Exercițiul 1

Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți de ce nu este corect și realizați o modificare astfel încât acesta să compileze fără a-i schimba funcționalitatea.

```
#include <iostream>
using namespace std;
class A
{
private:
    const int x = 5;

public:
    A() : x(2)
    {
    }
    const int getX() const
    {
        return x;
    }
};

int main()
{
    A *obj = new A();
    cout << obj->getX();
    return 0;
}
```

Rezolvare:

Programul compilează. Acesta afișează valoarea 2.

Explicație:

```
class A
{
private:
    const int x = 5; // inițializare (via inițializator implicit)

public:
    A() : x(2)  // inițializare (via lista de inițializare)
    {
    }
    const int getX() const
    {
        return x;
```

```
    }
};
```

Instrucțiunea `A *obj = new A();` creează o **instanță** a clasei `A`, alocarea find realizată în mod **dinamic** pe **heap**.

În momentul instanțierii, se "apelează" **constructorul fără parametri** al clasei `A` care folosește **lista de inițializare** a constructorului pentru a inițializa câmpul de date `x` cu valoarea `2`.

Câmpurile de date `const` ale unei clase nu pot fi *modificate*, dar pot fi *inițializate* folosind **lista de inițializare** sau **inițializarea implicită**.

Dacă ambele tipuri de inițializări sunt furnizate pentru același câmp de date, **inițializarea implicită** va fi ignorată.

De aceea `x` este inițializat în cele din urmă cu valoarea `2`. Câmpul de date este apoi afișat pe ecran folosind apelul către o **metodă constantă** `cout << obj->getX();`.

Exercițiul 2

Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți de ce nu este corect și realizați o modificare astfel încât acesta să compileze fără a-i schimba funcționalitatea.

```
#include <iostream>
using namespace std;
class A
{
private:
    int x = 5;

public:
    A() : x(2)
    {
        x = 6;
    }
    const int getX() const
    {
        return x;
    }
};

int main()
{
    A *obj = new A();
    cout << obj->getX();
    return 0;
}
```

Rezolvare:

Programul compilează. Acesta afișează valoarea 6.

Explicație:

```
class A
{
private:
    int x = 5; // inițializare (via inițializator implicit)

public:
    A() : x(2) // inițializare (via lista de inițializare)
    {
        x = 6; // atribuire
    }
    const int getX() const
    {
        return x;
    }
};
```

Instrucțiunea `A *obj = new A();` creează o **instanță** a clasei `A`, alocarea find realizată în mod **dinamic** pe **heap**.

În momentul instanțierii, se "apelează" **constructorul fără parametri** al clasei `A` care folosește **lista de inițializare** a constructorului pentru a inițializa câmpul de date `x` cu valoarea `2`.

Dacă ambele tipuri de inițializări sunt furnizate pentru același câmp de date, **inițializarea implicită** va fi ignorată indiferent dacă câmpul este *constant* sau nu.

De aceea `x` este inițializat în cele din urmă cu valoarea `2`, iar în corpul constructorului valoarea lui `x` devine 6.

Câmpul de date este apoi afișat pe ecran folosind apelul către o **metodă constantă** `cout << obj->getX();`.

Exercițiul 3

Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți de ce nu este corect și realizați o modificare astfel încât acesta să compileze fără a-i schimba funcționalitatea.

```
#include <iostream>
using namespace std;
class Test
{
public:
    Test(Test &) {}
    Test() {}
};
Test fun()
```

```
{
    cout << "fun() Called\n";
    Test t;
    return t;
}
int main()
{
    Test t1;
    Test t2 = fun();
    return 0;
}
```

Rezolvare:

Programul nu compilează. Acesta nu trece de compilare din cauza definiției greșite a **constructorului de copiere** `Test(Test &) {}`.

Funcția `fun ()` întoarce o valoare. Deci compilatorul creează un **obiect temporar** care este copiat în t2 folosind **constructorul de copiere** în functia main (Obiectul temporar este transmis ca argument către constructorul de copiere).

Motivul pentru eroarea compilatorului este că **obiectele temporare create de compilator** nu pot fi legate la **referințe non-const** și programul încearcă să facă acest lucru.

Nu are sens să **modificați** obiectele temporare create de compilator, deoarece **pot fi distruse** în orice moment.

Dacă folosim definiția corectă a **constructorului de copiere** atunci programul va compila:

```
#include <iostream>
using namespace std;
class Test
{
public:
    Test(const Test &) {}
    Test() {}
};
Test fun()
{
    cout << "fun() Called\n";
    Test t;
    return t;
}
int main()
{
    Test t1;
    Test t2 = fun();
    return 0;
}
```

Exercițiul 4

Spuneți dacă programul de mai jos este corect. În caz negativ spuneți de ce nu este corect.

```
#include <iostream>
using namespace std;
class Test
{
    Test self;
};
int main()
{
    Test t;
    getchar();
    return 0;
}
```

Rezolvare:

Programul nu compilează.

Dacă un obiect **non static** este membru, atunci declarația clasei este incompletă și compilatorul nu are cum să calculeze dimensiunea obiectelor clasei.

Pentru un compilator, toți membri unei clase au o **dimensiune fixă** indiferent de tipul de date pe care îl indică.

Exercițiul 5

Spuneți dacă programul de mai jos este corect. În caz negativ spuneți de ce nu este corect.

```
#include <iostream>
using namespace std;
class Test
{
    static Test self;
};
int main()
{
    Test t;
    getchar();
    return 0;
}
```

Rezolvare:

Programul compilează.

Exercițiul 6

Spuneți dacă programul de mai jos este corect. În caz negativ spuneți de ce nu este corect.

```
#include <iostream>
using namespace std;
class Test
{
    Test *self;
};
int main()
{
    Test t;
    getchar();
    return 0;
}
```

Rezolvare:

Programul compilează.

Concluzia:

Declarația unei clase `A`:

- Poate conține **obiecte statice** de tipul clasei `A`.
- Poate avea **pointeri** de tipul clasei `A`,
- **NU** poate avea un obiect **non static** de tipul clasei `A`.

Exercițiul 7

Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți de ce nu este corect și realizați o modificare astfel încât acesta să compileze fără a-i schimba funcționalitatea.

```
#include <iostream>
using namespace std;
class Test
{
    static int x;

public:
    Test() { x++; }
    static int getX() { return x; }
};
int Test::x = 0;
int main()
{
    cout << Test::getX() << " ";
    Test t[5];
    cout << Test::getX();
}
```

Rezolvare:

Programul compilează. Acesta afișează valorile `0` și `5`.

Explicație:

Definiția clasei `Test` conține un **câmp de date static** de tip întreg `x` care este inițializat cu valoarea `0` de către instrucțiunea `int Test::x = 0;`.

Instrucțiunea `cout << Test::getX() << " ";` afișează valoarea `0` și un spațiu apelând **metoda statică** `getX` care întoarce valoarea câmpului de date static `x`.

Apoi este instanțiat un **vector de dimensiune 5** cu obiecte de tipul clasei `Test`.

Pentru **fiecare instanțiere** este **incrementată** valoarea **câmpului static** `x`, astfel la executarea instrucțiunii `cout << Test::getX();` se va afișa valoarea `5` deoarece, *să nu uităm* că:

> Un câmp de date static "aparține unei clase, nu unei instanțe a clasei".

Exercițiul 8

Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți de ce nu este corect și realizați o modificare astfel încât acesta să compileze fără a-i schimba funcționalitatea.

```
#include <iostream>
using namespace std;
class Test
{
private:
    static int count;

public:
    Test &fun();
};
int Test::count = 0;
Test &Test::fun()
{

    Test::count++;
    cout << Test::count << " ";
    return *this;
}
int main()
{
    Test t;
    t.fun().fun().fun().fun();
    return 0;
}
```

Rezolvare:

Programul compilează. Acesta afișează: `1 2 3 4` .

Explicație:

Membri statici sunt accesibili în **metodele non-statice**, deci nu există nici o problemă în accesarea câmpului de date static `count` în metoda `fun`.

Exercițiul 9

Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți de ce nu este corect și realizați o modificare astfel încât acesta să compileze fără a-i schimba funcționalitatea.

```
#include <iostream>
using namespace std;
class Cls
{
    int x;

public:
    Cls(int i) : x(i) {}
    const int &f()
    {
        return x;
    }
};
int main()
{
    Cls a(14);
    int b = a.f()++;
    cout << b << '\n';
    return 0;
}
```

Rezolvare:

Programul nu compilează.

Problema apare de la instrucțiunea `a.f()++;` deoarece metoda `f` returnează o **referință constantă** `const int &` care nu poate fi incrementată.

O metodă simplă de rezolvare care va face programul să compileze este următoarea:

```
#include <iostream>
using namespace std;
class Cls
{
    int x;
```

```
public:
    Cls(int i) : x(i) {}
    int &f()
    {
        return x;
    }
};
int main()
{
    Cls a(14);
    int b = a.f()++;
    cout << b << '\n';
    return 0;
}
```

Am schimbat definiția metodei `f` din `const int &` în `int &`, deci din **referință constantă** în **referință neconstantă**.

Exercițiul 10

Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți de ce nu este corect și realizați o modificare astfel încât acesta să compileze fără a-i schimba funcționalitatea.

```
#include <iostream>
using namespace std;
class A
{
    int x;
    static int y;

public:
    A(int i, int j) : x(i), y(j)
    {
    }
    int f() const;
};
int A::y;
int A::f() const
{
    return y;
}
int main()
{
    const A a(21, 2);
    cout << a.f();
    return 0;
}
```

Rezolvare:

Programul nu compilează.

Definiția constructorului clasei `A` creează o **eroare de compilare** și anume `A(int i, int j) : x(i), y(j)` pentru că prin intermediul acesteia se încearcă **inițializarea unui membru static** folosind **lista de inițializare** a constructorului ceea ce nu este permis.

Un *membru static* poate fi inițializat doar *în afara definiției unei clase.*

O metodă de eliminare a acestei erori este de a **elimina inițializarea** lui `y` prin intermediul **listei de inițializare** a constructorului clasei `A`, realizând o **atribuire în corpul constructorului** pentru a nu schimba comportamentul dorit:

```
#include <iostream>
using namespace std;
class A
{
    int x;
    static int y;

public:
    A(int i, int j) : x(i)
    {
        y = j;
    }
    int f() const;
};
int A::y;
int A::f() const
{
    return y;
}
int main()
{
    const A a(21, 2);
    cout << a.f();
    return 0;
}
```

Exercițiul 11

Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți de ce nu este corect și realizați o modificare astfel încât acesta să compileze fără a-i schimba funcționalitatea.

```
#include <iostream>
using namespace std;
class A
{
```

```
    int x, *y;

public:
    A(int i)
    {
        x = i;
        y = new int[x];
    }
    A(A &a)
    {
        x = a.x;
        y = new int[x];
    }
    int getX() const
    {
        return x;
    }
};
int f(A a)
{
    return a.getX();
}
int main()
{
    const A a(5);
    cout << (a.getX() == f(a));
    return 0;
}
```

Rezolvare:

Programul nu compilează. Eroarea este cauzată de definiția **constructorului de copiere** al clasei `A`, deoarece acesta nu respectă definiția standard.

În momentul apelării funcției `f` folosind un **obiect constant**, compilatorul emite o eroare deoarece nu există o definiție a **constructorului de copiere** care să poată realiza o copie a acestui tip de obiect.

Corectând definiția constructorului de copiere programul va compila:

```
#include <iostream>
using namespace std;
class A
{
    int x, *y;

public:
    A(int i)
    {
        x = i;
        y = new int[x];
    }
    A(const A &a)
```

```
    {
        x = a.x;
        y = new int[x];
    }
    int getX() const
    {
        return x;
    }
};
int f(A a)
{
    return a.getX();
}
int main()
{
    const A a(5);
    cout << (a.getX() == f(a));
    return 0;
}
```

Exercițiul 12

Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți de ce nu este corect și realizați o modificare astfel încât acesta să compileze fără a-i schimba funcționalitatea.

```
#include <iostream>
using namespace std;
class Cls
{
    int x;

public:
    Cls(int i = 0)
    {
        cout << 1;
        x = i;
    }
    Cls(Cls &ob)
    {
        cout << 2;
        x = ob.x;
    }
    int getX() const
    {
        return x;
    }
};
Cls &f(Cls &c)
{
    return c;
```

```
}
int main()
{
    Cls r;
    Cls s = f(f(f(r)));
    cout << s.getX();
    return 0;
}
```

Rezolvare:

Programul compilează. Acesta afișează valoarea 120.

Explicație:

Instrucțiunea `Cls r;` creează o **instanță** a clasei `Cls`, moment în care este "apelat" **constructorul cu un parametru implicit** al clasei `Cls` în care se afișează valoarea `1`, iar câmpul de date `x` primește valoarea parametrului implicit adică `0`.

Functia `f` primește o **referință către un obiect** de tipul clasei `Cls` și întoarce o **referință către acel obiect**, deci este important de menționat că **nu este apelat constructorul de copiere**.

Constructorul de copiere al clasei `Cls` e apelat în momentul în care se inițializează obiectul `s`, se afișează valoarea `2`, iar campul de date `x` al lui `s` devine `0`.

În final se apelează metoda `getX` care returnează valoarea lui `x` adica `0`.

Exercițiul 13

Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți de ce nu este corect și realizați o modificare astfel încât acesta să compileze fără a-i schimba funcționalitatea.

```
#include <iostream>
using namespace std;
class Cls
{
    static int i;
    int j;

public:
    Cls(int x = 7)
    {
        j = x;
    }
    static int imp(int k)
    {
        Cls a;
        return i + k + a.j;
    }
    int getJ() const
```

```
    {
        return j;
    }
};
int Cls::i;
int main()
{
    int k = 5;
    cout << Cls::imp(k);
    return 0;
}
```

Rezolvare:

Programul compilează. Acesta afișează valoarea 12.

Explicație:

Observăm inițializarea **câmpului de date static** al clasei `Cls` cu valoarea implicită `0` realizată de instrucțiunea `int Cls::i;`.

În momentul apelării **metodei statice** `imp` a clasei `Cls` observăm că se creeaza un **obiect de tipul clasei** `Cls` în corpul acesteia.

Deci `a.j` primește valoarea 7 deoarece se apelează **constructorul cu un parametru implicit** al clasei `Cls`, astfel metoda `imp` returnează valoarea `0 + 5 + 7`, adică `12`.

Exercițiul 14

Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți de ce nu este corect și realizați o modificare astfel încât acesta să compileze fără a-i schimba funcționalitatea.

```
#include <iostream>
using namespace std;
class Cls
{
    int x;

public:
    Cls(int i = 32)
    {
        x = i;
    }
    int f() const
    {
        return x++;
    }
};
int main()
{
```

```
    const Cls d(-15);
    cout << d.f();
    return 0;
}
```

Rezolvare:

Programul nu compilează.

Eroarea apare din cauza instrucțiunii `return x++;` deoarece aceasta este inclusă într-o **metodă constantă**, iar metodele constante **nu pot modifica câmpurile de date ale unui clase**, le pot doar accesa.

Pentru a rezolva eroarea și pentru a păstra comportamentul dorit trebuie să eliminăm **flag-ul const** atât din declarația obiectului, cât și din definiția metodei:

```
#include <iostream>
using namespace std;
class Cls
{
    int x;

public:
    Cls(int i = 32)
    {
        x = i;
    }
    int f()
    {
        return x++;
    }
};
int main()
{
    Cls d(-15);
    cout << d.f();
    return 0;
}
```

Exercițiul 15

Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți de ce nu este corect și realizați o modificare astfel încât acesta să compileze fără a-i schimba funcționalitatea.

```
#include <iostream>
using namespace std;
int &fun()
{
    static int a = 10;
```

```
    return a;
}
int main()
{
    int &y = fun();
    y = y + 30;
    cout << fun();
    return 0;
}
```

Rezolvare:

Programul compilează. Acesta afișează valoarea 40.

Explicație:

Când o variabilă este declarată **statică**, spațiul pentru aceasta este alocat pentru **toată durata de viață a programului**.

Chiar dacă funcția este apelată de mai multe ori, spațiul pentru variabila statică este alocat o singură dată, iar valoarea variabilei din apelul anterior rămâne aceeași în apelurile viitoare.

Exercițiul 16

Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți de ce nu este corect și realizați o modificare astfel încât acesta să compileze fără a-i schimba funcționalitatea.

```
#include <iostream>
using namespace std;
class Test
{
    int value;

public:
    Test(int v = 0) { value = v; }
    int getValue() { return value; }
};
int main()
{
    const Test t;
    cout << t.getValue();
    return 0;
}
```

Rezolvare:

Programul nu compilează.

Obiectul `t` este declarat ca **obiect constant** astfel el poate apela doar **metode constante**, iar metoda `getValue` observăm că nu este constantă.

Dacă facem metoda `getValue` constantă atunci programul va compila:

```
#include <iostream>
using namespace std;
class Test
{
    int value;

public:
    Test(int v = 0) { value = v; }
    int getValue() const { return value; }
};
int main()
{
    const Test t;
    cout << t.getValue();
    return 0;
}
```

Exercițiul 17

Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți de ce nu este corect și realizați o modificare astfel încât acesta să compileze fără a-i schimba funcționalitatea.

```
#include <iostream>
using namespace std;
class X
{
private:
    static const int a = 76;

public:
    static int getA() { return a; }
};
int main()
{
    cout << X::getA() << endl;
    return 0;
}
```

Rezolvare:

Programul compilează. Acesta afișează valoarea 76.

Explicație:

Dacă un **câmp de date static** este de tip `const`, acesta poate fi inițializat direct în definiția clasei.

Exercițiul 18

Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți de ce nu este corect și realizați o modificare astfel încât acesta să compileze fără a-i schimba funcționalitatea.

```
#include <iostream>
using namespace std;
class A
{
    int aid;

public:
    A(int x)
    {
        aid = x;
    }
    void print()
    {
        cout << "A::aid = " << aid;
    }
};
class B
{
    int bid;

public:
    static A a;
    B(int i) { bid = i; }
};
int main()
{
    B b(10);
    b.a.print();
    return 0;
}
```

Rezolvare:

Programul nu compilează deoarece **câmpul de date static** `static A a;` din definiția clasei `B` nu este inițializat în afara clasei.

Pentru a rezolva problema, inițializăm câmpul de date static în afara clasei astfel:

```
#include <iostream>
using namespace std;
class A
{
```

```
    int aid;

public:
    A(int x)
    {
        aid = x;
    }
    void print()
    {
        cout << "A::aid = " << aid;
    }
};
class B
{
    int bid;

public:
    static A a;
    B(int i) { bid = i; }
};
A B::a(0);
int main()
{
    B b(10);
    b.a.print();
    return 0;
}
```

Exercițiul 19

Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți de ce nu este corect și realizați o modificare astfel încât acesta să compileze fără a-i schimba funcționalitatea.

```
#include <iostream>
using namespace std;
class A
{
    int id;
    static int count;

public:
    A()
    {
        count++;
        id = count;
        cout << "constructor called " << id << endl;
    }
    ~A()
    {
        cout << "destructor called " << id << endl;
```

```
    }
};
int A::count = 0;
int main()
{
    A a[2];
    return 0;
}
```

Rezolvare:

Programul compilează. Acesta afișează:

```
constructor called 1
constructor called 2
destructor called 2
destructor called 1
```

Explicație:

Se instanțiază un vector de obiecte de tipul clasei `A` care ulterior sunt distruse. Nimic special.

Exercițiul 20

Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți de ce nu este corect și realizați o modificare astfel încât acesta să compileze fără a-i schimba funcționalitatea.

```
#include <iostream>
using namespace std;
class Test
{
private:
    int x;
    static int count;

public:
    Test(int i = 0) : x(i) {}
    Test(const Test &rhs) : x(rhs.x) { ++count; }
    static int getCount() { return count; }
};
int Test::count = 0;
Test fun()
{
    return Test();
}
int main()
{
    Test a = fun();
```

```
    cout << Test::getCount();
    return 0;
}
```

Rezolvare:

Programul compilează, dar rezultatul furnizat de acesta depinde de compilatorul utilizat. Acesta poate să afișeze 0 sau 2.

Explicație:

Instrucțiunea `Test a = fun();` poate sau nu să apeleze **constructorul de copiere**.

Deci, ieșirea poate fi 0 sau 2. Acest fenomen se întâmplă când compilatorul realizează o optimizare ce se numeste **copy elision**, iar **constructorul de copiere nu va fi apelat.**

Dacă nu se realizează **copy elision**, va fi apelat **constructorul de copiere**.

Compilare cu g++ forțată să nu folosească copy elision:

```
anpopescu@ANPOPESCU-L:~/Documents/Tutoriate$ g++ -fno-elide-constructors
main.cpp -o main.out
anpopescu@ANPOPESCU-L:~/Documents/Tutoriate$ ./main.out
2
```

Compilare cu g++ cu standardul din 2017 care garantează că se va folosi copy elision:

```
anpopescu@ANPOPESCU-L:~/Documents/Tutoriate$ g++ -std=c++17 main.cpp -o
main.out
anpopescu@ANPOPESCU-L:~/Documents/Tutoriate$ ./main.out
0
```

Sugestie examen: Gândiți-vă bine pe ce standard vă faceți rezolvările.

Exercițiul 21

Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți de ce nu este corect și realizați o modificare astfel încât acesta să compileze fără a-i schimba funcționalitatea.

```
#include <iostream>
using namespace std;
class Cls
{
    int x;

public:
```

```
    int setX(int i)
    {
        int y = x;
        x = i;
        return x;
    }
    int getX() { return x; }
};
int main()
{
    Cls *p = new Cls[100];
    int i = 0;
    for (; i < 50; i++)
        p[i].setX(i);
    for (i = 5; i < 20; ++i)
    {
        cout << p[i].getX() << " ";
    }
    return 0;
}
```

Rezolvare:

Programul compilează. Acesta afișează:

```
5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

Exercițiul 22

Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți de ce nu este corect și realizați o modificare astfel încât acesta să compileze fără a-i schimba funcționalitatea.

```
#include <iostream>
using namespace std;
class Cls
{
    int x;

public:
    Cls(int i = 1)
    {
        x = i;
    }
    int setX(int i)
    {
        int y = x;

        x = i;
```

```
        return y;
    }
    int getX()
    {
        return x;
    }
};
int main()
{
    Cls *p = new Cls[10];
    int i = 0;
    for (; i < 10; ++i)
        p[i].setX(i);
    for (i = 0; i < 10; i++)
        cout << p[i].getX(i);
    return 0;
}
```

Rezolvare:

Programul nu compilează, deoarece metoda `getX` nu conține în antentul său nici un parametru, iar aceasta este apelată cu un parametru în `main`.

Codul care compilează:

```
#include <iostream>
using namespace std;
class Cls
{
    int x;

public:
    Cls(int i = 1)
    {
        x = i;
    }
    int setX(int i)
    {
        int y = x;

        x = i;
        return y;
    }
    int getX()
    {
        return x;
    }
};
int main()
{
    Cls *p = new Cls[10];
    int i = 0;
```

```
    for (; i < 10; ++i)
        p[i].setX(i);
    for (i = 0; i < 10; i++)
        cout << p[i].getX();
    return 0;
}
```

Exercițiul 23

Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți de ce nu este corect și realizați o modificare astfel încât acesta să compileze fără a-i schimba funcționalitatea.

```
#include <iostream>
using namespace std;
int a = 2;
class Test
{
    int &t = a;

public:
    Test(int &t) : t(t) {}
    int getT() { return t; }
};
int main()
{
    int x = 20;
    Test t1(x);
    cout << t1.getT() << endl;
    x = 30;

    cout << t1.getT() << endl;
    return 0;
}
```

Rezolvare:

Programul compilează. Acesta afișează:

```
20
30
```

Explicație:

Se instanțiază un obiect `t1` al cărui membru `t` (**referință**) se va referi la `x`-ul declarat in `main`.

Astfel, când modificăm `x`-ul, în mod automat se "modifică" și `t`-ul (valoarea referențiată de `t`, atât `x` cât și `t` se referă la aceeași zonă de memorie).

Exercițiul 24

Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează, în caz negativ spuneți de ce nu este corect și realizați o modificare astfel încât acesta să compileze fără a-i schimba funcționalitatea.

```
#include <iostream>
using namespace std;
class Point
{
private:
    int x;
    int y;

public:
    Point(int i = 0, int j = 0);
    Point(const Point &t);
};
Point::Point(int i, int j)
{
    x = i;
    y = j;
    cout << "Normal Constructor called\n";
}
Point::Point(const Point &t)
{
    y = t.y;
    cout << "Copy Constructor called\n";
}
int main()
{
    Point *t1, *t2;
    t1 = new Point(10, 15);
    t2 = new Point(*t1);
    Point t3 = *t1;
    return 0;
}
```

Rezolvare:

Programul compilează. Acesta afișează:

```
Normal Constructor called
Copy Constructor called
Copy Constructor called
```

Explicație:

Inițial este instanțiat pe `heap` un obiect de tipul clasei `Point`.

Apoi, se creează o altă instanță din obiectul `t1` și astfel se apelează **constructorul de copiere** în mod explicit.

În ultimul rând, la executarea instrucțiunii `Point t3 = *t1;` **constructorul de copiere** mai este apelat încă o dată.