#include <iostream>

#include <string>

#include <stdlib.h>

using namespace std**;**

int main**()**

**{**

int**\*** array **=** **(**int**\*)**malloc**(**10 **\*** **sizeof(**int**));**

int i**;**

**for** **(**i **=** 0**;** i **<** 10**;** i**++)**

array**[**i**]** **=** i**;**

**for** **(**i **=** 0**;** i **<** 10**;** i**++)**

cout **<<** array**[**i**];**

cout **<<** endl**;**

//free(array + 9); // Crapa pentru ca nu este un pointer valid din heap.

cout **<<** endl**;**

**for** **(**i **=** 0**;** i **<** 10**;** i**++)**

cout **<<** array**[**i**];**

cout **<<** endl**;**

free**(**array**);**

cout **<<** endl**;**

**for** **(**i **=** 0**;** i **<** 10**;** i**++)**

cout **<<** array**[**i**];**

cout **<<** endl**;**

free**(**array**);** // Crapa pentru ca nu avem voie sa dezalocam de doua ori aceeasi zona de memorie. Undefined behaviour. M

cout **<<** endl**;** // Solutie: Marcati cu NULL ce ati dezalocat si nu faceti free decat daca e diferit de NULL. i.e.| free(array); array=NULL; |

**}**

Nu putem dezaloca partial, nici cu delete, nici cu free. Free poate dezaloca doar un pointer alocat explicit cu malloc(). Chemarea functiei free pe o parte dintr-un astfel de pointer nu va produce rezultate.

memmove**(**array**,** array **+** 3**,** 7**);**

int **\***new\_array **=** realloc**(**array**,** 7 **\*** **sizeof(**int**));**

array **=** new\_array**;**

* Incapsularea datelor: private, public, protected. Metodele si membrii Private/protected nu pot fi accesati din afara clasei. Diferenta intre private si protected se observa la mostenire.
* Singura diferenta reala intre clase si structuri este faptul ca implicit membrii si metodele unei clase sunt private, pe cand cei dintr-o structura sunt publici. De asemenea, mostenirea implicita este publica la structuri si privata la clase.

#include <iostream>

// Clase si obiecte

using namespace std**;**

class Box

**{**

public**:**

double length**;** // Length of a box

double breadth**;** // Breadth of a box

double height**;** // Height of a box

**};**

int main**()**

**{**

Box Box1**;** // Declare Box1 of type Box

Box Box2**;** // Declare Box2 of type Box

double volume **=** 0.0**;** // Store the volume of a box here

// box 1 specification

Box1**.**height **=** 5.0**;**

Box1**.**length **=** 6.0**;**

Box1**.**breadth **=** 7.0**;**

// box 2 specification

Box2**.**height **=** 10.0**;**

Box2**.**length **=** 12.0**;**

Box2**.**breadth **=** 13.0**;**

// volume of box 1

volume **=** Box1**.**height **\*** Box1**.**length **\*** Box1**.**breadth**;**

cout **<<** "Volume of Box1 : " **<<** volume **<<**endl**;**

// volume of box 2

volume **=** Box2**.**height **\*** Box2**.**length **\*** Box2**.**breadth**;**

cout **<<** "Volume of Box2 : " **<<** volume **<<**endl**;**

**return** 0**;**

**}**

// Functii membre (metode), definite in fara clasei.

class Box

**{**

public**:**

double length**;** // Length of a box

double breadth**;** // Breadth of a box

double height**;** // Height of a box

double getVolume**(**void**);**

**};**

double Box**::**getVolume**(**void**)**

**{**

**return** length **\*** breadth **\*** height**;**

**}**

// Functii membre (metode), definite inauntrul clasei.

class Box

{

public:

double length; // Length of a box

double breadth; // Breadth of a box

double height; // Height of a box

double getVolume(void)

{

return length \* breadth \* height;

}

}

// Functii membre const.

class Box

**{**

public**:**

double length**;** // Length of a box

double breadth**;** // Breadth of a box

double height**;** // Height of a box

double getVolume**(**void**) const;** //const nu lasa in interiorul functiei sa modifici niciunul din membrii clasei

**};**

double Box**::**getVolume**(**void**)** const

**{**

//aici nu putem scrie length=5 -> eroare (din cauza const)

**return** length **\*** breadth **\*** height**;**

**}**

OBS: In header (.h) punem clasa cu componentele, in .cpp definim functiile din clasa.

#include <iostream>

// Incapsularea datelor.

using namespace std**;**

class Box

**{**

double length**;** // Length of a box

double breadth**;** // Breadth of a box

double height**;** // Height of a box

public**:**

// Member functions declaration

double getVolume**(**void**);**

void setLength**(**double len**);**

void setBreadth**(**double bre**);**

void setHeight**(**double hei**);**

**};**

// Member functions definitions

double Box**::**getVolume**(**void**)**

**{**

**return** length **\*** breadth **\*** height**;**

**}**

void Box**::**setLength**(**double len**)**

**{**

length **=** len**;**

**}**

void Box**::**setBreadth**(**double bre**)**

**{**

breadth **=** bre**;**

**}**

void Box**::**setHeight**(**double hei**)**

**{**

height **=** hei**;**

**}**

// Main function for the program

int main**()**

**{**

Box Box1**;** // Declare Box1 of type Box

Box Box2**;** // Declare Box2 of type Box

double volume **=** 0.0**;** // Store the volume of a box here

// box 1 specification

Box1**.**setLength**(**6.0**);**

Box1**.**setBreadth**(**7.0**);**

Box1**.**setHeight**(**5.0**);**

// box 2 specification

Box2**.**setLength**(**12.0**);**

Box2**.**setBreadth**(**13.0**);**

Box2**.**setHeight**(**10.0**);**

// volume of box 1

volume **=** Box1**.**getVolume**();**

cout **<<** "Volume of Box1 : " **<<** volume **<<** endl**;**

// volume of box 2

volume **=** Box2**.**getVolume**();**

cout **<<** "Volume of Box2 : " **<<** volume **<<** endl**;**

**return** 0**;**

**}**

#include <iostream>

// Constructorul implicit

using namespace std**;**

class Line

**{**

public**:**

void setLength**(**double len**);**

double getLength**(**void**);**

Line**();** // This is the constructor

private**:**

double length**;**

**};**

// Member functions definitions including constructor

Line**::**Line**(**void**)**

**{**

cout **<<** "Object is being created" **<<** endl**;**

**}**

void Line**::**setLength**(**double len**)**

**{**

length **=** len**;**

**}**

double Line**::**getLength**(**void**)**

**{**

**return** length**;**

**}**

// Main function for the program

int main**()**

**{**

Line line**;** //Automat se apeleaza constructorul

// set line length

line**.**setLength**(**6.0**);**

cout **<<** "Length of line : " **<<** line**.**getLength**()** **<<**endl**;**

**return** 0**;**

**}**

**// Inlocuiti Line line; cu Line\* line = new Line();**

int main()

{

Line \*line=new Line(); //Automat se apeleaza constructorul

//acum este pointer, deci folosim ->, nu .

line->setLength(6.0); //SAU (\*line).setLength

cout **<<** "Length of line : " **<<** line**.**getLength**()** **<<**endl**;**

delete(line);

**return** 0**;**

}

#include <iostream>

// Constructor cu parametri

using namespace std**;**

class Line

**{**

public**:**

void setLength**(** double len **);**

double getLength**(** void **);**

Line**(**double len**);** // This is the constructor

Line();

private**:**

double length**;**

**};**

// Member functions definitions including constructor

Line**::**Line**(**double len**)**

**{**

cout **<<** "Object is being created, length = " **<<** len **<<** endl**;**

length **=** len**;**

**}**

Line**::**Line**()**

**{**

cout **<<** "ceapa " **<<** endl**;**

**}**

void Line**::**setLength**(** double len **)**

**{**

length **=** len**;**

**}**

double Line**::**getLength**(** void **)**

**{**

**return** length**;**

**}**

// Main function for the program

int main**()**

**{**

Line line**(**10.0**);** Line lines[100]; //se apeleaza automat constructorul

// get initially set length.

cout **<<** "Length of line : " **<<** line**.**getLength**()** **<<**endl**;**

// set line length again

line**.**setLength**(**6.0**);**

cout **<<** "Length of line : " **<<** line**.**getLength**()** **<<**endl**;**

**return** 0**;**

**}**

**// Incercati sa definiti un vector de linii -> Line lines[100];**

Line**::**Line**(** double len**):** length**(**len**)** // Liste de initializare

**{**

cout **<<** "Object is being created, length = " **<<** len **<<** endl**;**

**}**

#include <iostream> // Destructorul

using namespace std**;**

class Line

**{**

public**:**

void setLength**(** double len **);**

double getLength**(** void **);**

Line**();** // This is the constructor declaration

**~**Line**();** // This is the destructor: declaration

private**:**

double length**;**

**};**

// Member functions definitions including constructor

Line**::**Line**(**void**)**

**{**

cout **<<** "Object is being created" **<<** endl**;**

**}**

Line**::~**Line**(**void**)** //se apeleaza automat la final de program, f. bun pt dezalocare memorie dinamica

**{**

cout **<<** "Object is being deleted" **<<** endl**;**

**}**

void Line**::**setLength**(** double len **)**

**{**

length **=** len**;**

**}**

double Line**::**getLength**(** void **)**

**{**

**return** length**;**

**}**

int main**()**

**{**

Line line**;**

// set line length

line**.**setLength**(**6.0**);**

cout **<<** "Length of line : " **<<** line**.**getLength**()** **<<**endl**;**

**return** 0**;**

**}**

**// Ce fel de cod am putea pune pe un destructor, intr-un caz real?**

int main()

{

Line \*lines= new Line[3];

delete []lines; //ca sa stearga toate elementele create

**return** 0**;**

}

#include <iostream> // Constructorul de copier

// Se cheama la initializari, copieri de parametrii si returnari ale obiectelor.

using namespace std**;**

class Line **{**

public**:**

int getLength**(** void **) const;**

Line**(** int len **);** // simple constructor

Line**(** const Line **&**obj**);** // copy constructor

**~**Line**();** // destructor

private**:**

int **\***ptr**;**

**};**

// Member functions definitions including constructor

Line**::**Line**(**int len**)** **{**

cout **<<** "Normal constructor allocating ptr" **<<** endl**;**

// allocate memory for the pointer;

ptr **=** new int**;**

**\***ptr **=** len**;**

**}**

Line**::**Line**(**const Line **&**obj**)** **{**

cout **<<** "Copy constructor allocating ptr." **<<** endl**;**

ptr **=** new int**;**

**\***ptr **=** **\***obj**.**ptr**;** // copy the value

**}**

Line**::~**Line**(**void**)** **{**

cout **<<** "Freeing memory!" **<<** endl**;**

delete ptr**;**

**}**

int Line**::**getLength**(** void **)** **const**

**{**

**return** **\***ptr**;**

**}**

void display**(const** Line &obj**)** //&-ca sa nu mai faca copie

//const ca sa nu modifice clasa din cauza lui &

**{**

cout **<<** "Length of line : " **<<** obj**.**getLength**()** **<<**endl**;**

**}**

// Main function for the program

int main**()** **{**

Line line1**(**10**);**

Line line2 **=** line1**;** // This also calls copy constructor

display**(**line1**);**

display**(**line2**);**

**return** 0**;**

**}**

**// Exercitiu: Hai sa facem ca functia display sa nu-l mai cheme.**

#include <iostream> // Functii friend

using namespace std**;**

class Box **{**

double width**;**

public**:**

friend void printWidth**(** Box box **);**

void setWidth**(** double wid **);**

**};**

// Member function definition

void Box**::**setWidth**(** double wid **)**

**{**

width **=** wid**;**

**}**

// Note: printWidth() is not a member function of any class.

void printWidth**(** Box box **)** //functia friend poate sa editeze membrii private din clasa

**{**

/\* Because printWidth() is a friend of Box, it can

directly access any member of this class \*/

cout **<<** "Width of box : " **<<** box**.**width **<<**endl**;**

**}**

// Main function for the program

int main**()** **{**

Box box**;**

// set box width without member function

box**.**setWidth**(**10.0**);**

// Use friend function to print the wdith.

printWidth**(** box **);**

**return** 0**;**

**}**

#include <iostream>

// Pointerul this

using namespace std**;**

class Box **{**

public**:**

// Constructor definition

Box**(**double l **=** 2.0**,** double b **=** 2.0**,** double h **=** 2.0**)**

**{**

cout **<<**"Constructor called." **<<** endl**;**

length **=** l**;**

breadth **=** b**;**

height **=** h**;**

**}**

double Volume**()**

**{**

**return** length **\*** breadth **\*** height**;**

**}**

int compare**(**Box b**)**

**{**

**return** this**->**Volume**()** **>** b**.**Volume**();**

**}**

private**:**

double length**;** // Length of a box

double breadth**;** // Breadth of a box

double height**;** // Height of a box

**};**

int main**(**void**)** **{**

Box Box1**(**3.3**,** 1.2**,** 1.5**);** // Declare box1

Box Box2**(**8.5**,** 6.0**,** 2.0**);** // Declare box2

**if(**Box1**.**compare**(**Box2**))**

**{**

cout **<<** "Box2 is smaller than Box1" **<<**endl**;**

**}**

**else**

**{**

cout **<<** "Box2 is equal to or larger than Box1" **<<**endl**;**

**}**

**return** 0**;**

**}**

#include <iostream>

// Membri statici

using namespace std**;**

class Box **{**

public**:**

static int objectCount**; //static va ramane valoarea lui objectCount pe tot parcursul programului si nu au un pointer “this”**

// Constructor definition

Box**(**double l **=** 2.0**,** double b **=** 2.0**,** double h **=** 2.0**)** **{**

cout **<<**"Constructor called." **<<** endl**;**

length **=** l**;**

breadth **=** b**;**

height **=** h**;**

// Increase every time object is created

objectCount**++;**

**}**

double Volume**()** **{**

**return** length **\*** breadth **\*** height**;**

**}**

private**:**

double length**;** // Length of a box

double breadth**;** // Breadth of a box

double height**;** // Height of a box

**};**

// Initialize static member of class Box

int Box**::**objectCount **=** 0**;**

int main**(**void**)** **{**

Box Box1**(**3.3**,** 1.2**,** 1.5**);** // Declare box1

Box Box2**(**8.5**,** 6.0**,** 2.0**);** // Declare box2

// Print total number of objects.

cout **<<** "Total objects: " **<<** Box**::**objectCount **<<** endl**;**

**return** 0**;**

**}**

#include <iostream>

// Metode statice

using namespace std**;**

class Box **{**

public**:**

static int objectCount**;**

// Constructor definition

Box**(**double l **=** 2.0**,** double b **=** 2.0**,** double h **=** 2.0**)** **{**

cout **<<**"Constructor called." **<<** endl**;**

length **=** l**;**

breadth **=** b**;**

height **=** h**;**

// Increase every time object is created

objectCount**++;**

**}**

double Volume**()** **{**

**return** length **\*** breadth **\*** height**;**

**}**

static int getCount**()** **{**

**return** objectCount**;**

**}**

private**:**

double length**;** // Length of a box

double breadth**;** // Breadth of a box

double height**;** // Height of a box

**};**

// Initialize static member of class Box

int Box**::**objectCount **=** 0**;**

int main**(**void**)** **{**

// Print total number of objects before creating object.

cout **<<** "Inital Stage Count: " **<<** Box**::**getCount**()** **<<** endl**;**

Box Box1**(**3.3**,** 1.2**,** 1.5**);** // Declare box1

Box Box2**(**8.5**,** 6.0**,** 2.0**);** // Declare box2

// Print total number of objects after creating object.

cout **<<** "Final Stage Count: " **<<** Box**::**getCount**()** **<<** endl**;**

**return** 0**;**

**}**