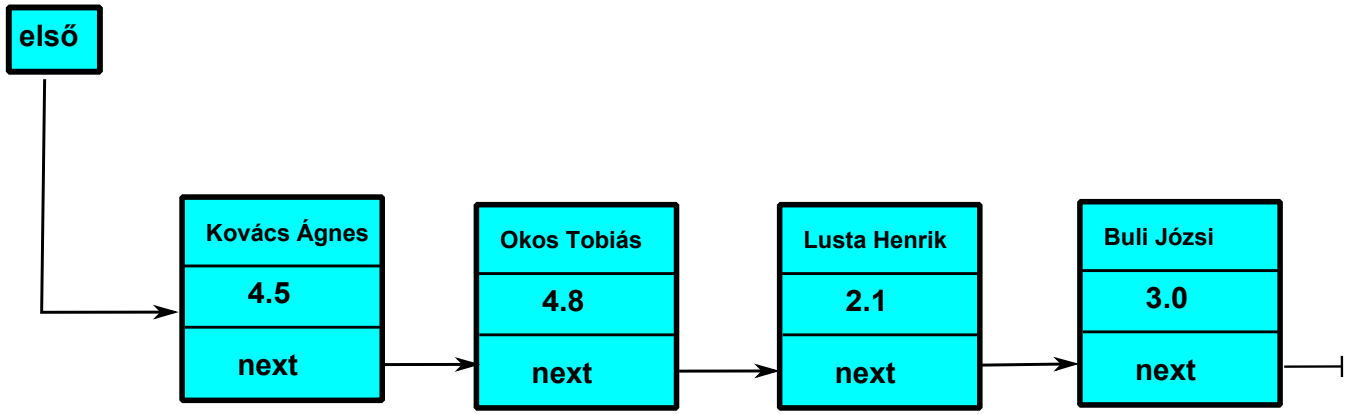
Funkcionális dekompozíció

Olvassuk be és írjuk ki azoknak a diákoknak az adatait, akiknek a tanulmányi átlaga az összátlag felett van. (megvalósítás nélkül)



Funkcionális dekompozíció

Olvassuk be és írjuk ki azoknak a diákoknak az adatait, akiknek a tanulmányi átlaga az összátlag felett van. (megvalósítás nélkül)



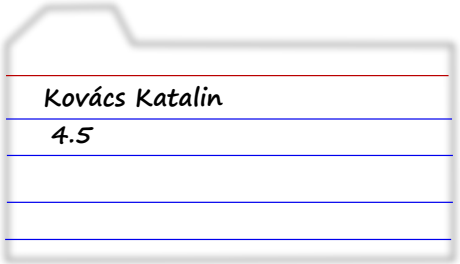

```
Lanc* elso;
double a;

elso = file_beolvasas("adat.txt");
a = atlag(elso);
print_okosok( elso, atlag);
```

OO dekompozíció

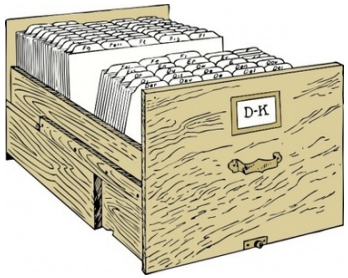
Szereplők:

OO dekompozíció

Szereplők:



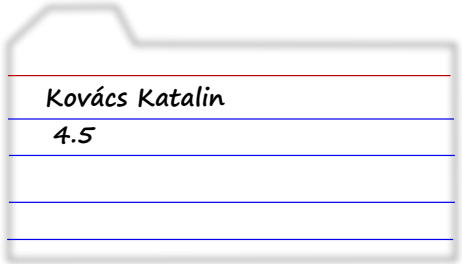
egy diák adata



tankör

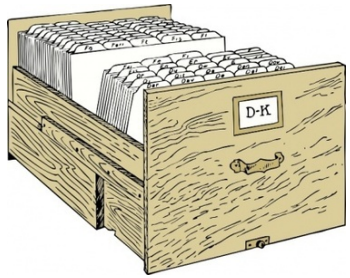
OO dekompozíció

Szereplők:



egy diák adata

```
beolvas();
kiir();
get_atlag();
get_nev();
```



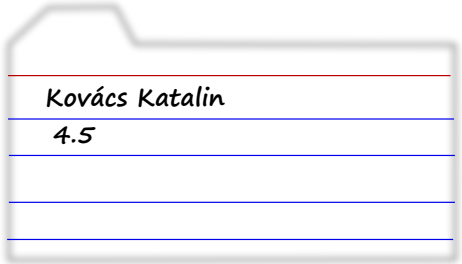
tankör

```
betesz();
kivesz();
elso_adat();
van_meg_adat();
kovetkezo_adat();
```

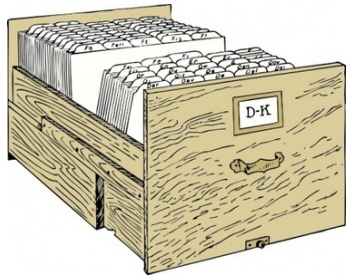
A megadott függvények képzik az adatok interfészét.
Mi van az implementációval?

OO dekompozíció

Szereplők:



egy diák adata



tankör

```
beolvas();
kiir();
get_atlag();
get_nev();
```

```
betesz();
kivesz();
elso_adat();
van_meg_adat();
kovetkezo_adat();
```

A megadott függvények képzik az adatok interfészét.
Mi van az implementációval? Kivülről nem látszik.

Diak_adat: struct vagy string
Tankor: láncolt lista, fa, dinamikus vektor, ...

Dátum típus létrehozása [3-7]

```
struct Date {
 int y;
 int m;
 int d;
 };

 Date x1;
x1.y=2005;
x1.m=3;
x1.d=15;
```

Adatok konzisztenciájának biztosítása [20-25]

```
Date x2;
x2.y=2000;
x2.m=2;
x2.d= 29; // Ki tudja van-e ilyen;
x2.m=15; // Huh!!

void init_date( Date& date,
                int year, int month, int day )
{
    if( !is_date(year,month,day) ) {
            throw "Hibas datum";
    }
    dd.y=year; dd.m=month; dd.d=day;
}
```

Hibás számítások elkerülése [27-44]

```
Date holnap( Date ma ) {
    Date holnap;
    holnap.y = ma.y;
    holnap.m = ma.m; // valóban ?
    holnap.d = ma.d+1; // Huh!!
    return holnap;
}
```

Hozzunk létre egy jó működő add_day() függvényt.

```
Date next_day( Date ma )
{
    Date holnap=ma;
    add_day( holnap, 1);
    return holnap;
}
```

Egységbe zárás [46-60]

```
struct Date {
    int y;
    int m;
    int d;

    Date(int year, int month, int day)
    {  // ellenőrzi az adatokat
       ...
    };

    void add_day(int number_of_day){ ... }
 };
```

Használat:

```
Date ma(2013,02, 26 );
Date holnap = ma; // Van ilyen?
holnap.add_day(1);
```

Adattagok elrejtése [64-78]

```
ma.m=13; // huh!!

struct Date {
private:
    int y;
    int m;
    int d;
public:
    Date(int year, int month, int day); // ellenőrzi az adatokat
    void add_day(int number_of_day);

    int year_of_date() { return y; }
    int month_of_date() { return m; }
    int day_of_date() { return d; }
 };

Date dd(2013,2,26);
int ev = dd.y; // HIBA !! private
int ev = dd.year_of_date(); // Igy már jó
```

Mekkora a getter függvények overhead-je?

struct vs. class [80-90]

```
struct A{
};
```

$\triangleq$

```
class A{
public:
};
```

```
class A{
};
```

$\triangleq$

```
struct A{
private:
};
```

Viselkedéssel nem rendelkező típusnál struct-ot szokás használni.

Hónap és nap felcserélésének kivédése [94-110]

Hogyan vesszük észre ha Date dd(2013,2,3); helyett Date dd(2013,3,2); írunk?

```
class Date {
public:
  enum Month {  jan=1, feb, mar, apr, may,
                jun, jul, aug, sep, oct, nov, dec };

  Date(int year, Month month, int day); // ellenőrzi az adatokat
  void add_day(int d );
  int year_of_date() {return y; }
  Month month_of_date() {return m; }
  int day_of_date() {return d; }

private:
   int y;
   Month m;
   int d;
};
```

enum használata [111-121]

```
Date::Month m = Date::dec;
int x = m; // OK FONTOS
m = 10;      // Hiba!
m = Date::Month(10); // OK

Date ma( 2013, Date::feb, 26 ); // OK

// Rossz paraméter sorrend.
Date ma( 2013, 26, Date::feb ); // Fordítási hiba

// Újabb problémák
const holnap(2013, Date::feb, 27 );
int ev=holnap.year_of_date(); // Hiba mert nem konstans függvény
Date szabadnapok[30]; // Hiba nincs paraméter nélküli ctor
```

Újabb gondok kiküszöbölése [124-145]

```
class Date {
public:
    enum Month { jan=1, feb, mar, apr, may, jun, jul, aug,
         sep, oct, nov, dec };

    class Invalid{}; // throw-hoz

  Date(int year, Month month, int day);
  Date(); // alapértelmezett konstruktor
  void add_day(int number_of_day );

  int year_of_date() const { return y; }
  Month month_of_date() const { return m; }
  int day_of_date() const { return d; }

private:
  int y;
  Month m;
  int d;
};
```

Tagfüggvények definiálása osztályon kívül [148-175]

```
Date::Date()  {
    static const Date dd(2001,Date::jan,1);
    y=dd.y;     m = dd.m;     d = dd.d;
}

Date::Date(int y, Date::Month month, int day) {
    if (!is_date(y,month,day)) throw Invalid();
    this->y = y; // Paraméter és tagváltozó megkülönböztetése
    m = month;
    d = day;
}

void Date::add_day(int number_of_day) {
    if (number_of_day<0) throw Invalid();
    while (days_in_month(y,m)<number_of_day+d) {
        number_of_day -= days_in_month(y,m);
        if( m == dec ) { m=jan; y++; }
        else m = Month(m+1);
    }
    d += number_of_day;
}
```

Segédfüggvények [175-200]

```
bool leapyear(int year) {
    if (year%4) return false;
    if (year%100==0 && year%400) return false;
    return true;
}

int days_in_month(int year, Date::Month month) {
    switch (month) {
        case Date::feb:         // Februári napok változnak
            return (leapyear(year))?29:28;
        case Date::apr: case Date::jun: case Date::sep: case Date::nov:
            return 30;
        default:
            return 31;
    }
}

bool is_date(int year, Date::Month  month, int day) {
     // Évet nem lehet rosszul megadni
     if (day<=0) return false; // day mindig pozitív
     if (days_in_month(year,month)<day) return false;
     return true;
}
```

Kiíratás és beolvasás [202-220]

```
ostream& operator<<(ostream& os, const Date& d) {
  return os << '(' << d.year_of_date() << ','
            << d.month_of_date() << ',' << d.day_of_date() << ')';
}

istream& operator>>(istream& is, Date& dd) {
    int y, m, d;
    char ch1, ch2, ch3, ch4;
    is >> ch1 >> y >> ch2 >> m >> ch3 >> d >> ch4;
    if (!is) return is;
    if (ch1!='(' || ch2!=',' ||
        ch3!=',' || ch4!=')') { // oops: format error
        is.clear(ios_base::failbit); // set the fail bit
        return is;
    }
    dd = Date(y,Date::Month(m),d);      // update dd
    return is;
}
```

Osztály interfész kialakításának a szempontjai

- Legyen teljes, minden szükséges szolgáltatást biztosítson.
- Legyen minimális. Ami az adatok realizációjától független, az inkább legyen külső függvény.
- Inicializálja az objektumot.
- Támogassa a másolást (következő óra).
- A függvények argumentum típusa olyan legyen, hogy azt jól lehessen ellenőrizni.
- Az interfész szolgálja ki a konstans objektumokat is.
- Szabadítson fel minden lefoglalt erőforrást.