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 B1 { public: int x; };
class B2 { public: int y; };
class B3 { public: int z; };
class B4 { public: int t; };
class D: public B1, private B2, protected B3, B4 { public: int u; };
int main() {
    D d;
    cout << d.u;
    cout << d.x;
    cout << d.y;
    cout << d.z;
    cout << d.t;
    return 0;
}
```

Rezolvare:

Programul nu compilează. Programul nu este corect din cauza *tipurilor de moștenire* folosite în definirea clasei D, astfel compilatorul va semnala erori pentru următoarele instrucțiuni:

```
cout << d.y;
cout << d.z;
cout << d.t;
```

Acest lucru se întâmplă pentru că:

- clasa D moștenește clasa B2 în mod **private explicit**, iar această moștenire presupune că totul din clasa de *BAZĂ (B2)* devine **private** în clasa *DERIVATĂ (D)* ;
- clasa D moștenește clasa B3 în mod **protected**, iar această moștenire presupune că totul din clasa de *BAZĂ (B3)* devine **protected** în clasa *DERIVATĂ (D)* ;
- clasa D moștenește clasa B4 în mod **private implicit** (fără a preciza în mod explicit tipul de moștenire), iar această moștenire presupune că totul din clasa de *BAZA (B4)* devine **private** în clasa *DERIVATĂ (D)*.

Având în vedere punctele menționate mai sus programul *nu compilează* deoarece câmpurile de date y, t sunt **private**, iar câmpul de date z este **protected** în *clasa D*, deci nu sunt accesibile în mod direct printr-o *instanță* a clasei D.

Pentru a face programul să compileze și pentru a nu îi schimba funcționalitatea putem moșteni cele 4 clase în mod **public** astfel:

```
#include <iostream>
using namespace std;
class B1 { public: int x; };
class B2 { public: int y; };
class B3 { public: int z; };
class B4 { public: int t; };
class D: public B1, public B2, public B3, public B4 { public: int u; };
int main() {
    D d;
    cout << d.u;
    cout << d.x;
    cout << d.y;
    cout << d.z;
    cout << d.t;
    return 0;
}
```

Dar daca în clasele *"B1", "B2", "B3", "B4"* în loc de *"public"* scriam *"private"* sau *"protected"*?

Rezolvare:

Programul nu ar fi compilat prima eroare find semnalată la instrucțiunea:

```
cout << d.x;
```

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 B
{
protected:
    int a;

public:
    B() { a = 7; }
};

class D : public B
{
```

```
public:
    int b;
    D() { b = a + 7; }
};

int main()
{
    D d;
    cout << d.b;
    return 0;
}
```

Rezolvare:

Programul compilează. Acesta afișează valoarea 14.

Explicație:

Instrucțiunea `D d` creează o *instanță* a clasei D.

Clasa D moștenește în mod **public** clasa B, astfel știm că în *constructorul* din *clasa derivată (D)* este apelat *constructorul* din *clasa de BAZĂ (B)*, deci câmpul de date `a` primește valoarea 7.

Apoi execuția se continuă cu *constructorul clasei D* care poate accesa câmpul de date `a` din clasa B, deoarece acesta este declarat ca find **protected** în clasa B, iar tipul moștenirii este **public** deci va rămâne **protected** în clasa D.

În *constructorul clasei D*, câmpul de date **public** `b` primește valoarea câmpului de date **protected** `a` deci 7 plus valoarea 7.

Așadar, `b` primește valoarea 14. `b` find un câmp de date **public** poate fi acesat de către orice *instanță* a clasei D.

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 cls1
{
protected:
    int x;

public:
    cls1(int i = 10) { x = i; }
    int get_x() { return x; }
};
class cls2 : cls1
```

```
{
public:
    cls2(int i) : cls1(i) {}
};
int main()
{
    cls2 d(37);
    cout << d.get_x();
    return 0;
}
```

Rezolvare:

Programul nu compilează. Programul nu este corect din cauza *tipului de moștenire* folosit în definirea clasei `cls2`.

Clasa `cls2` moștenește în mod **private implicit** clasa `cls1`, astfel totul din *clasa de BAZĂ (`cls1`)* devine **private** în *clasa DERIVATĂ (`cls2`)*, deci metoda `get_x()` definită în clasa de BAZĂ (`cls1`) nu este *accesibilă* printr-o *instanță* a clasei DERIVATE (`cls2`).

Pentru ca programul să compileze schimbăm tipul de moștenire în **public**:

```
#include <iostream>
using namespace std;
class cls1
{
protected:
    int x;

public:
    cls1(int i = 10) { x = i; }
    int get_x() { return x; }
};
class cls2 : public cls1
{
public:
    cls2(int i) : cls1(i) {}
};
int main()
{
    cls2 d(37);
    cout << d.get_x();
    return 0;
}
```

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 B1
{
public:
    int x;
};
class B2
{
    int y;
};
class B3
{
public:
    int z;
};
class B4
{
public:
    int t;
};
class D : private B1, protected B2, public B3, B4
{
    int u;
};
int main()
{
    D d;
    cout << d.u;
    cout << d.x;
    cout << d.y;
    cout << d.z;
    cout << d.t;
    return 0;
}
```

Rezolvare:

Programul nu compilează. Programul nu este corect, în primul rând, din cauza câmpului de date `u` declarat în mod **private** în definiția clasei `D`.

În al doilea rând, din cauza *tipurilor de moștenire* folosite în definirea clasei `D`.

Astfel, compilatorul va semnala erori pentru următoarele instrucțiuni:

```
cout << d.u;
cout << d.x;
cout << d.y;
cout << d.t;
```

Explicație:

Acest lucru se întâmplă pentru că:

- clasa D conține un câmp **private** (u) care se încearcă a fi accesat printr-o *instanță* a acestei clase;
- clasa D *moștenește* clasa B1 în mod **private explicit**, iar această moștenire presupune că totul din *clasa de BAZĂ (B1)* devine **private** în *clasa DERIVATĂ (D)* ;
- clasa D *moștenește* clasa B2 în mod **protected**, iar această moștenire presupune că totul din *clasa de BAZĂ (B2)* devine **protected** în *clasa DERIVATĂ (D)* ;
- clasa D *moștenește* clasa B4 în mod **private implicit** (fără a preciza în mod explicit tipul de moștenire), iar această moștenire presupune că totul din *clasa de BAZA (B4)* devine **private** în *clasa DERIVATĂ (D)*.

Având în vedere punctele menționate mai sus, programul nu compilează deoarece câmpurile de date u, x, t sunt **private**, iar câmpul de date y este **protected** în clasa D, deci *nu sunt accesibile în mod direct printr-o instanță a clasei D.*

Pentru a face programul să compileze și pentru a nu îi schimba funcționalitatea putem moșteni cele 4 clase în mod **public** și să schimbăm *modificatorii de access* din clasele B2 și D din **private** în **public** astfel:

```
#include <iostream>
using namespace std;
class B1
{
public:
    int x;
};
class B2
{
    public:
    int y;
};
class B3
{
public:
    int z;
};
class B4
{
public:
    int t;
};
class D : public B1, public B2, public B3, public B4
{
    public:
    int u;
};
int main()
{
```

```
    D d;
    cout << d.u;
    cout << d.x;
    cout << d.y;
    cout << d.z;
    cout << d.t;
    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
{
    int x;

public:
    B(int i = 0) { x = i; }
};
class D : public B
{
public:
    D() : B(15) {}
    int f() { return x; }
};
int main()
{
    D d;
    cout << d.f();
    return 0;
}
```

Rezolvare:

Programul nu compilează. Programul nu este corect din cauza câmpului de date declarat în *clasa de BAZĂ (`B`)* în mod **private** folosit în *clasa DERIVATĂ (`D`)*.

Explicație:

Clasa `D` *moștenește* în mod **public** clasa `B`, astfel toate câmpurile din clasa de BAZĂ (`B`) își păstrează *modificatorii de acces*, mai puțin câmpul de date `x` care devine **inaccesibil** în clasa DERIVATĂ (`D`).

De aceea, în momentul definirii metodei `f()` din clasa D, compilatorul semnalează o *eroare de compilare*.

Pentru a rezolva această eroare putem schimba *modificatorul de acces* al câmpului de date `x`:

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

public:
    B(int i = 0) { x = i; }
};
class D : public B
{
public:
    D() : B(15) {}
    int f() { return x; }
};
int main()
{
    D d;
    cout << d.f();
    return 0;
}
```

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 A
{
protected:
    int x;

public:
    A(int i = 14) { x = i; }
};
class B : A
{
public:
    B(B &b)
    {
        x = b.x;
    }
    void afisare()
    {
        cout << x;
    }
```

```
};
int main()
{
    B b1, b2(b1);
    b2.afisare();
    return 0;
}
```

Rezolvare:

Programul nu compilează. Programul nu funcționează deoarece în momentul declarării *constructorului de copiere* în mod explicit *constructorul implicit* definit de către compilator devine **inaccesibil**.

Pentru a rezolva această problemă adaugăm un *constructor fără parametri* in clasa B:

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

public:
    A(int i = 14) { x = i; }
};
class B : A
{
public:
    B() {}
    B(B &b)
    {
        x = b.x;
    }
    void afisare()
    {
        cout << x;
    }
};
int main()
{
    B b1, b2(b1);
    b2.afisare();
    return 0;
}
```

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
{
protected:
    int x;

public:
    A(int i = 14) { x = i; }
};
class B : A
{
public:
    B() : A(2) {}
    B(B &b) { x = b.x - 14; }
    void afisare() { cout << x; }
};
int main()
{
    B b1, b2(b1);
    b2.afisare();
    return 0;
}
```

Rezolvare:

Programul compilează. Programul afișează valoarea -12.

Explicație:

Instrucțiunea `B b1` creează o *instanță* a clasei `B`. Clasa `B` *moștenește* în mod **private** clasa `A`, astfel câmpul de date `x` declarat **protected** în clasa `A` devine **private** în clasa `B` **(poate fi în continuare accesat)**.

În momentul *instanțierii obiectului* `b1` se apelează, în primul rând, *constructorul* clasei `A` cu valorea `2`, astfel `x` primește valoarea `2`, iar, în al doilea rând,*constructorul* clasei `B` care nu are nici un efect asupra datelor obiectului.

Observăm că obiectul `b2` este *instanțiat* ca o *copie* a obiectului `b1` folosind *constructorul de copiere* definit în clasa `B`, deci `b2.x` primește valoarea `b1.x` (`2`) din care scădem valoarea `14` astfel `b2.x` devine `-12`.

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
{
```

```
    int x;

public:
    A(int i) : x(i) {}
    int get_x() { return x; }
};
class B : public A
{
    int y;

public:
    B(int i, int j) : y(i), A(j) {}
    int get_y() { return y; }
};
class C : protected B
{
    int z;

public:
    C(int i, int j, int k) : z(i), B(j, k) {}
    int get_z() { return z; }
};
int main()
{
    C c(1, 2, 3);
    cout << c.get_x() + c.get_y() + c.get_z();
    return 0;
}
```

Rezolvare:

Programul nu compilează. Programul nu funcționează din cauza *tipului de moștenire* utilizat în clasa `C`.

Explicație:

Clasa `B` *moștenește* în mod **public** clasa `A`, astfel metoda `get_x()` definită în clasa `A` rămâne **accesibilă** de către orice *instanță* a clasei `B`.

Clasa `C` *moștenește* în mod **protected** clasa `B`, deci metodele `get_x()` și `get_y()` devin **protected**.

Putem rezolva această eroare schimbând *tipul de moștenire* utilizat în definirea clasei `C` astfel:

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

public:
    A(int i) : x(i) {}
    int get_x() { return x; }
};
```

```
class B : public A
{
    int y;

public:
    B(int i, int j) : y(i), A(j) {}
    int get_y() { return y; }
};
class C : public B
{
    int z;

public:
    C(int i, int j, int k) : z(i), B(j, k) {}
    int get_z() { return z; }
};
int main()
{
    C c(1, 2, 3);
    cout << c.get_x() + c.get_y() + c.get_z();
    return 0;
}
```

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 Base1
{
public:
    Base1()
    {
        cout << " Base1's constructor called" << endl;
    }
};

class Base2
{
public:
    Base2()
    {
        cout << "Base2's constructor called" << endl;
    }
};

class Derived : public Base1, public Base2
```

```
{
public:
    Derived()
    {
        cout << "Derived's constructor called" << endl;
    }
};

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

Rezolvare:

Programul compilează. Programul afisează:

```
Base1’s constructor called
Base2’s constructor called
Derived’s constructor called
```

Explicație:

În cazul *moștenirii multiple*, *constructorii* claselor de bază sunt întotdeauna apelați în ordinea de derivare de la **stânga la dreapta**, iar *destructorii* sunt apelați în **ordine inversă**.

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 P
{
public:
    void print()
    {
        cout << " Inside P::";
    }
};

class Q : public P
{
public:
    void print()
```

```
    {
        cout << " Inside Q";
    }
};

class R : public Q
{
};

int main(void)
{
    R r;

    r.print();
    return 0;
}
```

Rezolvare:

Programul compilează. Programul afișează:

```
Inside Q
```

Explicație:

Metoda `print` nu este definită în clasa R. Deci este căutată în *ierarhia de moștenire*.

`print ()` este prezentă în ambele clase P și Q. *Care dintre ele trebuie apelată?*

Dacă există *moștenire pe mai multe niveluri*, atunci metoda este căutată liniar în *ierarhia moștenirii* până când se găsește **prima** metodă care se potrivește.

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
{
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. Clasa B *moștenește* în mod **private** clasa A.

Explicație:

Deoarece se utilizează *tipul de moștenire* **private**, toți membri clasei A devin **private** în clasa B.

Clasa C este o clasă *moștenită* de clasa B. O clasă moștenită nu poate accesa datele membre **private** din clasa părinte, dar metoda `print()` din clasa C încearcă să acceseze un membru **private**, de aceea primim *eroarea*.

```
#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 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 Base
{
protected:
    int x;

public:
    Base(int i) { x = i; }
};

class Derived : public Base
{
public:
    Derived(int i) : x(i) {}
    void print() { cout << x; }
};

int main()
{
    Derived d(10);
    d.print();
}
```

Rezolvare:

Programul nu compilează. În programul de mai sus, x este un câmp de date declarat ca **protected**, deci este **accesibil** în *clasa derivată*.

Constructorul clasei derivate încearcă să utilizeze *lista de inițializare* a constructorului pentru a-l inițializa pe x, ceea ce **nu este permis** chiar dacă x este **accesibil**.

Membri clasei de bază pot fi *inițializați* doar printr-un **constructor al clasei de bază**.

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

```
public:
    Base(int i) { x = i; }
};

class Derived : public Base
{
public:
    Derived(int i) : Base(i) {}
    void print() { cout << x; }
};

int main()
{
    Derived d(10);
    d.print();
}
```