UNIVERSITATEA DE STAT DIN MOLDOVA

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

DEPARTAMENTUL INFORMATICĂ

Lupascu Iulii

Lucrul individual

la disciplina „POO”

Coordonator: Cemîrtan Cristian *Nume, prenume grad științific*

Chișinău, 2025

Cuprins

1-12: Lucrarea de laborator nr.1

12-25: Lucrarea de laborator nr.2a

25-44: Lucrarea de laborator nr.2b

44-49: Lucrare de laborator nr.3

49-54: Lucrare de laborator nr.4

Lucrarea de laborator nr.1

16. Se consideră clasa Apartament care conține membrii privat: adresa, numărul de odăi, suprafața, prețul și membrii public: un constructor fără parametri, un constructor cu parametri, distructor, funcție(ții) care permite citirea de la tastatură a valorilor membrilor private și funcție(ții) care permite afișarea la ecran a valorilor membrilor private. Opțional, clasa poate conține și alți membri public dacă studentul decide că este necesar. În funcția main() este declarat un tablou de obiecte a clasei Apartament (tabloul poate fi unul static sau dinamic, la dorință). Tot în funcția main() este descris următorul meniu (fiecare punct din meniu conține o funcție care execută fiecare din cerințele formulate mai jos): 1. citirea de la tastatura a valorilor membrilor private a obiectelor tabloului definit; 2. afișarea obiectelor tabloului la ecran; 3. sortarea obiectelor în ordine alfabetică după adresa; 4. afișarea la ecran a obiectelor pentru care se cunoaște că prețul este o mărime z (z se citește de la tastatură); 5. se adaugă un obiect nou pe poziția k în tabloul de obiecte; 6. se șterge obiectul pentru care se cunoaște că suprafața este o mărime p (p se citește de la tastatură).

```#include <iostream>
#include <algorithm>

using namespace std;

class Apartament {
private:
    int pret, numarul_odaii, suprafata;
    string adresa;
public:
    Apartament() {
        pret = 0;
        numarul_odaii = 0;
        suprafata = 0;
    }

    Apartament(const Apartament& a) {
        pret = a.pret;
```

```
 numarul_odaii = a.numarul_odaii;
        suprafata = a.suprafata;
        adresa = a.adresa;
    }

    Apartament(int a, int b, int c, string d) {
        pret = a;
        numarul_odaii = b;
        suprafata = c;
        adresa = d;
    }

    ~Apartament() {
        clog << "Adresa: " << adresa << "; Numarul odaii: " << numarul_odaii << "; Suprafata:
" << suprafata << "; Pretul: " << pret << ";" << endl;
    }

    void setPret(int pret) { this->pret = pret; }
    void setNumarulOdaii(int numarul_odaii) { this->numarul_odaii = numarul_odaii; }
    void setSuprafata(int suprafata) { this->suprafata = suprafata; }
    void setAdresa(string adresa) { this->adresa = adresa; }

    int getPret() const { return pret; }
    int getNumarulOdaii() const { return numarul_odaii; }
    int getSuprafata() const { return suprafata; }
    string getAdresa() const { return adresa; }

    void afiseaza() const {
```

```
cout << "Pretul: " << getPret() << endl;
        cout << "Numarul de odaie: " << getNumarulOdaii() << endl;
        cout << "Suprafata: " << getSuprafata() << endl;
        cout << "Adresa: " << getAdresa() << endl;
    }

    void introducere() {
        cout << "Introduceti adresa: ";
        cin >> adresa;
        cout << "Introduceti numarul de odăi: ";
        cin >> numarul_odaii;
        cout << "Introduceti suprafata (in metri patrati): ";
        cin >> suprafata;
        cout << "Introduceti pretul: ";
        cin >> pret;
    }
};

// Functie de comparare pentru sortare (ordonare alfabetica dupa adresa)
bool compareAdresa(const Apartament* a1, const Apartament* a2) {
    return a1->getAdresa() < a2->getAdresa();
}

void meniu() {
    const int MAX_APARTAMENTE = 100;  // Dimensiunea maxima a tabloului de
apartamente
    Apartament* apartamente[MAX_APARTAMENTE];
    int numar_apartamente = 0;  // Urmarim numarul de apartamente din tablou
```

```
    for (; ; ) {
        cout << "\nMeniu:\n";
        cout << "1. citirea de la tastatura a valorilor membrilor private a obiectelor tabloului definit\n";
        cout << "2. afișarea obiectelor tabloului la ecran\n";
        cout << "3. sortarea obiectelor în ordine alfabetică după adresa\n";
        cout << "4. afișarea la ecran a obiectelor pentru care se cunoaște că prețul este o mărime z (z se citește de la tastatură)\n";
        cout << "5. se adaugă un obiect nou pe poziția k în tabloul de obiecte\n";
        cout << "6. se șterge obiectul pentru care se cunoaște că suprafața este o mărime p (p se citește de la tastatură)\n";
        cout << "7. Stop\n";
        cout << "Alegeti optiunea: ";

        int optiune;
        cin >> optiune;

        switch (optiune) {
        case 1: {
            int n;
            cout << "Introduceti numarul de apartamente (maxim " << MAX_APARTAMENTE << "): ";
            cin >> n;

            // Verificăm că nu depășim dimensiunea maximă a tabloului
            if (n < 0 || numar_apartamente + n > MAX_APARTAMENTE) {
                int diff = MAX_APARTAMENTE - numar_apartamente;
                cout << "Numar invalid de apartamente. Puteti adauga maxim " << diff << " apartamente.\n";
                n = diff;
```

```
}

        for (int i = 0; i < n; i++) {
            apartamente[numar_apartamente] = new Apartament;
            apartamente[numar_apartamente]->introducere();
            numar_apartamente++;
        }
        break;
    }

    case 2:
        for (int i = 0; i < numar_apartamente; i++) {
            apartamente[i]->afiseaza();
            cout << "-------------" << endl;
        }
        break;

    case 3:
    // Sortăm apartamentele după adresă
    sort(apartamente, &apartamente[numar_apartamente], compareAdresa);

    // Afișăm apartamentele după sortare
    for (int i = 0; i < numar_apartamente; i++) {
        apartamente[i]->afiseaza();
        cout << "-------------" << endl;
    }
    cout << "Apartamentele au fost sortate dupa adresa.\n";
    break;
```

```
case 4: {
    int z;
    cout << "Introduceti pretul (z): ";
    cin >> z;
    for (int i = 0; i < numar_apartamente; i++) {
        if (apartamente[i]->getPret() == z) {
            apartamente[i]->afiseaza();
            cout << "----------" << endl;
        }
    }
    break;
}

case 5: {
    int k;
    cout << "Introduceti pozitia (k) pentru adaugarea unui nou apartament: ";
    cin >> k;

    // Verificăm dacă poziția este validă
    if (k >= 0 && k <= numar_apartamente && k < MAX_APARTAMENTE) {
        // Mutăm elementele pentru a face loc unui nou apartament
        for (int i = numar_apartamente; i > k; --i) {
            apartamente[i] = apartamente[i - 1];
        }

        apartamente[k] = new Apartament;
        apartamente[k]->introducere();
        numar_apartamente++;
    } else {
```

```
            cout << "Pozitie invalida.\n";
        }
        break;
    }

    case 6: {
        float p;
        cout << "Introduceti suprafata (p) pentru care doriti sa stergeti apartamentul: ";
        cin >> p;

        bool gasit = false;
        for (int i = 0; i < numar_apartamente; ++i) {
            if (apartamente[i]->getSuprafata() == p) {
                delete apartamente[i];
                // Mutăm elementele pentru a "șterge" apartamentul
                for (int j = i; j < numar_apartamente - 1; ++j) {
                    apartamente[j] = apartamente[j + 1];
                }
                numar_apartamente--;
                gasit = true;
                break;
            }
        }
        if (!gasit) {
            cout << "Nu s-a gasit apartamentul cu suprafata specificata.\n";
        }
break;
    }
```

```
        case 7:
            cout << "Stop.\n";

            for (int i = 0; i < numar_apartamente; ++i)
            {
                delete apartamente[i];
            }

            return;

        default:
            cout << "Optiune invalida.\n";
            break;
        }
    }
}

int main() {
    bool b;

    do
    {
        meniu();
        std::cout << "Reia?";
        std::cin >> b;
} while (b);

    return 0;
}```
```

```
Meniu:
1. citirea de la tastatura a valorilor membrilor private a obiectelor tabloului definit
2. afișarea obiectelor tabloului la ecran
3. sortarea obiectelor în ordine alfabetică după adresa
4. afișarea la ecran a obiectelor pentru care se cunoaște că prețul este o mărime z (z se citește de la tastatură)
5. se adaugă un obiect nou pe poziția k în tabloul de obiecte
6. se șterge obiectul pentru care se cunoaște că suprafața este o mărime p (p se citește de la tastatură)
7. Stop
Alegeti optiunea: 1
Introduceti numarul de apartamente (maxim 100): 3
Introduceti adresa: Chisinau
Introduceti numarul de odăi: 3
Introduceti suprafata (in metri patrati): 50
Introduceti pretul: 50000
Introduceti adresa: Balti
Introduceti numarul de odăi: 2
Introduceti suprafata (in metri patrati): 30
Introduceti pretul: 20000
Introduceti adresa: Amsterdam
Introduceti numarul de odăi: 1
Introduceti suprafata (in metri patrati): 25
Introduceti pretul: 100000
```

```
Meniu:
1. citirea de la tastatura a valorilor membrilor private a obiectelor tabloului definit
2. afișarea obiectelor tabloului la ecran
3. sortarea obiectelor în ordine alfabetică după adresa
4. afișarea la ecran a obiectelor pentru care se cunoaște că prețul este o mărime z (z se citește de la tastatură)
5. se adaugă un obiect nou pe poziția k în tabloul de obiecte
6. se șterge obiectul pentru care se cunoaște că suprafața este o mărime p (p se citește de la tastatură)
7. Stop
Alegeti optiunea: 2
Pretul: 50000
Numarul de odaie: 3
Suprafata: 50
Adresa: Chisinau
-------------
Pretul: 20000
Numarul de odaie: 2
Suprafata: 30
Adresa: Balti
-------------
Pretul: 100000
Numarul de odaie: 1
Suprafata: 25
Adresa: Amsterdam
-------------
```

```
Meniu:
1. citirea de la tastatura a valorilor membrilor private a obiectelor tabloului definit
2. afișarea obiectelor tabloului la ecran
3. sortarea obiectelor în ordine alfabetică după adresa
4. afișarea la ecran a obiectelor pentru care se cunoaște că prețul este o mărime z (z se citește de la tastatură)
5. se adaugă un obiect nou pe poziția k în tabloul de obiecte
6. se șterge obiectul pentru care se cunoaște că suprafața este o mărime p (p se citește de la tastatură)
7. Stop
Alegeti optiunea: 3
Pretul: 100000
Numarul de odaie: 1
Suprafata: 25
Adresa: Amsterdam
-------------
Pretul: 20000
Numarul de odaie: 2
Suprafata: 30
Adresa: Balti
-------------
Pretul: 50000
Numarul de odaie: 3
Suprafata: 50
Adresa: Chisinau
-------------
Apartamentele au fost sortate dupa adresa.
```

```
Meniu:
1. citirea de la tastatura a valorilor membrilor private a obiectelor tabloului definit
2. afișarea obiectelor tabloului la ecran
3. sortarea obiectelor în ordine alfabetică după adresa
4. afișarea la ecran a obiectelor pentru care se cunoaște că prețul este o mărime z (z se citește de la tastatură)
5. se adaugă un obiect nou pe poziția k în tabloul de obiecte
6. se șterge obiectul pentru care se cunoaște că suprafața este o mărime p (p se citește de la tastatură)
7. Stop
Alegeti optiunea: 2
Pretul: 100000
Numarul de odaie: 1
Suprafata: 25
Adresa: Amsterdam
-------------
Pretul: 20000
Numarul de odaie: 2
Suprafata: 30
Adresa: Balti
-------------
Pretul: 50000
Numarul de odaie: 3
Suprafata: 50
Adresa: Chisinau
-------------
```

```
Meniu:
1. citirea de la tastatura a valorilor membrilor private a obiectelor tabloului definit
2. afișarea obiectelor tabloului la ecran
3. sortarea obiectelor în ordine alfabetică după adresa
4. afișarea la ecran a obiectelor pentru care se cunoaște că prețul este o mărime z (z se citește de la tastatură)
5. se adaugă un obiect nou pe poziția k în tabloul de obiecte
6. se șterge obiectul pentru care se cunoaște că suprafața este o mărime p (p se citește de la tastatură)
7. Stop
Alegeti optiunea: 4
Introduceti pretul (z): 50000
Pretul: 50000
Numarul de odaie: 3
Suprafata: 50
Adresa: Chisinau
-------------
```

```
Meniu:
1. citirea de la tastatura a valorilor membrilor private a obiectelor tabloului definit
2. afișarea obiectelor tabloului la ecran
3. sortarea obiectelor în ordine alfabetică după adresa
4. afișarea la ecran a obiectelor pentru care se cunoaște că prețul este o mărime z (z se citește de la tastatură)
5. se adaugă un obiect nou pe poziția k în tabloul de obiecte
6. se șterge obiectul pentru care se cunoaște că suprafața este o mărime p (p se citește de la tastatură)
7. Stop
Alegeti optiunea: 5
Introduceti pozitia (k) pentru adaugarea unui nou apartament: 2
Introduceti adresa: Paris
Introduceti numarul de odăi: 2
Introduceti suprafata (in metri patrati): 30
Introduceti pretul: 300000
```

```
Meniu:
1. citirea de la tastatura a valorilor membrilor private a obiectelor tabloului definit
2. afișarea obiectelor tabloului la ecran
3. sortarea obiectelor în ordine alfabetică după adresa
4. afișarea la ecran a obiectelor pentru care se cunoaște că prețul este o mărime z (z se citește de la tastatură)
5. se adaugă un obiect nou pe poziția k în tabloul de obiecte
6. se șterge obiectul pentru care se cunoaște că suprafața este o mărime p (p se citește de la tastatură)
7. Stop
Alegeti optiunea: Pretul: 50000
Numarul de odaie: 3
Suprafata: 50
Adresa: Chisinau
-------------
Pretul: 25000
Numarul de odaie: 2
Suprafata: 30
Adresa: Balti
-------------
Pretul: 300000
Numarul de odaie: 2
Suprafata: 30
Adresa: Paris
-------------
Pretul: 100000
Numarul de odaie: 1
Suprafata: 20
Adresa: Amsterdam
-------------
```

```
Meniu:
1. citirea de la tastatura a valorilor membrilor private a obiectelor tabloului definit
2. afișarea obiectelor tabloului la ecran
3. sortarea obiectelor în ordine alfabetică după adresa
4. afișarea la ecran a obiectelor pentru care se cunoaște că prețul este o mărime z (z se citește de la tastatură)
5. se adaugă un obiect nou pe poziția k în tabloul de obiecte
6. se șterge obiectul pentru care se cunoaște că suprafața este o mărime p (p se citește de la tastatură)
7. Stop
Alegeti optiunea: 6
Introduceti suprafata (p) pentru care doriti sa stergeti apartamentul: 30
Adresa: Balti; Numarul odaii: 2; Suprafata: 30; Pretul: 20000;
```

```
Meniu:
1. citirea de la tastatura a valorilor membrilor private a obiectelor tabloului definit
2. afișarea obiectelor tabloului la ecran
3. sortarea obiectelor în ordine alfabetică după adresa
4. afișarea la ecran a obiectelor pentru care se cunoaște că prețul este o mărime z (z se citește de la tastatură)
5. se adaugă un obiect nou pe poziția k în tabloul de obiecte
6. se șterge obiectul pentru care se cunoaște că suprafața este o mărime p (p se citește de la tastatură)
7. Stop
Alegeti optiunea: 7
Stop.
Adresa: Chisinau; Numarul odaii: 3; Suprafata: 50; Pretul: 50000;
Adresa: Paris; Numarul odaii: 2; Suprafata: 20; Pretul: 300000;
Adresa: Amsterdam; Numarul odaii: 1; Suprafata: 20; Pretul: 100000;
Reia?Nu
```

Lucrarea de laborator nr. 2a

Se consideră ierarhia de clase din schemă (fiecare student are o schemă care corespunde variantei). Clasa de bază conține 4 membri 2 dintre care protected sunt moșteniți de clasele derivate xxx și yyy (xxx și yyy sunt denumiri de clase simbolice, ceea ce înseamnă că studentul va înlocui aceste valori după cum crede de cuviință). Fiecare clasă xxx și yyy va conține și câte 2 membri private proprii (pe lângă cei 2 membri moșteniți de la clasa de bază). Membrii public ai fiecărei clase din ierarhie sunt: un constructor fără parametri, un constructor cu parametri, funcție(ții) care permite citirea de la tastatură a valorilor membrilor private și funcție(ții) care permite afișarea la ecran a valorilor membrilor private. Opțional, clasa poate conține și alți membri public dacă studentul decide că este necesar. În funcția main() este declarat câte un tablou de obiecte a fiecărei clase din ierarhie (tablourile pot fi statice sau dinamice, la dorință). Tot în funcția main() este posibilitatea alegerii cu care tablou se va lucra și respectiv, pentru acel tablou este descris un meniu (fiecare punct din meniu conține o funcție care execută fiecare din cerințele formulate mai jos): 1. citirea de la tastatură a valorilor membrilor obiectelor tabloului definit; 2. afișarea obiectelor tabloului la ecran; 3. sortarea obiectelor în ordine definită de student; 4. afișarea la ecran a obiectelor pentru care se cunoaște că un câmp satisface o condiție formulată de student; 5. se modifică date obiectului pe poziția k în tabloul de obiecte (k se citește de la tastatură); 6. se șterge obiectul pentru care se cunoaște că un câmp satisface o condiție formulată de student.

Pacient numele, diagnoza, data internării, numele medicului xxx câmp1_moștenit, câmp2_moștenit, câmp1x, câmp2x yyy câmp1_moștenit, câmp2_moștenit, câmp1y, câmp2y

```#include <iostream>

#include <algorithm>


using namespace std;
```

```
class Pacient {
private:
   string diagnoza;
   int varsta;
protected:
   string numele, numele_medicului;
public:
   Pacient() {
numele = "";
      numele_medicului = "";
   }

   Pacient(string x, string y) {
      numele = x;
      numele_medicului = y;
   }

   virtual ~Pacient() {
      clog << "Numele pacientului: " << numele << "; Numele medicului: " <<
numele_medicului << ";" << endl;
   }

   void setNumele(string numele) { this->numele = numele; }
   void setNumeMedic(string numele_medicului) { this->numele_medicului =
numele_medicului; }

   string getNumele() const { return numele; }
   string getNumeMedic() const { return numele_medicului; }
```

```
    virtual void citire() {
        cout << "Introduceti numele pacientului: " << endl;
        cin >> numele;
        cout << "Introduceti numele doctorului: " << endl;
        cin >> numele_medicului;
    }
virtual void afisare() const {
        cout << "numele pcientului: " << numele << "; numele dr.: " << numele_medicului << ";
" << endl;
    }
};

class Pprioritar : public Pacient {
private:
    int cost;
    string data_iternarii;
public:
    Pprioritar(): Pacient() {
        cost = 0;
        data_iternarii = "";
    }
    Pprioritar(string x, string y, int c, string d): Pacient(x,y) {
        cost = c;
        data_iternarii = d;
    }

    ~Pprioritar() {
        clog << "Cost: " << cost << "; Data iternarii: " << data_iternarii << ";\n";
    }
```

```
    void setCost(int cost) { this->cost = cost; }
    void setTipOp(string data_iternarii) { this->data_iternarii = data_iternarii; }

    int getCost() const { return cost; }
string getTipOp() const { return data_iternarii; }

    void citire() override {
        Pacient::citire();
        cout << "Introduceti costul: " << endl;
        cin >> cost;
        cout << "Introduceti data iternarii: " << endl;
        cin >> data_iternarii;
    }

    void afisare() const override {
        Pacient::afisare();
        cout << "costul: " << cost << "; data iternarii: " << data_iternarii << "; " << endl;
    }
};

class Pobisnuit : public Pacient {
private:
    int cost;
    string data_iternarii;
public:
Pobisnuit() : Pacient() {
        cost = 0;
        data_iternarii = "";
```

```
    }
    Pobisnuit(string x, string y, int c, string d): Pacient(x,y) {
        cost = c;
        data_iternarii = d;
}

    ~Pobisnuit() {
        clog << "Cost (pacient obisnuit): " << cost << "; data iternarii (pacient obisnuit): " <<
data_iternarii << ";\n";
    }

    void setCost(int cost) { this->cost = cost; }
    void setTipOp(string data_iternarii) { this->data_iternarii = data_iternarii; }

    int getCost() const { return cost; }
    string getTipOp() const { return data_iternarii; }

    void citire() override {
        Pacient::citire();
        cout << "Introduceti costul: " << endl;
        cin >> cost;
        cout << "Introduceti data iternarii: " << endl;
        cin >> data_iternarii;
    }

    void afisare() const override {
        Pacient::afisare();
        cout << "costul: " << cost << "; data iternarii: " << data_iternarii << "; " << endl;
    }
```

```
};

void meniu() {
int maxPacienti = 100;
   Pacient* pacienti[maxPacienti];  // Tablou de pointeri la Pacient
   int indexPacient = 0;

   for (;;) {
      cout << "\nMeniu:\n";
      cout << "1. citirea de la tastatură a valorilor membrilor obiectelor tabloului definit;\n";
      cout << "2. afișarea obiectelor tabloului la ecran;\n";
      cout << "3. sortarea obiectelor în ordine definită de student;\n";
      cout << "4. afișarea la ecran a obiectelor pentru care se cunoaște că un câmp satisface o condiție formulată de student;\n";
      cout << "5. se modifică date obiectului pe poziția k în tabloul de obiecte (k se citește de la tastatură);\n";
      cout << "6. se șterge obiectul pentru care se cunoaște că un câmp satisface o condiție formulată de student;\n";
      cout << "7. Stop;\n";
      cout << "Alegeti optiunea: ";

      int optiune;
      cin >> optiune;

      switch (optiune) {
         case 1: {
            int n;
            cout << "Introduceti numarul de pacienti (maxim " << maxPacienti << "): ";
            cin >> n;
```

```
        if (n < 0 || n > maxPacienti) {
            cout << "Numar invalid de pacienti. Puteti adauga maxim " << maxPacienti << " pacienti.\n";
            n = maxPacienti;
        }

        if (indexPacient + n > maxPacienti) {
            cout << "Nu mai sunt locuri disponibile pentru alți pacienți.\n";
            n = maxPacienti - indexPacient;
        }
    int alegere;
    cout << "Este pacientul 1)Obisnuit; sau 2)Prioritar" << endl;
    cin >> alegere;
    while (alegere!=1 || alegere!=2) {
    if (alegere==2) {
        for (int i = 0; i < n; i++) {
            pacienti[indexPacient] = new Pprioritar;
            pacienti[indexPacient]->citire();
            indexPacient++;
        }
    }
    else if (alegere==1) {
        for (int i = 0; i < n; i++) {
            pacienti[indexPacient] = new Pobisnuit;
            pacienti[indexPacient]->citire();
            indexPacient++;
        }
    }
    else {
```

```
cout << "Introdu o valoare corecta" << endl;
            }
            }
                break;
            }

        case 2: {
            cout << "\nAfisare pacienți:\n";
            for (int i = 0; i < indexPacient; i++) {
                pacienti[i]->afisare();
                cout << endl;
            }
            break;
        }

        case 3: {
//Dupa nume
            sort(pacienti, pacienti + indexPacient, [](Pacient* a, Pacient* b) {
                return a->getNumele() < b->getNumele();
            });

            cout << "\nPacienți sortați:\n";
            for (int i = 0; i < indexPacient; i++) {
                pacienti[i]->afisare();
                cout << endl;
            }
            break;
        }
        case 4: {
```

```
        int bugetobisnuiti, bugetprioritari;
        cout << "Introduceti bugetul de care dispun pacientii obisnuiti si apoi cei prioritari: ";
        cin >> bugetobisnuiti >> bugetprioritari;

        for (int i = 0; i < indexPacient; i++) {
            pacienti[i]->afisare(); // Afișăm informațiile pacientului

            // Pentru pacientii de tip 'xxx', verificăm costul
            if (Pprioritar* pacientX = dynamic_cast<Pprioritar*>(pacienti[i])) {
                if (bugetprioritari >= pacientX->getCost()) {
                    cout << "Pacientul prioritar are bani de operatie" << endl;
                }
                else {
                    cout << "Pacientul prioritar nu are bani de operatie" << endl;
                }
            }

            // Pentru pacientii de tip 'yyy', verificăm varsta
            if (Pobisnuit* pacientY = dynamic_cast<Pobisnuit*>(pacienti[i])) {
                if (bugetobisnuiti <= pacientY->getCost()) {
                    cout << "Pacientul obisnuit are bani de operatie" << endl;
                }
                else {
                    cout << "Pacientul obisnuit nu are bani de operatie" << endl;
                }
            }
            cout << "-------------" << endl;
    }
        break;
```

```
}

case 5: {
    int k;
    cout << "Introduceti pozitia (k) pentru schimbarea acestui obiect: ";
    cin >> k;

    // Verificăm dacă poziția este validă
    if (k >= 0 && k < indexPacient) {
        pacienti[k]->citire(); // Citim datele pacientului la pozitia k
    } else {
        cout << "Pozitie invalida!\n";
    }
    break;
}

case 6: {
    string p;
    cout << "Introduceti numele (p) pentru care doriti sa stergeti obiectul: ";
    cin >> p;

    bool gasit = false;
    for (int i = 0; i < indexPacient; ++i) {
        // Verificăm pentru pacientii prioritari
        if (Pprioritar* pacientX = dynamic_cast<Pprioritar*>(pacienti[i])) {
            if (pacientX->getTipOp() == p) {
                delete pacienti[i];
                // Mutăm elementele pentru a "șterge" pacientul
                for (int j = i; j < indexPacient - 1; ++j) {
```

```
                pacienti[j] = pacienti[j + 1];
            }
            indexPacient--;
            gasit = true;
            break;  // Ieșim din buclă
        }
    }

    // Verificăm pentru pacientii de tip 'yyy'
    if (Pobisnuit* pacientY = dynamic_cast<Pobisnuit*>(pacienti[i])) {
        if (pacientY->getTipOp() == p) {
            delete pacienti[i];
            // Mutăm elementele pentru a "șterge" pacientul
            for (int j = i; j < indexPacient - 1; ++j) {
                pacienti[j] = pacienti[j + 1];
            }
            indexPacient--;
            gasit = true;
            break;  // Ieșim din buclă
        }
    }
}
if (!gasit) {
    cout << "Nu s-a gasit obiectul cu valoarea specificata.\n";
}
break;
}

case 7:
```

```
            cout << "Stop.\n";

            for (int i = 0; i < indexPacient; ++i)
            {
                delete pacienti[i];
            }

            return;

        default:
            cout << "Optiune invalida.\n";
            break;
        }
    }
}

int main() {
    bool b;
    do
    {
        meniu();
        cout << "Reia?";
        cin >> b;
    } while (b);
    return 0;
}```
```

Meniu:

1. citirea de la tastatură a valorilor membrilor obiectelor tabloului definit;
2. afișarea obiectelor tabloului la ecran;

3. sortarea obiectelor în ordine definită de student;

4. afișarea la ecran a obiectelor pentru care se cunoaște că un câmp satisface o condiție formulată de student;

5. se modifică date obiectului pe poziția k în tabloul de obiecte (k se citește de la tastatură);

6. se șterge obiectul pentru care se cunoaște că un câmp satisface o condiție formulată de student;

7. Stop;

Alegeti optiunea: 1

Introduceti numarul de pacienti (maxim 100): 2

Este pacientul 1)Obisnuit; sau 2)Prioritar

2

Introduceti numele pacientului:

John

Introduceti numele doctorului:

Dr. Smith

Introduceti costul:

500

Introduceti data iternarii:

2025-05-01

Este pacientul 1)Obisnuit; sau 2)Prioritar

1

Introduceti numele pacientului:

Jane

Introduceti numele doctorului:

Dr. Doe

Introduceti costul:

300

Introduceti data iternarii:

2025-05-02

Afișare pacienți:

numele pcientului: John; numele dr.: Dr. Smith;

costul: 500; data iternarii: 2025-05-01;

numele pcientului: Jane; numele dr.: Dr. Doe;

costul: 300; data iternarii: 2025-05-02;

Sortare pacienți:

Pacienți sortați:

numele pcientului: Jane; numele dr.: Dr. Doe;

costul: 300; data iternarii: 2025-05-02;

numele pcientului: John; numele dr.: Dr. Smith;

costul: 500; data iternarii: 2025-05-01;

Lucrul de laborator nr. 2b

Se consideră ierarhia de clase din schema de mai jos. Clasa de bază conține 4 membri 2 dintre care protected (câmp1_moștenit, câmp2_moștenit) sunt moșteniți de clasele derivate xxx și yyy (xxx și yyy sunt denumiri de clase simbolice, ceea ce înseamnă că studentul va înlocui aceste valori după cum crede de cuviință). Fiecare clasă xxx și yyy vor conține și câte 2 membri private proprii (pe lângă cei 2 membri moșteniți de la clasa de bază). Clasa zzz conține câmpurile câmp1_moștenit, câmp2_moștenit moștenite de la clasa de bază, un câmp câmp_moștenit_x moștenit de la clasa xxx, un câmp câmp_moștenit_y moștenit de la clasa yyy și un câmp private propriu câmp_z. Opțional, fiecare clasă poate conține și alți membri public dacă studentul decide că este necesar. La ierarhia descrisă se adaogă și o clasă abstractă de la care derivă clasa de bază descrisă în schemă. Membrii public ai fiecărei clase din ierarhie sunt: un constructor fără parametri, un constructor cu parametri, funcție(ții) care permite citirea de la tastatură a valorilor membrilor private și protected și funcție(ții) care permite afișarea la ecran a valorilor membrilor private și protected. În funcția main() este declarat un tablou dinamic de obiecte care va conține obiecte a tuturor claselor din ierarhie.

Tot în funcția main() este descris un meniu (fiecare punct din meniu conține o funcție care execută fiecare din cerințele formulate mai jos): 1. citirea de la tastatură a valorilor membrilor obiectelor diferitor clase din ierarhie în tabloul definit; 2. afișarea obiectelor tabloului la ecran; 3. sortarea obiectelor în ordine definită de student după un câmp comun tuturor claselor din ierarhie; 4. afișarea la ecran a obiectelor pentru care se cunoaște că un câmp comun tuturor claselor din ierarhie satisface o condiție formulată de student; 5. se modifică datele obiectului pe poziția k în tabloul de obiecte (k se citește de la tastatură); 6. se șterge obiectul pentru care se cunoaște că un câmp comun tuturor claselor din ierarhie satisface o condiție formulată de student.

Pacient numele, diagnoza, data internării, numele medicului xxx câmp1_moștenit, câmp2_moștenit, câmp1x, câmp2x yyy câmp1_moștenit, câmp2_moștenit, câmp1y, câmp2y zzz câmp1_moștenit, câmp2_moștenit, câmp_moștenit_x, câmp_moștenit_y, câmp_z

```#include <iostream>
#include <algorithm>

using namespace std;

class Pacient {
private:
    string diagnoza;
    int varsta;
protected:
    string numele, numele_medicului;
public:
    Pacient() {
        numele = "";
        numele_medicului = "";
    }

    Pacient(string x, string y) {
        numele = x;
numele_medicului = y;
    }

    virtual ~Pacient() {
        clog << "Numele pacientului: " << numele << "; Numele medicului: " <<
numele_medicului << ";" << endl;
    }

    void setNumele(string numele) { this->numele = numele; }
    void setNumeMedic(string numele_medicului) { this->numele_medicului =
numele_medicului; }
```

```
    string getNumele() const { return numele; }
    string getNumeMedic() const { return numele_medicului; }

    virtual int getCost() const = 0;
    virtual string getTipOp() const = 0;

    virtual void citire() {
        cout << "Introduceti numele pacientului: " << endl;
        cin >> numele;
        cout << "Introduceti numele doctorului: " << endl;
        cin >> numele_medicului;
    }

    virtual void afisare() const {
        cout << "numele pcientului: " << numele << "; numele dr.: " << numele_medicului << ";
" << endl;
    }
};
class Pprioritar : public virtual Pacient {
private:
    int cost;
    string data_iternarii;
public:
    Pprioritar(): Pacient() {
        cost = 0;
        data_iternarii = "";
    }
    Pprioritar(string x, string y, int c, string d): Pacient(x,y) {
```

```
        cost = c;
        data_iternarii = d;
    }

    ~Pprioritar() override {
        clog << "Cost: " << cost << "; Data iternarii: " << data_iternarii << ";\n";
    }

    void setCost(int cost) { this->cost = cost; }
    void setTipOp(string data_iternarii) { this->data_iternarii = data_iternarii; }

    int getCost() const override { return cost; }
    virtual string getTipOp() const override { return data_iternarii; }

    void citire() override {
        Pacient::citire();
        cout << "Introduceti costul: " << endl;
cin >> cost;
        cout << "Introduceti data iternarii: " << endl;
        cin >> data_iternarii;
    }

    void afisare() const override {
        Pacient::afisare();
        cout << "costul: " << cost << "; data iternarii: " << data_iternarii << "; " << endl;
    }

    virtual int CostX() const {return cost;}
};
```

```
class Pobisnuit : public virtual Pacient {
private:
   int cost;
   string data_iternarii;
public:
Pobisnuit() : Pacient() {
      cost = 0;
      data_iternarii = "";
   }
   Pobisnuit(string x, string y, int c, string d): Pacient(x,y) {
      cost = c;
      data_iternarii = d;
   }
~Pobisnuit() override {
      clog << "Cost (pacient obisnuit): " << cost << "; data iternarii (pacient obisnuit): " <<
data_iternarii << ";\n";
   }

   void setCost(int cost) { this->cost = cost; }
   void setTipOp(string data_iternarii) { this->data_iternarii = data_iternarii; }

   int getCost() const override { return cost; }
   virtual string getTipOp() const override { return data_iternarii; }

   void citire() override {
      Pacient::citire();
      cout << "Introduceti costul: " << endl;
      cin >> cost;
```

```
        cout << "Introduceti data iternarii: " << endl;
        cin >> data_iternarii;
    }

    void afisare() const override {
        Pacient::afisare();
        cout << "costul: " << cost << "; data iternarii: " << data_iternarii << "; " << endl;
    }

    virtual string DataIternariiY() const {return data_iternarii;}
};

class Copil : public Pprioritar, public Pobisnuit {
private:
bool bolnav;
public:
    Copil() {
        bolnav="";
    }

    Copil(bool z) {
        bolnav=z;
    }

    ~Copil() override {
        clog << "Copilul este bolnav " << bolnav << endl;
    }

    void setBolnav(bool copilas) {this->bolnav=copilas;}
```

```
bool getBolnav() const {return bolnav;}

int getCost() const override {return Pprioritar::getCost();}
string getTipOp() const override {return Pprioritar::getTipOp();}

void citire() override {
   Pobisnuit::citire();
}

void citire(bool estePrioritar) {
   if (estePrioritar) {
      Pprioritar::citire();
   }
   else {
      citire();
   }
}

void afisare() const override {
   Pobisnuit::afisare();
}

void afisare(bool estePrioritar) const {
   if (estePrioritar) {
      Pprioritar::afisare();
   }
   else {
      afisare();
   }
```

```
    }
};

void meniu() {
    int maxPacienti = 100;
    Pacient* pacienti[maxPacienti];  // Tablou de pointeri la Pacient
    int indexPacient = 0;

    for (;;) {
        cout << "\nMeniu:\n";
        cout << "1. citirea de la tastatură a valorilor membrilor obiectelor diferitor clase din ierarhie în tabloul definit;\n";
        cout << "2. afișarea obiectelor tabloului la ecran;\n";
        cout << "3. sortarea obiectelor în ordine definită de student după un câmp comun tuturor claselor din ierarhie;\n";
        cout << "4. afișarea la ecran a obiectelor pentru care se cunoaște că un câmp comun tuturor claselor din ierarhie satisface o conditie formulată de student;\n";
        cout << "5. se modifică datele obiectului pe poziția k în tabloul de obiecte (k se citește de la tastatură);\n";
        cout << "6. se șterge obiectul pentru care se cunoaște că un câmp comun tuturor claselor din ierarhiit::afisare();e satisface o condiție formulată de student;\n";
        cout << "7. Stop;\n";
        cout << "Alegeti optiunea: ";

        int optiune;
        cin >> optiune;

        switch (optiune) {
            case 1: {
                int n;
                cout << "Introduceti numarul de pacienti (maxim " << maxPacienti << "): ";
```

```
        cin >> n;

        while (n < 0 || n > maxPacienti) {
            cout << "Numar invalid de pacienti. Puteti adauga maxim " << maxPacienti << "
pacienti.\n";
            cin >> n;
        }

        if (indexPacient + n > maxPacienti) {
            cout << "Nu mai sunt locuri disponibile pentru alți pacienți.\n";
            n = maxPacienti - indexPacient;
        }
    int alegere;
    do {
    cout << "Este pacientul 1)Obisnuit; sau 2)Prioritar; sau 3)Copil;" << endl;
    cin >> alegere;
    if (alegere==2) {
        for (int i = 0; i < n; i++) {
            pacienti[indexPacient] = new Pprioritar;
            pacienti[indexPacient]->citire();
            indexPacient++;
        }
    }
    else if (alegere==1) {
        for (int i = 0; i < n; i++) {
            pacienti[indexPacient] = new Pobisnuit;
            pacienti[indexPacient]->citire();
            indexPacient++;
        }
```

```
    }
    else if (alegere==3) {
        for (int i = 0; i < n; i++) {
            Copil *copil = new Copil;
            pacienti[indexPacient] = copil;

            bool bolnav;
            cout << "Copilul dat este pacient prioritar?(da/nu): " << endl;
            cin >> bolnav;

            copil->citire(bolnav);
            indexPacient++;
        }
    }
    else {
        cout << "Introdu o valoare corecta" << endl;
    }
    } while (alegere!=1 && alegere!=2 && alegere!=3);
        break;
    }

case 2: {
    cout << "\nAfisare pacienți:\n";
    for (int i = 0; i < indexPacient; i++) {
        pacienti[i]->afisare();
        cout << endl;
    }
    break;
}
```

```
        case 3: {
//Dupa nume
            sort(pacienti, pacienti + indexPacient, [](Pacient* a, Pacient* b) {
                return a->getNumele() < b->getNumele();
            });

            cout << "\nPacienți sortați:\n";
            for (int i = 0; i < indexPacient; i++) {
                pacienti[i]->afisare();
                cout << endl;
            }
            break;
        }

        case 4: {
            float bugetobisnuiti, bugetprioritari, bugetparinti;
            cout << "Introduceti bugetul de care dispun pacientii obisnuiti si apoi cei prioritari si parintii copiilor: ";
            cin >> bugetobisnuiti >> bugetprioritari >> bugetparinti;


            for (int i = 0; i < indexPacient; i++) {
                pacienti[i]->afisare(); // Afișăm informațiile pacientului

                // Pentru pacientii de tip 'prioritari', verificăm costul
                if (dynamic_cast<Pprioritar*>(pacienti[i])!=NULL) {
                    if (bugetprioritari >= pacienti[i]->getCost()) {
                        cout << "Pacientul prioritar are bani de operatie" << endl;
```

```
            }
            else {
                cout << "Pacientul prioritar nu are bani de operatie" << endl;
            }
        }

        // Pentru pacientii de tip 'obisnuit'
        else if (dynamic_cast<Pobisnuit*>(pacienti[i])!=NULL) {
            if (bugetobisnuiti >= pacienti[i]->getCost()) {
                cout << "Pacientul obisnuit are bani de operatie" << endl;
            }
            else {
                cout << "Pacientul obisnuit nu are bani de operatie" << endl;
            }
        }

        else if (dynamic_cast<Copil*>(pacienti[i]) !=NULL) {
            if (bugetparinti >= pacienti[i]->getCost()) {
                cout << "Pacientul este copil si parintii sai au bani de operatie" << endl;
            }
            else {
                cout << "Pacientul este copil, dar parintii sai nu au bani de operatie" << endl;
            }
        }
        cout << "----------" << endl;
    }
    break;
}
```

```
case 5: {
    int k;
    cout << "Introduceti pozitia (k) pentru schimbarea acestui obiect: ";
    cin >> k;

    // Verificăm dacă poziția este validă
    if (k >= 0 && k < indexPacient) {
        pacienti[k]->citire(); // Citim datele pacientului la pozitia k
    } else {
        cout << "Pozitie invalida!\n";
    }
    break;
}

case 6: {
    string p;
    cout << "Introduceti numele (p) pentru care doriti sa stergeti obiectul: ";
    cin >> p;

    bool gasit = false;
    for (int i = 0; i < indexPacient; ++i) {
          if (pacienti[i]->getNumele() == p) {
              delete pacienti[i];
              // Mutăm elementele pentru a "șterge" pacientul
              for (int j = i; j < indexPacient - 1; ++j) {
                  pacienti[j] = pacienti[j + 1];
              }
              pacienti[indexPacient-1] = nullptr;
              indexPacient--;
```

```
                gasit = true;
                break;  // Ieșim din buclă
            }
        }

        if (!gasit) {
            cout << "Nu s-a gasit obiectul cu valoarea specificata.\n";
        }
        break;
    }

    case 7:
        cout << "Stop.\n";

        for (int i = 0; i < indexPacient; ++i)
        {
            delete pacienti[i];
        }

        return;

    default:
        cout << "Optiune invalida.\n";
        break;
    }
  }
}

int main() {
```

```
    bool b;


    do
    {
        meniu();
        cout << "Reia?";
        cin >> b;
    } while (b);


    return 0;
}```

Meniu:

1. citirea de la tastatură a valorilor membrilor obiectelor diferitor clase din ierarhie în tabloul definit;

2. afișarea obiectelor tabloului la ecran;

3. sortarea obiectelor în ordine definită de student după un câmp comun tuturor claselor din ierarhie;

4. afișarea la ecran a obiectelor pentru care se cunoaște că un câmp comun tuturor claselor din ierarhie satisface o conditie formulată de student;

5. se modifică datele obiectului pe poziția k în tabloul de obiecte (k se citește de la tastatură);

6. se șterge obiectul pentru care se cunoaște că un câmp comun tuturor claselor din ierarhii satisface o condiție formulată de student;

7. Stop;

Alegeti optiunea: 1

Introduceti numarul de pacienti (maxim 100): 2

Este pacientul 1)Obisnuit; sau 2)Prioritar; sau 3)Copil;

2

Introduceti numele pacientului:

Ion

Introduceti numele doctorului:
```

Dr. Popescu

Introduceti costul:

500

Introduceti data iternarii:

01-05-2025

Introduceti numele pacientului:

Maria

Introduceti numele doctorului:

Dr. Ionescu

Introduceti costul:

600

Introduceti data iternarii:

02-05-2025

Meniu:

1. citirea de la tastatură a valorilor membrilor obiectelor diferitor clase din ierarhie în tabloul definit;

2. afișarea obiectelor tabloului la ecran;

3. sortarea obiectelor în ordine definită de student după un câmp comun tuturor claselor din ierarhie;

4. afișarea la ecran a obiectelor pentru care se cunoaște că un câmp comun tuturor claselor din ierarhie satisface o conditie formulată de student;

5. se modifică datele obiectului pe poziția k în tabloul de obiecte (k se citește de la tastatură);

6. se șterge obiectul pentru care se cunoaște că un câmp comun tuturor claselor din ierarhii satisface o condiție formulată de student;

7. Stop;

Alegeti optiunea: 2

Afisare pacienți:

numele pcientului: Ion; numele dr.: Dr. Popescu;

costul: 500; data iternarii: 01-05-2025;

---

numele pcientului: Maria; numele dr.: Dr. Ionescu;

costul: 600; data iternarii: 02-05-2025;

Meniu:

1. citirea de la tastatură a valorilor membrilor obiectelor diferitor clase din ierarhie în tabloul definit;

2. afișarea obiectelor tabloului la ecran;

3. sortarea obiectelor în ordine definită de student după un câmp comun tuturor claselor din ierarhie;

4. afișarea la ecran a obiectelor pentru care se cunoaște că un câmp comun tuturor claselor din ierarhie satisface o conditie formulată de student;

5. se modifică datele obiectului pe poziția k în tabloul de obiecte (k se citește de la tastatură);

6. se șterge obiectul pentru care se cunoaște că un câmp comun tuturor claselor din ierarhie satisface o conditie formulată de student;

7. Stop;

Alegeti optiunea: 3

Pacienți sortați:

numele pcientului: Ion; numele dr.: Dr. Popescu;

costul: 500; data iternarii: 01-05-2025;

---

numele pcientului: Maria; numele dr.: Dr. Ionescu;

costul: 600; data iternarii: 02-05-2025;

Meniu:

1. citirea de la tastatură a valorilor membrilor obiectelor diferitor clase din ierarhie în tabloul definit;

2. afișarea obiectelor tabloului la ecran;

3. sortarea obiectelor în ordine definită de student după un câmp comun tuturor claselor din ierarhie;

4. afișarea la ecran a obiectelor pentru care se cunoaște că un câmp comun tuturor claselor din ierarhie satisface o conditie formulată de student;

5. se modifică datele obiectului pe poziția k în tabloul de obiecte (k se citește de la tastatură);

6. se șterge obiectul pentru care se cunoaște că un câmp comun tuturor claselor din ierarhie satisface o conditie formulată de student;

7. Stop;

Alegeti optiunea: 4

Introduceti bugetul de care dispun pacientii obisnuiti si apoi cei prioritari si parintii copiilor:

700 500 600

numele pcientului: Ion; numele dr.: Dr. Popescu;

costul: 500; data iternarii: 01-05-2025;

Pacientul prioritar are bani de operatie

---

numele pcientului: Maria; numele dr.: Dr. Ionescu;

costul: 600; data iternarii: 02-05-2025;

Pacientul prioritar are bani de operatie

Meniu:

1. citirea de la tastatură a valorilor membrilor obiectelor diferitor clase din ierarhie în tabloul definit;

2. afișarea obiectelor tabloului la ecran;

3. sortarea obiectelor în ordine definită de student după un câmp comun tuturor claselor din ierarhie;

4. afișarea la ecran a obiectelor pentru care se cunoaște că un câmp comun tuturor claselor din ierarhie satisface o conditie formulată de student;

5. se modifică datele obiectului pe poziția k în tabloul de obiecte (k se citește de la tastatură);

6. se șterge obiectul pentru care se cunoaște că un câmp comun tuturor claselor din ierarhie satisface o conditie formulată de student;

7. Stop;

Alegeti optiunea: 5

Introduceti pozitia (k) pentru schimbarea acestui obiect: 1

Introduceti numele pacientului:

Alex

Introduceti numele doctorului:

Dr. Georgescu

Introduceti costul:

700

Introduceti data iternarii:

05-06-2025

Meniu:

1. citirea de la tastatură a valorilor membrilor obiectelor diferitor clase din ierarhie în tabloul definit;

2. afișarea obiectelor tabloului la ecran;

3. sortarea obiectelor în ordine definită de student după un câmp comun tuturor claselor din ierarhie;

4. afișarea la ecran a obiectelor pentru care se cunoaște că un câmp comun tuturor claselor din ierarhie satisface o conditie formulată de student;

5. se modifică datele obiectului pe poziția k în tabloul de obiecte (k se citește de la tastatură);

6. se șterge obiectul pentru care se cunoaște că un câmp comun tuturor claselor din ierarhie satisface o conditie formulată de student;

7. Stop;

Alegeti optiunea: 6

Introduceti numele (p) pentru care doriti sa stergeti obiectul: Ion

Meniu:

1. citirea de la tastatură a valorilor membrilor obiectelor diferitor clase din ierarhie în tabloul definit;

2. afișarea obiectelor tabloului la ecran;

3. sortarea obiectelor în ordine definită de student după un câmp comun tuturor claselor din ierarhie;

4. afișarea la ecran a obiectelor pentru care se cunoaște că un câmp comun tuturor claselor din ierarhie satisface o conditie formulată de student;

5. se modifică datele obiectului pe poziția k în tabloul de obiecte (k se citește de la tastatură);

6. se șterge obiectul pentru care se cunoaște că un câmp comun tuturor claselor din ierarhie satisface o conditie formulată de student;

7. Stop;

Alegeti optiunea: 7

Stop.

Lucrare de laborator nr. 3

6. Proiectați și implementați clasa Complex care să permită lucrul cu numere complexe. Constructorul cu parametri ai clasei va avea ca argumente partea reală, respectiv imaginară a numărului complex iar pentru constructorul implicit aceste valori iau mărimea 0. Se va asigura și un constructor de copiere. Se va redefini operatorul >> ca funcție prieten (friend) pentru citirea unui număr complex de la intrarea standard și se va redefini operatorul << ca funcție prieten (friend) pentru afișarea unui număr complex la ieșirea standard. Se vor prevedea funcții membri pentru: − accesul la partea reală, respectiv imaginară a numărului complex; − supraîncărcarea operatorului ob1-= ob1 scăderea numărului complex cu un alt număr complex dat ca argument; − supraîncărcarea operatorului ob1++ postfix și ++ob1 prefix; − supraîncărcarea operatorului ob1>= ob2 care permite compararea a două numere complexe; − supraîncărcarea operatorilor x/ob1, ob1/x pentru a permite operația de împărțire între x (număr întreg) și ob1(obiect a clasei respective); − supraîncărcarea operatorului de atribuire ob1= ob2. Funcția main() va conține exemple de aplicare corectă a operatorilor descriși relativ la clasa dată. Se vor descrie aceiași operatori (care au fost descriși ca funcții membre) și ca funcții prietene într-un al program similar.

```#include <iostream>


using namespace std;


class Complex {

private:
```

```
    int a, b;  // părțile reale și imaginare ale numărului complex
public:
    Complex() {
        a = 0;
        b = 0;
    }

    Complex(const Complex& q) {
        a = q.a;
        b = q.b;
    }

    // Constructor care acceptă și valori de tip double pentru părțile reale și imaginare
    Complex(double real, double imag) {
        a = static_cast<int>(real);  // Convertim valorile de tip double în int
        b = static_cast<int>(imag);
    }

    friend istream& operator>>(istream& in, Complex& z) {
        in >> z.a >> z.b;
        return in;
    }

    friend ostream& operator<<(ostream& out, const Complex& z) {
        out << z.a << " + " << z.b << "i";
        return out;
    }

    Complex& operator-=(const Complex& other) {
```

```
    a -= other.a;
    b -= other.b;
    return *this;
}

Complex operator++(int) { // postfix ++a ++b
    Complex temp = *this;
    a++;
    b++;
    return temp;
}

Complex& operator++() { // prefix ++a ++b
    a++;
    b++;
    return *this;
}

bool operator>=(const Complex& comp) const {
    return (this->a >= comp.a && this->b >= comp.b);
}

Complex operator/(int x) const {
    return Complex(static_cast<double>(a) / x, static_cast<double>(b) / x);
}

Complex operator/(double x) const {
    return Complex(static_cast<double>(a) / x, static_cast<double>(b) / x);
}
```

```
    Complex& operator=(const Complex& var) {
        if (this != &var) {
            a = var.a;
            b = var.b;
        }
        return *this;
    }

    int getA() const { return a; }
    int getB() const { return b; }

    void afisare() const {
        cout << *this << endl;
    }

    ~Complex() {
        clog << "Stergere" << endl;
    }

};

int main() {
    Complex z1,z2,z3,z4;
    cout << "Introdu a, b: ";
    cin >> z1;
    cout << "Afisare: " << z1 << endl;

    cout << "Introdu a, b: ";
```

```
cin >> z2;
cout << "Afisare: " << z2 << endl;

z1-=z2;
cout << "rezultatul scaderii: " << z1 << endl;

z3 = z1++;
cout << "valoarea lui z1 inainte de incrementare: " << z3 << endl;
cout << "rezultatul incrimentarii: " << z1 << endl;

++z1;
cout << "rezultatul incrimentarii prefixe: " << z1 << endl;

if (z1>=z2) {
   cout << "z1 mai mare sau egal" << endl;
}
else {
   cout << "z2 mai mare" << endl;
}

z4 = z1 / 2;
cout << "rezultatul impartirii: " << z4 << endl;

// Exemplu de împărțire a unui număr complex la un număr real
Complex z5 = z1 / 2.0;
cout << "Impartire cu numar real: " << z5 << endl;

Complex z6 = z1;
cout << "atribuire lui z6: " << z6 << endl;
```

```
    return 0;
}```

Introdu a, b:

3 4

Afisare: 3 + 4i

Introdu a, b:

1 2

Afisare: 1 + 2i

rezultatul scaderii: 2 + 2i

valoarea lui z1 inainte de incrementare: 2 + 2i

rezultatul incrimentarii: 3 + 3i

rezultatul incrimentarii prefixe: 4 + 4i

z1 mai mare sau egal

rezultatul impartirii: 2 + 2i

Impartire cu numar real: 2 + 2i

atribuire lui z6: 4 + 4i

---

Lucrare de laborator nr. 4

Pentru realizarea acestui laborator ca bază se va lua laboratorul nr. 3 la care se vor adăuga următoarele: 1. baza clasei descrise în laboratorul 3 se va crea o clasă-șablon asupra căreia pot fi aplicați aceeași operatori; 2. înafara clasei se va descrie o funcție-șablon care permite lucrul cu aceste obiecte și/sau cu parametri de alte tipuri iar în main() vor fi exemple de apelare a acestei funcții; 3. în cadrul programului (sau în descrierea operatorilor sau membrilor clasei, sau în funcția-șablon) va fi integrat try-catch care va permite prelucrarea cel puțin a unei erori. Pentru eliminarea unor neînțelegeri adresează-te la profesor.

```#include <iostream>

#include <stdexcept> // Pentru gestionarea excepțiilor

using namespace std;

// Clasa șablon Complex
```
```

```
template <typename T>
class Complex {
private:
   T a, b;  // părțile reale și imaginare ale numărului complex
public:
   Complex() : a(0), b(0) {}

   Complex(const Complex& q) : a(q.a), b(q.b) {}

   Complex(T real, T imag) : a(real), b(imag) {}

   friend istream& operator>>(istream& in, Complex& z) {
      in >> z.a >> z.b;
      return in;
   }

   friend ostream& operator<<(ostream& out, const Complex& z) {
      out << z.a << " + " << z.b << "i";
      return out;
   }

   Complex& operator-=(const Complex& other) {
      a -= other.a;
      b -= other.b;
      return *this;
   }

   Complex operator++(int) { // postfix ++a ++b
      Complex temp = *this;
```

```
    ++*this;
    return temp;
}

Complex& operator++() { // prefix ++a ++b
    a++;
    b++;
    return *this;
}

bool operator>=(const Complex& comp) const {
    return (this->a >= comp.a && this->b >= comp.b);
}

Complex operator/(T x) const {
    if (x == 0) {  // Prevenim diviziunea cu zero
        throw invalid_argument("Divizarea la zero nu este permisa!");
    }
    return Complex(a / x, b / x);
}

Complex& operator=(const Complex& var) {
    if (this != &var) {
        a = var.a;
        b = var.b;
    }
    return *this;
}
```

```
    T getA() const { return a; }
    T getB() const { return b; }

    void afisare() const {
        cout << *this << endl;
    }

    ~Complex() {
        clog << "Stergere" << endl;
    }
};

// Funcție șablon pentru a lucra cu obiecte de tip Complex
template <typename T>
void operatiiComplex(Complex<T>& z1, Complex<T>& z2) {
    try {
        cout << "Introduceti a, b pentru primul numar complex: ";
        cin >> z1;

        cout << "Introduceti a, b pentru al doilea numar complex: ";
        cin >> z2;

        // Exemplu de scădere
        z1 -= z2;
        cout << "rezultatul scaderii: " << z1 << endl;

        // Exemplu de incrementare postfix
        Complex<T> z3 = z1++;
        cout << "valoarea lui z1 inainte de incrementare: " << z3 << endl;
```

```
        cout << "rezultatul incrementarii: " << z1 << endl;

        // Exemplu de incrementare prefix
        ++z1;
        cout << "rezultatul incrementarii prefixe: " << z1 << endl;

        // Comparare
        if (z1 >= z2) {
            cout << "z1 este mai mare sau egal cu z2." << endl;
        } else {
            cout << "z2 este mai mare." << endl;
        }

        // Exemplu de împărțire
        Complex<T> z4 = z1 / 2;
        cout << "rezultatul impartirii: " << z4 << endl;

        // Exemplu de atribuire
        Complex<T> z5 = z1;
        cout << "atribuirea lui z5: " << z5 << endl;
    }
    catch (const exception& e) {
        cout << "Eroare: " << e.what() << endl;
    }
}

int main() {
    Complex<int> z1, z2, z3;
    Complex<double> z4, z5;
```

```
    cout << "Operatii pe numere complexe de tip int:" << endl;
    operatiiComplex(z1, z2);

    cout << "\nOperatii pe numere complexe de tip double:" << endl;
    operatiiComplex(z4, z5);

    return 0;
}
```

```
Operatii pe numere complexe de tip double:
Introduceti a, b pentru primul numar complex: 5.5 6.5
Introduceti a, b pentru al doilea numar complex: 2.5 3.5
rezultatul scaderii: 3 + 3i
valoarea lui z1 inainte de incrementare: 3 + 3i
rezultatul incrementarii: 4 + 4i
rezultatul incrementarii prefixe: 5 + 5i
z1 este mai mare sau egal cu z2.
rezultatul impartirii: 2 + 2i
atribuirea lui z5: 5 + 5i
```