# 9. gyakorlat

## Még 4 gyakorlat

Utána ZH

## Pluszminusz

0 lett, 1 db -om van eddig.

## Konstans objektumok

const MyVector copied **=** v**;**

copied**.**prettyPrint**();**//hiba

Konstans objektumon csak konstans tagfüggvény (member) hívható.

Ezért érdemes minden tagfüggvényt konstansként deklarálni, ami nem változtatja az objektumot:

void prettyPrint**()** **const;**//headerbe

void MyVector**::**prettyPrint**()** **const {**//cpp-be

**for(**int i**=**0**;**i**<**\_size**;++**i**){** std**::**cout**<<**\_items**[**i**]<<**", "**;}**

std**::**cout**<<**\_size**<<**"items printed"**<<**std**::**endl**;**

std**::**cout**<<**"capacity is "**<<**\_capacity**<<**std**::**endl**;**

**}**

ZH-ban lesz ilyen konstans objektum felvéve, ketteshez is kell.

A fordító ezt enforce​olja. Konstans member​en belül nem lehet módosítani a reprezentációt.

const MyVector**\*** const **this** //a rejtett paraméter

### Túlterhelés, ha mindkét verzió kell

int**&** at**(**int index**);**

int at**(**int index**)** const**;**

Egyik verzió referenciát ad vissza:

int**&** MyVector**::**at**(**int index**){**

**return** \_items**[**index**];**

**}**

Másik csak egy másolatot ad vissza:

int MyVector**::**at**(**int index**)** const**{**

**return** \_items**[**index**];**

**}**

## Operátorok

Speciális alakban hívható függvények.

Miket szoktunk felülírni:

* indexelés
* kiírás és beolvasás
* függvényhívás zárójele (functor)
* vektorba tegyük bele

## Példa: [\_] indexelés

int **operator[](**int index**)** const**;**

int**&** **operator[](**int index**);**

int MyVector**::operator[](**int index**)** const**{**

**return** at**(**index**);**

**}**

int**&** MyVector**::operator[](**int index**){**

**return** at**(**index**);**

**}**

## Példa: kiírás

Első próbálkozásunk csak v2**.operator<<(**cout**);** formában működik. Ezt cout**<<**v2**;**-re javítva:

* Legyen globális függvény.
  + De ekkor a reprezentáció minden típusát globálissá kellene tenni. Ez nem elfogadható.
  + Vagy getter és setter mindenre. Ez meg olyan Java-s.
* friend

### Végső megoldás

void **operator<<(**std**::**ostream**&** os**,** MyVector**&** v**){**

std**::**copy**(**v**.**\_items**,**v**.**\_items**+**v**.**\_size**,**std**::**ostream\_iterator**<**int**>(**os**,**", "**));**

vagy:

for(int i=0;i<v.\_size;++i){

os<<v.\_items[i]<<", ";

}

**}**

Copy paraméterei: eleje, vége, eredmény.

Eredménynek egy iterátort adtunk meg.

hf: std::back\_inserter segítségével beolvasást hasonlóan megírni. (Méretre ellenőrzéssel.)

## Első template, amit látunk: copy

template**<**class InputIterator**,** class OutputIterator**>**

OutputIterator copy **(** InputIterator first**,** InputIterator last**,** OutputIterator result **){**

**while** **(**first**!=**last**)** **\***result**++** **=** **\***first**++;**

**return** result**;**

**}**

**\***result**++** kiolvasása: **\*(**result**++)**

Növeli, visszaadja az eredetit, lekérdezi az eredeti címét.

# Sablonok (template)

Lehet írni függvényeket, osztályokat.

Nem fordítandó kód. Nem is -be rakjuk, hanem -ba.

A fordító gyárt belőle kódot, amikor használjuk egy példányát. Például ha int-re és double-ra használtuk, akkor két példányt gyárt belőle a fordító. Egy int és egy double verziót.

Függvényekre lehet túlterhelni nem template függvénnyel, class-ra nem.

class helyett typename is lehet, ugyan azt jelenti. Kifejezőbb, de hosszabb leírni.

## Template paraméter lehet

### Konstans

template **<**class T**,** int size**>**

class Vector**{**

//...

**}**

//Példányosítás:

Vector**<**int**,** 10**>** v**;**

Vector**<**char**,** 20**>** v2**;** //Más a típusa, mint v-nek.

Vector**<**int**,** 20**-**10**>** v3**;** //Konstans kifejezést természetesen kiértékeli, ezért v és v3 típusa ekvivalens.

### Külső szerkesztésű objektum vagy függvény címe

(?) Skalárok is lehetnek a paraméterek. (?)

### Nem túlterhelt tagra mutató pointer

(nem fogjuk venni)

## Max

Megírni minden típusra nem jó. Ha új típust vezetünk be, arra is meg kellene írni. Ugyan az a függvény törzse, csak a típusok mások.

Define makró sem jó, láttuk miért. Ha agyon is zárójelezzük. Pl.: x**++**-t adunk neki azt kétszer is megnöveli.

## Max template

template **<**class T**>** const T**&** max **(** const T**&** a**,** const T**&** b **)** **{**

**return** **(**a**<**b**)?**b**:**a**;**  // or: return comp(a,b)?b:a; for the comp version

**}**

## Swap template

template **<**class T**>** void swap **(** T**&** a**,** T**&** b **){**

T c**(**a**);** a**=**b**;** b**=**c**;**

**}**

## Default értéket lehet adni template paramétereknek is

template **<**class T**,** int s**=**100**>**

Gyakorlat vége.