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
{
public:
    void print() { cout << "A::print()"; }
};
class B : private A
{
public:
    void print() { cout << "B::print()"; }
};
class C : public B
{
public:
    void print() { A::print(); }
};
int main()
{
    C b;
    b.print();
}
```

Rezolvare:

Programul nu compilează. Va da eroare de compilare deoarece clasa `B` moștenește clasa `A` în mod private și chiar dacă clasa `C` moștenește clasa `B` în mod public nu vom avea acces la metoda `print` din clasa `A` deoarece aceasta devine metodă privată în clasa `B`, deci nu va putea fi accesată din clasa `C`.

O posibilă rezolvare ar fi ca clasa `B` să moștenească clasa `A` în mod public sau protected.

```
#include <iostream>
using namespace std;
class A
{
public:
    void print() { cout << "A::print()"; }
};
class B : public A
{
public:
    void print() { cout << "B::print()"; }
};
```

```
class C : public B
{
public:
    void print() { A::print(); }
};
int main()
{
    C b;
    b.print();
}
```

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 base
{
public:
    void show() { cout << " In Base \n"; }
};
class derived : public base
{
    int x;

public:
    void show() { cout << "In derived \n"; }
    derived()
    {
        x = 10;
    }
    int getX() const { return x; }
};
int main()
{
    derived d;
    base *bp = &d;
    bp->show();
    cout << bp->getX();
    return 0;
}
```

Rezolvare:

Programul va da eroare de compilare la instrucțiunea `cout << bp->getX()` deoarece clasa de baza nu conține metoda `int getX() const { return x; }`. În acest program se realizează procedeul de `Upcasting` : mai întâi este declarat și instanțiat pe stivă un obiect de tip `derived`, apoi se face

`Upcasting` printr-un pointer de tipul clasei `base` care va face referire la obiectul anterior. Instrucțiunea `bp->show()` ar rula și ar afișa `In base \n` dar când compilatorul ajunge la linia în care se încearcă accesarea metodei `bp->getX()` acesta va semnala că clasa `base` nu are un asemenea membru.

O posibilă rezolvare ar fi să schimbăm tipul pointer-ului `bp` și să îl facem de tip `derived` :

```
#include <iostream>
using namespace std;
class base
{
public:
    void show() { cout << " In Base \n"; }
};
class derived : public base
{
    int x;

public:
    void show() { cout << "In derived \n"; }
    derived()
    {
        x = 10;
    }
    int getX() const { return x; }
};
int main()
{
    derived d;
    derived *bp = &d;
    bp->show();
    cout << bp->getX();
    return 0;
}
```

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 B
{
    int x;

public:
    B(int v) { v = x; }
    int get_x() { return x; }
};
```

```
class D : private B
{
    int y;

public:
    D(int v) : B(v) {}
    int get_x() { return x; }
};
int main()
{
    D d(10);
    cout << d.get_x();
    return 0;
}
```

Rezolvare:

Programul nu va trece de compilare deoarece clasa `D` moștenește clasa `B`, iar cum `x` este un membru privat al clasei `B` acesta nu va fi moștenit în clasa `D`, deci prin urmare nu va putea fi accesat.

O posibilă rezolvare ar fi să îl facem pe `x` membru protected sau public (nerecomandat). Atenție mare aici, rezultatul rulării va fi unul random deși în unele cazuri are compilatorul grijă să facă zonele de memorie 0 acest lucru nu este garantat.

```
#include <iostream>
using namespace std;
class B
{
protected:
    int x;

public:
    B(int v) { v = x; }
    int get_x() { return x; }
};
class D : private B
{
    int y;

public:
    D(int v) : B(v) {}
    int get_x() { return x; }
};
int main()
{
    D d(10);
    cout << d.get_x();
    return 0;
}
```

Rezultat compilare:

```
anpopescu@ANPOPESCU-L:~/Documents/Tutoriate$ g++ -std=c++17 main.cpp -o
main.out
anpopescu@ANPOPESCU-L:~/Documents/Tutoriate$ ./main.out
-922170048
anpopescu@ANPOPESCU-L:~/Documents/Tutoriate$ ./main.out
772136096
anpopescu@ANPOPESCU-L:~/Documents/Tutoriate$ ./main.out
-1146780464
```

Exercițiul 4

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 B
{
    int a;
    B(int i = 0) { a = i; }
    int get_a() { return a; }
};
class D : protected B
{
public:
    D(int x = 0) : B(x) {}
    int get_a() { return B::get_a(); }
};
int main()
{
    D d(-89);
    cout << d.get_a();
    return 0;
}
```

Rezolvare:

Programul nu va trece de compilare deoarece toți membri clasei `B` sunt declarați ca private inclusiv constructorul acestuia este redefinit, deci nu vom putea instanția un obiect folosind pattern-ul clasei `B`. Astfel când se încearcă crearea unui obiect de tipul clasei `D` compilatorul va semnala o eroare că constructorul clasei `B` este private.

O posibilă soluție ar fi să facem constructorul și metoda `get_a()` membri public sau protected.

```
#include <iostream>
using namespace std;
class B
{
    int a;

public:
    B(int i = 0) { a = i; }
    int get_a() { return a; }
};
class D : protected B
{
public:
    D(int x = 0) : B(x) {}
    int get_a() { return B::get_a(); }
};
int main()
{
    D d(-89);
    cout << d.get_a();
    return 0;
}
```

Exercițiul 5

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 B
{
protected:
    int x;

public:
    B(int i = 16) { x = i; }
    B f(B ob) { return x + ob.x; }
    void afisare() { cout << x; }
};
class D : public B
{
public:
    B f(B ob)
    {
        return x + 1;
    }
};
```

```
int main()
{
    B *p1 = new D, *p2 = new B, *p3 = new B(p1->f(*p2));
    p3->afisare();
    return 0;
}
```

Rezolvare:

Programul trece de compilare cu succes și va afișa valoarea `32`. Vă recomandăm să luați programele de genul acesta pe foaie și să le rulați pas cu pas exact cum ar face compilatorul ca să vă prindeți de răspuns. În cazul de față pointer-ul `p1` va face referire către un obiect de tipul clasei `D`, deci membrul `x` al său va fi inițializat cu valoarea `16` deoarece este apelat constructorul implicit al clasei `D` care va apela la rândul lui constructorul cu parametru implicit al clasei `B` in cazul nostru (`i = 16`). Apoi pointer-ul `p2` va face referire la un obiect de tipul clasei `B` deci de asemenea i se va apela constructorul cu valoare implicită, deci `x`-ul lui va fi de asemenea `16`. Pointer-ul `p3` va fi instanțiant folosind clasa `B`, dar folosind o metodă definită din `p1` și anume `f` care primește ca parametru obiectul `p2`. Acum această funcție va întoarce un obiect care are în membrul `x` valoarea `32` deoarece `x`-ul lui `p1` este `16` și cel al lui `p2` este de asemenea `16`. Știm că pare ciudat, dar așa funcționează lucrurile in C++ pentru că funcția întoarce un obiect de tip `B`, dar acest obiect are în constructor un întreg de tip `int` și atunci compilatorul știe singur să creeze un obiect (e ca si cum ar apela singur constructorul cu parametrul `32`). În realitate de cele mai multe ori în cazuri ca acesta compilatorul știe să facă ceva ce se numește `Copy Ellision` (am vorbit despre Copy Ellision și în tutoriatele trecute).

Exercițiul 6

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 B
{
public:
    int x;
    B(int i = 16) { x = i; }
    B f(B ob) { return x + ob.x; }
};
class D : public B
{
public:
    D(int i = 25)
    {
        x = i;
    }
    B f(B ob)
    {
        return x + ob.x + 1;
```

```
    }
    void afisare()
    {
        cout << x;
    }
};
int main()
{
    B *p1 = new D, *p2 = new B, *p3 = new B(p1->f(*p2));
    cout << p3->x;
    return 0;
}
```

Rezolvare:

Programul este asemănator cu cel de la exercițiul 5. Încercați să îl faceți singuri.

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 A
{
public:
    int x;
    A(int i = 0)
    {
        x = i;
    }
    A minus()
    {
        return 1 - x;
    }
};
class B : public A
{
public:
    B(int i = 0) { x = i; }
    void afisare() { cout << x; }
};
int main()
{
    A *p1 = new B(18);
    *p1 = p1->minus();
    p1->afisare();
    return 0;
}
```

Rezolvare:

Programul nu va trece de compilare deoarece clasa `A` nu conține metoda `afisare`. Chiar dacă noi instanțiem un obiect de tipul clasei `B` în `main` se realizează procedeul de `Upcasting`, deci vom face referire la el printr-un obiect de tipul clasei `A` care nu are definită metoda `afișare()`.

O posibilă soluție este să definim metoda `afisare()` și în clasa `A`:

```
#include <iostream>
using namespace std;
class A
{
public:
    int x;
    A(int i = 0)
    {
        x = i;
    }
    A minus()
    {
        return 1 - x;
    }
    void afisare() { cout << x; }
};
class B : public A
{
public:
    B(int i = 0) { x = i; }
    void afisare() { cout << x; }
};
int main()
{
    A *p1 = new B(18);
    *p1 = p1->minus();
    p1->afisare();
    return 0;
}
```

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 A
{
protected:
```

```
    int x;

public:
    A(int i = -16) { x = i; }
    A f(A a) { return x + a.x; }
    void afisare() { cout << x; }
};
class B : public A
{
public:
    B(int i = 3) : A(i) {}
    B f(B b) { return x + b.x + 1; }
};
int main()
{
    A *p1 = new B, *p2 = new A, *p3 = new A(p1->f(*p2));
    p3->afisare();
    return 0;
}
```

Rezolvare:

Foarte asemănător cu exercițiile 5 și 6. Încercați să îl faceți singuri.

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 B
{
    int i;

public:
    B() { i = 1; }
    int get_i() { return i; }
};
class D : public B
{
    int j;

public:
    D() { j = 2; }
    int get_i() { return B::get_i() + j; }
};
int main()
{
    const int i = cin.get();
```

```
    if (i % 3)
    {
        D o;
    }
    else
    {
        B o;
    }
    cout << o.get_i();
    return 0;
}
```

Rezolvare:

Programul nu va trece de compilare deoarece obiectul `o` este creat indiferent de caz în `if` sau `else`, deci nu va fi accesibil în blocul exterior acestora și când se încearcă accesarea lui, compilatorul va semnala o eroare că obiectul `o` nu a fost declarat în scopul blocului respectiv (cel al lui main) .

O posibilă rezolvare ca să treacă de compilare este să declarăm un pointer de tipul clasei `B` și să realizăm instanțierea folosind procedeul de `Upcasting`:

```
#include <iostream>
using namespace std;
class B
{
    int i;

public:
    B() { i = 1; }
    int get_i() { return i; }
};
class D : public B
{
    int j;

public:
    D() { j = 2; }
    int get_i() { return B::get_i() + j; }
};
int main()
{
    const int i = cin.get();
    B *o;
    if (i % 3)
    {
        o = new D;
    }
    else
    {
        o = new B;
    }
```

```
    cout << o->get_i();
    return 0;
}
```

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 B
{
protected:
    int x;

public:
    B(int i = 28) { x = i; }
    B f(B ob) { return x + ob.x + 1; }
    void afisare() { cout << x; }
};
class D : public B
{
public:
    D(int i = -32) : B(i) {}
    B f(B ob) { return x + ob.x - 1; }
};
int main()
{
    B *p1 = new D, *p2 = new B, *p3 = new B(p1->f(*p2));
    p3->afisare();
    return 0;
}
```

Rezolvare:

Programul nu va trece de compilare deoarece se încearcă accesarea unui membru protected în clasa `D` și anume `x`.

O soluție posibilă ar fi să facem membrul `x` al clasei `B` public:

```
#include <iostream>
using namespace std;
class B
{
public:
    int x;
```

```
public:
    B(int i = 28) { x = i; }
    B f(B ob) { return x + ob.x + 1; }
    void afisare() { cout << x; }
};
class D : public B
{
public:
    D(int i = -32) : B(i) {}
    B f(B ob) { return x + ob.x - 1; }
};
int main()
{
    B *p1 = new D, *p2 = new B, *p3 = new B(p1->f(*p2));
    p3->afisare();
    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 Base
{
public:
    int fun()      { cout << "Base::fun() called"; }
    int fun(int i) { cout << "Base::fun(int i) called"; }
};

class Derived: public Base
{
public:
    int fun(char x)   { cout << "Derived::fun(char ) called"; }
};

int main()
{
    Derived d;
    d.fun();
    return 0;
}
```

Rezolvare:

Programul nu trece de compilare. În programul de mai sus, metoda `fun()` a clasei de bază nu este accesibilă în clasa derivată. Dacă o clasă derivată creează o metodă cu același nume ca una dintre metodele din clasa de bază, atunci toate metodele clasei de bază cu acest nume rămân ascunse în clasa derivată.

O posibilă rezolvare ar fi să folosim operatorul de rezoluție pentru a accesa metoda `fun()` din clasa `B`:

```
#include<iostream>
using namespace std;

class Base
{
public:
    int fun()      { cout << "Base::fun() called"; }
    int fun(int i) { cout << "Base::fun(int i) called"; }
};

class Derived: public Base
{
public:
    int fun(char x)   { cout << "Derived::fun(char ) called"; }
};

int main()
{
    Derived d;
    d.Base::fun();
    return 0;
}
```