# 7. gyakorlat

## Jegyzeteket ne tegyük fel a netre

Sok a rossz ZH megoldás, félreértett magyarázat a tanulók honlapjain. Pongyolán megfogalmazott dolgokat tényként kezeljük, ha le van írva.

Ezt nem tudják kitörölni a tanulók honlapjairól, nem is fogják, de kurzusfórumról ki lett bannolva (!?) minden ilyen link.

Gyakorlatvezető sem teheti fel az STR segédanyagait, le is vetették vele.

Én azért is felteszem az enyémet. Használd fenntartással, kedves olvasó!

## Múlt órai procedurális bohóckodás folytatása

const int CAPACITY **=** 10**;**

//Reprezentáció:

int**\*** v**;**

int size**;**

int curr**;**

//(Felmerül a kérdés, hogy vajon tényleg hívatjuk-e reprezentációnak.)

void prettyPrint**(){**

 **for(**int i**=**0**;**i**<**size**;++**i**){**

 std**::**cout**<<**v**[**i**]<<**", "**;**

 **}**

**}**

void init**(**int \_size**){**

 size**=**\_size**;**

 curr**=**0**;**

 v**=new** int**[**size**];**

 **for(**int i**=**0**;**i**<**size**;++**i**){**

 v**[**i**]=**0**;**

 **}**

**}**

int at**(**int index**){**

 //ToDo: index validation

 **return** v**[**index**];**

**}**

void push\_back**(**int e**){**

 //ToDo: CAPACITY check

 v**[**curr**++]=**e**;**//Mivel a postfix ++ művelet úgy működik, hogy lemásolja, az eredetit megnöveli, de a másolatot adja vissza. Tehát 0-s pozícióra is kerül érték, helyesen.

**}**

int main**(){**

 init**(**CAPACITY**);**

 prettyPrint**()**

 push\_back**(**4**);**

 push\_back**(**5**);**

 prettyPrint**()**

**}**

Probléma:

Nem lehet példányosítani, kétszer használni. Csak egy vektor szimulálására alkalmas a program. (Ha minden függvénynek átadjuk a reprezentációt (azt a 3 változót), az bonyolult.)

## Struktúra

C-ben szabvány előtt volt limitált struktúra struct adattagok összefogására.

1. generáció: globális változókon dolgozik a függvény. Azon túl, hogy a kódban egymás mellé írjuk a változókat, nem tudjuk erősebben összefogni.

2. generáció: struktúrában összefoglalhatjuk a változókat:

struct rep**{**

 int**\*** v**;**

 int size**;**

 int curr**;**

**};**//Kell mögé pontosvessző

void init**(**rep r**,** int \_size**){**

 r**.**size**=**\_size**;**

 r**.**curr**=**0**;**

 //...

**}**

int main**(){**

 rep r**;**

 init**(**r**,**CAPACITY**);**

 //...

**}**

Probléma: Tagfüggvények nem voltak ezeken a struktúrákon. Minden függvénynek meg kell adni, melyik struktúrán dolgozzon.

Objektum orientált paradigmának az az értelme/lényege, hogy a reprezentációt beburkoljuk egy interfésszel. Csak az interfészen keresztül érhető el a belső reprezentációs rész.

## OOP három alappillére

Enkapszuláció

Öröklődés

Polimorfizmus

## Osztály (Class)

class vagy struct ugyan annyit tud. Egy különbség az alapértelmezett elérhetőség: C $struct$-ja publikus, C++ $class$-ja privát. És az öröklődésnél ugyan e szabály miatt lesz valami.

Típusobjektum létrehozás konstruktor műveletekkel.

Kölcsönös állomány befoglalás ellen include guard ("include őrszem"). (?Ezt minek ismételjük?)

$using namespace std;$ ZH-n és egyébként is tilos, mert elrontja a fájlokat, amik include-olják ezt a headert. (?Ezt minek ismételjük?)

Típus alapértelmezett műveletei:

* Rule of three:
Ezeket mindig felül kell írni, ha (valami).
	+ Default konstruktor
	+ Destruktor
	+ Copy konstruktor
* Továbbá:
	+ Értékadás operátor
	+ Címképző operátorok.

class myVector**{**

private**:** //Ez a default, nem kell/szokás kiírni.

 int \_size**;** //Az adattagok előtti aláhúzás (\_) jel konvenció.

 int \_capacity**;**

 int \_curr**;**

 int**\*** \_items**;**

**};** esetén $4\*4$ byte$=16$ bájtos objektumot hoz létre a myVector m**;** deklaráció.

De az m**.**size**;** m**.**curr**;** enkapszulált reprezentáció tagok nem érhetőek el.

Elérhetőségi specifikátor (hiding): public**,** protected**,** private kulcsszavak. Alapértelmezett a private.

Típus egyúttal névteret is definiál. MyVector**::**\_size

Konstruktor: nincs visszatérési érték, (valami, valami).

Vagy a header​ben írjuk meg (de ekkor kötelezően inline lesz)

Saját típuson belüli tagfüggvények látják az aktuális objektum reprezentációját. Nem kell MyVector**::**\_size.

Destruktor: memory leak elkerülése végett felszabadítjuk (**delete**) a tömböket, stb. Akkor van szükség destruktor​ra, ha van ilyen felszabadítandó tagunk.

Pointeren keresztüli elérés: **(\***p**).**prettyPrint**();** helyett p**->**prettyPrint**();** Szebb.

## Teljes kód eddig

|  |
| --- |
| //MyVector.h#ifndef MyVector\_H //ZH-n kötelező az include guard használata#define MyVector\_H//using namespace std; //ZH-n és egyébként is tilos, mert elrontja a fájlokat, amik include-olják ezt a headert. (?Ezt minek ismételjük?)class MyVector**{**private**:** //Ez a default, nem kell/szokás kiírni. int \_size**;** //Az adattagok előtti aláhúzás (\_) jel konvenció. int \_capacity**;** int \_curr**;** int**\*** \_items**;**public**:** MyVector**();** **~**MyVector**();**  void prettyPrint**();** void init**(**int \_size**);** int at**(**int index**);** void push\_back**(**int e**);****};**#endif //MyVector\_H//MyVector.cpp#incldue "myVector.h"#incldue <iostream>MyVector**::**MyVector**(){** \_curr**=**0**;** \_size**=**10**;** \_capacity**=**100**;** \_items**=new** int**[**\_size**];** **for(**int i**=**0**;**i**<**\_size**;++**i**){** \_items**[**i**]=**0**;** **}****}**MyVector**::~**MyVector**(){** **delete[]** \_items**;** \_items**=**0**;**//Extra biztonság, konvenció**}**//Saját típuson belüli tagfüggvények látják az aktuális objektum reprezentációját.void MyVector**::**prettyPrint**(){** **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**;****}**//main.cpp#include "MyVector.h"int main**(){** MyVector m**;** //Default konstruktor hívódik. MyVector v**(**20**,** 120**);** MyVector v2**(**20**);** m**.**prettyPrint**();** v**.**prettyPrint**();** v2**.**prettyPrint**();**  v**.**push\_back**(**4**);** v**.**push\_back**(**5**);** v**.**push\_back**(**6**);** v**.**push\_back**(**7**);****}** |

Gyakorlat vége.