# 11. gyakorlat

## ZH példa - Asszociatív mátrix

|  |
| --- |
|  |
| #include <iostream>#include "amatrix.h"#include <string>#include <algorithm>struct Subjects**{** int max**;** Subjects**():**max**(**0**)** **{}** void **operator()(**const std**::**pair**<**char**,** int**>&** p**){** std**::**cout **<<** ' ' **<<** p**.**first **<<** ":" **<<** p**.**second **<<** std**::**endl**;** **if** **(**p**.**second **>** max**)** max **=** p**.**second**;** **}****};**struct Print**{** int best**;** Print**():**best**(**0**)** **{}** template **<**class Element**>** void **operator()(**const Element**&** e**){** std**::**cout **<<** e**.**first **<<** ":" **<<** std**::**endl**;** int i **=** std**::**for\_each**(**e**.**second**.**begin**(),** e**.**second**.**end**(),** Subjects**()).**max**;** **if** **(**i **>** best**)** best **=** i**;** **}****};**const int max **=** 10000**;**int main**(){** int yourMark **=** 1**;** // 2-es AssocMatrix**<**std**::**string**,** char**,** int**>** my\_marks**;** my\_marks**.**insert**(**"Analizis"**,** 'g'**,** 4**);** my\_marks**.**insert**(**"Analizis"**,** 'e'**,** 5**);** my\_marks**.**insert**(**"C++"**,** 'g'**,** 2**);** const AssocMatrix**<**std**::**string**,** char**,** int**>** cmarks **=** my\_marks**;** AssocMatrix**<**int**,** int**,** double**>** test**;** **for(**int i **=** 0**;** i **<** max**;** **++**i**)** test**.**insert**(**i**,** i**,** 5.2**);** yourMark **=** cmarks**.**rows**();** // 3-as AssocMatrix**<**std**::**string**,** char**,** int**>** test2**;** **for(**int i **=** 0**;** i **<=** max**;** **++**i**)** test2**.**insert**(**"Testing"**,** 'T'**,** i**);** **if** **(**4 **==** cmarks**.**at**(**"Analizis"**,** 'g'**)** **&&** max **==** test2**.**at**(**"Testing"**,** 'T'**)){** **++**my\_marks**(**"C++"**,**'g'**);** yourMark **=** test2**.**rows**()** **+** my\_marks**.**rows**();** **}** // 4-es **if** **(**2 **==** cmarks**(**"C++"**,**'g'**))** yourMark **=** cmarks**.**cols**()** **+** my\_marks**.**cols**();** // 5-os yourMark **=** std**::**for\_each**(**cmarks**.**begin**(),** cmarks**.**end**(),** Print**()).**best**;** std**::**endl**(**std**::**cout**);** std**::**cout **<<** "Your mark is " **<<** yourMark **<<** std**::**endl**;****}** |

## Megoldás

|  |
| --- |
| #ifndef ASSOC\_MATRIX\_\_H#define ASSOC\_MATRIX\_\_H#include <map>template **<**class Key1**,** class Key2**,** class T**>**class AssocMatrix**{** std**::**map**<**Key1**,** std**::**map**<**Key2**,** T**>** **>** matrix**;**public**:** const T**&** at**(**const Key1**&** k1**,** const Key2**&** k2**)** const**{** **return** **(**matrix**.**find**(**k1**)->**second**).**find**(**k2**)->**second**;** **}** **typedef** typename std**::**map**<**Key1**,** std**::**map**<**Key2**,** T**>** **>::**const\_iterator const\_iterator**;** void insert**(**const Key1**&** k1**,** const Key2**&** k2**,** const T**&** t**){** matrix**[**k1**][**k2**]** **=** t**;** **}** const\_iterator begin**()** const**{** **return** matrix**.**begin**();** **}** const\_iterator end**()** const**{** **return** matrix**.**end**();** **}** const T**&** **operator()(**const Key1**&** k1**,** const Key2**&** k2**)** const**{** **return** at**(**k1**,** k2**);** **}** T**&** **operator()(**const Key1**&** k1**,** const Key2**&** k2**){** **return** **(**matrix**.**find**(**k1**)->**second**).**find**(**k2**)->**second**;** **}** int rows**()** const**{** **return** matrix**.**size**();** **}** int cols**()** const**{** unsigned int i **=** 0**;** **for(**const\_iterator ci **=** matrix**.**begin**();** ci**!=**matrix**.**end**();++**ci**){** **if** **(**ci**->**second**.**size**()** **>** i**)** i **=** ci**->**second**.**size**();** **}** **return** i**;** **}****};**#endif |

## Megjegyzések

Aki egyből ugrik a kettesre, az csak később veszi észre, hogy -et kellett volna használni. El lehet jutni négyesig vektorokkal, de ötösért át kell írni a programot mátrixosra. És amúgy is kényelmesebb a .

A indexelő operátora (std**::**map**::operator[]**) nem panaszkodik nem létező elemek miatt, de nem is ad vissza memóriaszemetet. Ha nem létező elemre indexelünk rá, automatikusan létrehozza.

Nem kell saját copy konstruktor, mert nincs pointer adattagunk. ZH-n lehet, hogy lesz.

A legtöbb tagfüggvény egysoros. Nem kell sok, csak ismerni kell az STL-t.

## Megjegyzések kettesért

Látszik, hogy konstans mátrixon hívjuk a függvényt. Ebből tudjuk, hogy konstansnak kell deklarálni. Kettesért ilyen biztos lesz.

Stressz teszt. Kettesért is kell -res mátrixot bírnia a reprezentációnknak.

Plusz direkt -nak nevezték el a változót, hogy még véletlen se tudjunk: **using** **namespace** std**;**-t használni a header​ben.

## Megjegyzések hármasért

Konstans funkciót kell írni. Referencia szerint adja vissza a típusú elemet.

A -nek van saját tagfüggvénye. Iterátort ad vissza. Ha ezt csillaggal dereferáljuk, akkor egy pár () van mögötte. Ennek a párnak most mindkétszer a második eleme kell.

**return** **(**matrix**.**find**(**k1**)->**second**).**find**(**k2**)->**second**;**

A **++**my\_marks**(**"C++"**,**'g'**);** kódrészlet sok gondot okozott. Nem tudták a tanulók, mit vár tőlük a program. (Szöveges leírás a feladathoz nincs, nem is lesz.) Operátor precedencia alapján helyesen kell kiolvasni, hogy ez a kódrészlet növeli a C++ gyakorlati jegyünket. Tehát a zárójel operátort implementálni. (Nem a **++**-t.) Olyan, mint az at, de referencia szerint kell visszaadni, hogy menjen rá az inkrementáló operátor.

T**&** **operator()(**const Key1**&** k1**,** const Key2**&** k2**){**

 **return** **(**matrix**.**find**(**k1**)->**second**).**find**(**k2**)->**second**;**

**}**

## Megjegyzések négyesért

Konstans **operator()**-et kell írni. Túlterheléssel:

const T**&** **operator()(**const Key1**&** k1**,** const Key2**&** k2**)** const**{**

 **return** at**(**k1**,** k2**);**

**}**

Meg maximumkeresést (oszlopok száma melyik sorban maximális):

int cols**()** const**{**

 unsigned int i **=** 0**;**

 **for(**const\_iterator ci **=** matrix**.**begin**();** ci**!=**matrix**.**end**();++**ci**){**

 **if** **(**ci**->**second**.**size**()** **>** i**)**

 i **=** ci**->**second**.**size**();**

 **}**

 **return** i**;**

**}**

Az iterátort **typedef**feljük const\_iterator-ra: **typedef** typename std**::**map**<**Key1**,** std**::**map**<**Key2**,** T**>** **>::**const\_iterator const\_iterator**;**

Mert nem akarjuk kiírni mindig, hogy: typename std**::**map**<**Key1**,** std**::**map**<**Key2**,** T**>** **>::**const\_iterator

Azért kell a typename kulcsszó, mert egyébként nem tudja eldönteni a fordító, hogy statikus mező vagy típusnév.

## Megjegyzések ötösért

Bonyolultnak néz ki, de nem az. Be akarja járni az egész mátrixot. Közben kiírja és megkeresi a legjobb eredményt. Csak ez kell hozzá:

const\_iterator begin**()** const**{**

 **return** matrix**.**begin**();**

**}**

const\_iterator end**()** const**{**

 **return** matrix**.**end**();**

**}**

A dupla functort már megírták nekünk. Következő órán részletesebben megnézzük -t.

Gyakorlat vége.

## Update: Következő órán még szó volt erről

Miért jobb a functor, mint egy függvény? Állapota lehet. Jelen esetben a :

struct Subjects**{**

 int max**;**

 Subjects**():**max**(**0**)** **{}**

 void **operator()(**const std**::**pair**<**char**,** int**>&** p**){**

 std**::**cout **<<** ' ' **<<** p**.**first **<<** ":" **<<** p**.**second **<<** std**::**endl**;**

 **if** **(**p**.**second **>** max**)**

 max **=** p**.**second**;**

 **}**

**};**

Update vége.