# 8. előadás

Késtem 10 percet, nem hiszem, hogy sok mindenről lemaradtam volna.

## Típusok

A típus absztrakció. Nincs ilyen fogalom számítógépben.

int

**typedef** user**{**

string name**;**

string email**;**

//...

**};**

Komplex számokat többféleképpen lehet ábrázolni. Pl.: valós és képzetes rész.

Régen komplex szám abszolút értékét számoló függvény két paraméterként kapta meg a komplex számot. Ez a két paraméter egybe tartozott, de ez nem működik így sokáig… Két komplex szám összege…

Ekkor jött létre a rekord típus. (Egymáshoz tartozó típusok általában a memóriában is egymás mellé kerülnek.)

|  |  |
| --- | --- |
|  | |
|  |  |

Komplex számokat összeadó függvény ekkor már megy a típussal.

Az objektum orientáltság ennél még tovább megy: Komplex számokhoz tartozó műveletek: Van legalább egy komplex szám paramétere. Típusok mögött nem csak reprezentáció áll, hanem a műveletek is. Osztályok belül megírt tagfüggvények. Ábrázolás + műveletek.

# Példa: dátum

## Tesztprogram

|  |
| --- |
| Tesztprogram |
| #include <iostream>  #include "date.h"  **using** **namespace** std**;**  /\* program a date osztály tesztelésére \*/  int main**(){**  date d**;**  **while** **(** cin **>>** d **){**  cout **<<** **(**d**+=**40**)** **<<** endl**;**  **}**  date d2**(**2004**,**3**);**  **if** **(** d **<** d2 **){++**d**;}**  cout **<<** d **<<** endl**;**  **return** 0**;**  **}** |

Dátum létrehozása:

* 3 paraméterrel: date x**(**2012**,**11**,**12**);**
* 2 paraméterrel: date d2**(**2004**,**3**);**
* 1 paraméterrel: date d3**(**2005**);**
* Szövegből: date d4**(**"2012.11.12"**);**
* Nem adunk meg semmit, ekkor default értékekkel jön létre egy dátum.

Java-ban dátum objektum létrehozásakor nem jön létre semmi, csak miután inicializáltuk is. De C++-ban egyből létrejön, memóriaszeméttel. Nem NULL referencia lesz. Valid objektum a tárterületen.

Dátumok összehasonlítása támogatva van. Dátumok inkrementálása működik, 1-1 nappal lép előre.

## Header fájl

|  |
| --- |
| Header fájl |
| #ifndef DATE\_H  #define DATE\_H  #include <iostream>  /\*  \* egy egyszerû dátum típus  \*  \* egy bonyolulthoz lásd:  \* http://www3.sympatico.ca/date4cpp/index.html  \*  \*/  class date**{**  public**:**  /\* konstruktorok \*/  date**(** int y**=**2000**,** int m**=**1**,** int d**=**1**)** **{** set**(** y**,** m**,** d**);** **}**  date**(** const char **\***s**);**  /\* lekérdezések \*/  int get\_year**()** const **{** **return** year**;** **}**  int get\_month**()** const **{** **return** month**;** **}**  int get\_day**()** const **{** **return** day**;** **}**  /\* módosító mûveletek \*/  date**&** set\_year**(** int y**)** **{** year **=** y**;** **return** **\*this;** **}**  date**&** set\_month**(** int m**)** **{** month **=** m**;** **return** **\*this;** **}**  date**&** set\_day**(** int d**)** **{** day **=** d**;** **return** **\*this;** **}**  date**&** next**();**  date**&** add**(** int n**);**  /\* operátorok \*/  date**&** **operator++()** **{** **return** next**();** **}**  date **operator++(**int**)** **{** date curr**(\*this);** next**();** **return** curr**;** **};**  date**&** **operator+=(** int n**)** **{** **return** add**(**n**);** **}**  /\* beolvasás - kivitel \*/  void get**(** std**::**istream**&** is**);**  void put**(** std**::**ostream**&** os**)** const**;**  private**:**  /\* segéd-függvények \*/  void set**(** int y**,** int m**,** int d**){**  /\* TODO: ellenõrzések \*/  year **=** y**;** month **=** m**;** day **=** d**;**  **}**  /\* adattagok \*/  int year**;**  int month**;**  int day**;**  **};**  /\* globális mûveletek az osztályon \*/  bool **operator<(** date d1**,** date d2**);**  inline bool **operator==(** date d1**,** date d2**)** **{** **return** **!(**d1**<**d2 **||** d2**<**d1**);** **}**  inline bool **operator!=(** date d1**,** date d2**)** **{** **return** d1**<**d2 **||** d2**<**d1**;** **}**  inline bool **operator<=(** date d1**,** date d2**)** **{** **return** **!(**d2**<**d1**);** **}**  inline bool **operator>=(** date d1**,** date d2**)** **{** **return** **!(**d1**<**d2**);** **}**  inline bool **operator>(** date d1**,** date d2**)** **{** **return** d2**<**d1**;** **}**  /\* beolvasó és kiíró operátor \*/  std**::**istream**&** **operator>>(** std**::**istream**&** is**,** date**&** d**);**  std**::**ostream**&** **operator<<(** std**::**ostream**&** os**,** const date **&**d**);**  #endif /\* DATE\_H \*/ |

Header fájl célja: Fordító lássa, hány bájtos egy-egy dátum. Ha ezt nem tudná, nem fordulna a program.

Itt is fontos az include-guard. #ifndef DATE\_H #define DATE\_H /\* ... \*/ #endif

Tagfüggvények meghívása: d**.**get\_day**();**

## Konstruktor

date**(** int y**=**2000**,** int m**=**1**,** int d**=**1**)** **{** set**(** y**,** m**,** d**);** **}**

date**(** const char **\***s**);**

Nincs visszatérési értéke. Felsoroljuk, milyen paraméterekkel hívható meg a konstruktor. (Mi az a minimális információ, ami szükséges egy objektum létrehozásához.)

Ha adunk egy paraméter után default értéket, akkor azt nem kötelező megadni létrehozáskor.

Ha egy paramétert sem adunk meg, akkor úgynevezett default konstruált objektumot kapunk.

A konstruktor túlterhelhető. Ugyan azzal a névvel, de más paraméter szignatúrával működik.

A konstruktort nem kell egyből a header​ben implementálni. De ha egyből implementáljuk, lesz. (Már ha nem túl bonyolult a törzse.)

Konstruktorhívás paraméter nélkül nem lehet date d**();**, az függvény deklaráció lenne. Csak date d**;** lehet.

## Lekérdezések ( kulcsszóval)

int get\_day**()** const **{** **return** day**;}**

Ezzel jelezzük, hogy az objektum nem változik meg a függvényhívás eredményeképp. Tehát meghívható konstans (és nem konstans) objektumon is.

Nem csak jelezzük, a fordító ellenőrzi is, tényleg betartottuk-e ezt a nem módosítjuk szabályt.

## Módosító műveletek

date**&** set\_day**(**int d**)** **{**day **=** d**;** **return** **\*this;}**

Rejtett paraméter: **this** (pointer arra az objektumra, amin keresztül a függvényt meghívtuk.)

Vissza is adjuk, referencia szerint, hogy lehessen d2**=**d**.**set\_day**(**1**);**-t csinálni. (**\*this** a mutatott teljes objektum.)

Továbbá ez egy álnév -re (date**&**). Újra és újra ehet rajta műveleteket hívni: d**.**set\_day**(**1**).**set\_month**(**1**);**

Továbbá nem másolat jön létre.

## Inkrementáló operátorok

date**&** **operator++()** **{** **return** next**();** **}**

date **operator++(**int**)** **{** date curr**(\*this);** next**();** **return** curr**;** **};**

Visszatérési érték típus nem ugyan az. Miért? pre- és postfix operátorok. Melyik melyik?

Aminek nem adunk meg paramétert, az a prefix: **++**d**;**. A paraméteres a postfix: d**++;**

Prefix sokkal hatékonyabb, mint a postfix.

Az átadott paraméter mindig , csak arra kell, hogy meg tudja különböztetni a compiler a pre- és postfix operátort.

A postfix működése költségesebb. Létrehoz egy ideiglenes dátumot (másolat), megnöveli az eredetit, majd visszaadja a meg nem növelt másolatot.

## Növelő operátor

date**&** **operator+=(** int n**)** **{** **return** add**(**n**);** **}**

A prefix inkrementálással van szinkronban.

## Beolvasás, kiírás

void get**(** std**::**istream**&** is**);**

void put**(** std**::**ostream**&** os**)** const**;**

-eket kötelező referencia szerint átadni.

## Globális műveletek az osztályon

Egyszer megírjuk a "kisebb" operátort:

bool **operator<(** date d1**,** date d2**);**

Ez volt a lényeg. Ennek segítségével megírjuk a többit. Előnye: Nem követünk el hibát. Tranzitivitás, szimmetria megmarad. Nem kell másolni ugyan azt a kódot többször.

inline bool **operator==(** date d1**,** date d2**)** **{** **return** **!(**d1**<**d2 **||** d2**<**d1**);** **}**

inline bool **operator!=(** date d1**,** date d2**)** **{** **return** d1**<**d2 **||** d2**<**d1**;** **}**

inline bool **operator<=(** date d1**,** date d2**)** **{** **return** **!(**d2**<**d1**);** **}**

inline bool **operator>=(** date d1**,** date d2**)** **{** **return** **!(**d1**<**d2**);** **}**

inline bool **operator>(** date d1**,** date d2**)** **{** **return** d2**<**d1**;** **}**

Miért nem tagfüggvények? Hogy minkét paraméteren ugyan az a konverziós szabály legyen érvényes.

Konstruktoron keresztül minden (például integer) konvertálódik dátummá, ami tud: **if(**a**<**2013**)**//működik, 2013 dátum lesz

Ha tagfüggvény lenne: a**.operator==(**b**);**//működik

a**.operator<(**2013**);**//működik, automatikus konverzió. Ideiglenes dátum objektum létrejön a konstruktoron keresztül

2013**.operator>(**a**);**//Nem működik! Nincs paraméterkonverzió this-en keresztül.

Az összehasonlításokat általában globális műveletként valósítjuk meg.

## Implementáció

|  |
| --- |
| Implementáció |
| #include <iostream>  #include <cstring>  #include <cstdlib>  #include "date.h"  **using** **namespace** std**;**  /\* konstruktor \*/  date**::**date**(** const char **\***s**){**  /\* strchr és strrchr a <cstring>-bõl \*/  char **\***p1 **=** strchr**(** s**,** '.'**);**  char **\***p2 **=** strrchr**(** s**,** '.'**);**  **if** **(** p1 **&&** p2 **&&** p1 **!=** p2 **)**  **{**  /\* atoi a <cstdlib>-bõl> \*/  int y **=** atoi**(**s**);**  int m **=** atoi**(**p1**+**1**);**  int d **=** atoi**(**p2**+**1**);**  set**(** y**,** m**,** d**);**  **}**  **}**  /\* módosítók \*/  date**&** date**::**next**(){**  static int day\_in\_month**[]** **=**  **{** 31**,** 28**,** 31**,** 30**,** 31**,** 30**,** 31**,** 31**,** 30**,** 31**,** 30**,** 31 **};**  **++**day**;**  /\* TODO: szökõév \*/  **if** **(** day**-**1 **==** day\_in\_month**[**month**-**1**])**  **{**  day **=** 1**;**  **++**month**;**  **}**  **if** **(** 13 **==** month **)**  **{**  month **=** 1**;**  **++**year**;**  **}**  **return** **\*this;**  **}**  date**&** date**::**add**(** int n**){**  **for** **(**int i **=** 0**;** i **<** n**;** **++**i**)**  next**();**  **return** **\*this;**  **}**  /\* input-output \*/  void date**::**get**(** std**::**istream**&** is**){**  // nem lenne igazán jó:  // is >> year >> month >> day;  int y**,** m**,** d**;**  **if** **(** is **>>** y **>>** m **>>** d **)**  **{**  set**(** y**,** m**,** d**);**  **}**  **}**  void date**::**put**(** std**::**ostream**&** os**)** const**{**  // nem lenne igazán jó: os << year << month << day;  os **<<** "[ " **<<** get\_year**()** **<<** "," **<<** get\_month**()**  **<<** "," **<<** get\_day**()** **<<** " ]"**;**  **}**  /\* globális mûveletek \*/  bool **operator<(** date d1**,** date d2**){**  **return** d1**.**get\_year**()** **<** d2**.**get\_year**()** **||**  d1**.**get\_year**()** **==** d2**.**get\_year**()** **&&** d1**.**get\_month**()** **<** d2**.**get\_month**()** **||**  d1**.**get\_year**()** **==** d2**.**get\_year**()** **&&** d1**.**get\_month**()** **==** d2**.**get\_month**()**  **&&** d1**.**get\_day**()** **<** d2**.**get\_day**();**  **}**  istream**&** **operator>>(** istream**&** is**,** date**&** d**){**  d**.**get**(** is**);**  **return** is**;**  **}**  ostream**&** **operator<<(** ostream**&** os**,** const date **&**d**){**  d**.**put**(** os**);**  **return** os**;**  **}** |

Előadás vége.