PROGRAMARE ORIENTATĂ PE OBIECTE

LABORATOR 1 – PARTEA A II-A

Anul universitar 2023 – 2024

Semestrul II

1 Noțiuni teoretice

1.1 Cuvântul cheie `this`

În limbajul de programare C++, `this` este un pointer special către obiectul curent în cadrul căruia este apelată o metodă. Acest pointer este disponibil în fiecare metodă a unei clase și este folosit pentru a face referire la obiectul pe care este apelată metoda respectivă.

Câteva utilizări comune ale cuvântului cheie `this` sunt:

1. **Accesarea membrilor obiectului**: this poate fi folosit pentru a accesa și modifica membrii obiectului curent. De exemplu, `this->member` este echivalent cu `member` în cadrul unei metode.
2. **Returnarea obiectului curent**: Metodele pot returna obiectul curent utilizând `return *this;`. Aceasta este utilă în implementarea unor metode de tip chaining (în lanț) sau pentru a permite apeluri concatenate de metode.
3. **Evitarea ambiguității**: Atunci când numele unui parametru sau o variabilă locală dintr-o metodă are același nume cu un membru al clasei, utilizarea `this` poate ajuta la evitarea ambiguității în cadrul codului.
4. **Utilizare în constructori și destructori**: În special în constructori și destructori, `this` poate fi util pentru a accesa obiectul curent și pentru a inițializa sau a face referire la membrii săi.
5. **Compararea cu `nullptr`**: În metodele care primesc un pointer către obiect ca argument, `this` poate fi folosit pentru a verifica dacă obiectul curent este diferit de `nullptr`.

Deși utilizarea explicită a `this` este adesea opțională în C++, compilerul implicit folosește `this` în fundal pentru a accesa membrii obiectului în cadrul metodelor. Utilizarea directă a this este adesea folosită pentru a face codul mai explicit și mai ușor de înțeles.

Cuvântul cheie `this` reprezintă un pointer către obiectul al cărei funcție membru este executată. Se folosește în interiorul unei funcții membru a unei clase pentru a se face referirea chiar la obiectul respectiv.

Una dintre utilități constă în verificarea transmiterii chiar a acelui obiect ca parametru pentru o funcție membru. De exemplu:

Exemplul 1:

```
#include <iostream>

using namespace std;

class MyClass {
private:
    int value;

public:

    // Accesarea membrului value folosind pointerul this
    MyClass(int value) {
        this->value = value;
    }

    // Modificarea membrului value folosind pointerul this
    void setValue(int newValue) {
        this->value = newValue;
    }

    // Returnarea membrului value folosind pointerul this
    int getValue() const {
        return this->value;
    }
};

int main() {
    MyClass obj(10);
    cout << "Valoarea obiectului: " << obj.getValue() << endl; // Afiseaza 10
    obj.setValue(20);
    cout << "Valoarea obiectului dupa modificare: " << obj.getValue() << endl; // Afiseaza
20
    return 0;
}
```

Exemplul 2:

```
#include <iostream>

using namespace std;

class MyClass {
private:
    int value;
```

```
public:
    MyClass(int value) : value(value) {}

    MyClass& increment() {
        this->value++; // Incrementarea membrului value folosind pointerul this
        return *this; // Returnarea obiectului curent
    }

    int getValue() const {
        return this->value;
    }
};

int main() {
    MyClass obj(5);
    cout << "Valoarea obiectului initial: " << obj.getValue() << endl; // Afiseaza 5
    obj.increment().increment().increment(); // Apeluri concatenate ale metodei increment
    cout << "Valoarea obiectului dupa incrementare multipla: " << obj.getValue() << endl;
// Afiseaza 8
    return 0;
}
```

În ambele exemple, `this` este folosit pentru a face referire la obiectul curent și pentru a accesa sau modifica membrii săi. În primul exemplu, `this` este utilizat pentru a accesa și modifica membrul `value`, în timp ce în al doilea exemplu, `this` este utilizat pentru a returna obiectul curent din metoda `increment()`.

1.2 Membri statici

O clasă poate conține membri statici, atât date cât și funcții.

În limbajul de programare C++, cuvântul cheie static poate avea mai multe semnificații în funcție de contextul în care este utilizat.

1.2.1 Variabile statice de membru

Atunci când este folosit în cadrul unei clase, static indică faptul că o variabilă este partajată între toate instanțele de obiecte ale acelei clase. Cu alte cuvinte, există o singură copie a acelei variabile care este comună tuturor obiectelor clasei. Variabilele statice de membru sunt inițializate o singură dată, în general în afara claselor, și pot fi accesate fără a crea o instanță a clasei.

O dată membru statică se mai numește și "variabilă de clasă", deoarece ea reprezintă o variabilă comună pentru toate obiectele din aceeași clasă, distribuind aceeași valoare: adică valoarea ei nu diferă de la un obiect la altul al acelei clase.

```
#include <iostream>

using namespace std;

class MyClass {
public:
    static int count; // Variabila statica de membru

    MyClass() {
        count++; // Incrementam variabila statica in constructor
    }
};

int MyClass::count = 0; // Initializare a variabilei statice de membru

int main() {
    MyClass obj1;
    MyClass obj2;
    MyClass obj3;

    cout << "Numarul de obiecte create: " << MyClass::count << endl; // Afiseaza 3
    return 0;
}
```

De fapt, membrii statici au aceleași proprietăți ca variabilele care nu sunt membri dar intră în domeniul clasei. Din acest motiv, precum și pentru a evita declararea lor de mai multe ori, nu pot fi inițializați direct în interiorul clasei, dar trebuie inițializați undeva în exteriorul ei.

1.2.2 Funcții statice de membru

Clasele pot avea și funcții ca membri statici. Ele reprezintă același lucru: membrii unei clase care sunt comuni tuturor obiectelor din acea clasă, acționând exact la fel ca funcțiile care nu sunt membri dar se accesează ca membri ai clasei. Deoarece sunt ca funcțiile ne-membri, ele nu pot accesa membri nestatici ai clasei (nici variabile, nici funcții). De asemenea, nu pot folosi cuvântul cheie `this`.

Atunci când este folosit în cadrul unei clase, static indică faptul că o metodă este asociată cu clasa, nu cu instanțele de obiecte ale acelei clase. Aceasta înseamnă că metoda poate fi apelată fără a crea o instanță a clasei.

```
#include <iostream>

using namespace std;

class Math {
public:
    static int add(int a, int b) {
```

```
        return a + b;
    }
};

int main() {
    int sum = Math::add(5, 3); // Apelarea metodei statice add() fără a crea un obiect Math
    cout << "Suma: " << sum << endl; // Afisează 8
    return 0;
}
```

1.3 Funcții membru const

În limbajul de programare C++, `const` este un cuvânt cheie care poate fi folosit pentru a declara constante, a specifica că un obiect sau o variabilă nu poate fi modificată și pentru a indica că o metodă nu modifică starea obiectului pe care este apelată.

Când un obiect al unei clase este calificat ca obiect const:

```
const Clasa_mea obiectul_meu;
```

accesul la datele membru ale sale din afara clasei se poate face doar pentru citire, ca și cum toate datele membru ar fi const pentru tot ceea ce le apelează din afara clasei.

Funcțiile membru precizate ca fiind const nu pot modifica date membru nestatice și nici nu pot apela alte funcții membru care nu sunt const. În esență, membrilor const nu li se permite să modifice starea unui obiect.

Obiectele const sunt restricționate la accesarea numai a membrilor marcați cu const, dar obiectele non-const nu sunt restricționate și pot accesa atât membri const cât și non-const.

V-ați putea gândi că oricum rar este nevoie să declarăm obiecte const, deci nu merită efortul de a declara const toți membrii care să nu modifice obiectul, însă obiectele const sunt chiar foarte întâlnite.

2 Aplicații

2.1 Adunare fracții

Implementați o clasă, *AdunareFractie*, care are doi membri, *numărător* și *numitor*, și o metodă statică, *adunare*, și returnează valoarea nesimplificată a adunării a două fracții.

3 Rezolvări

3.1 Adunare fracții

```
#include <iostream>

using namespace std;

class Fractie {
private:
    int numarator;
    int numitor;

public:
    Fractie(int numarator, int numitor) : numarator(numarator), numitor(numitor) {}

    // Metoda statica pentru adunarea a doua fractii si afisarea sumei
    static void aduna(const Fractie& f1, const Fractie& f2) {
        int numaratorSuma = f1.numarator * f2.numitor + f2.numarator * f1.numitor;
        int numitorSuma = f1.numitor * f2.numitor;

        cout << "Suma fractiilor: " << numaratorSuma << "/" << numitorSuma << endl;
    }

    void afiseaza() const {
        cout << numarator << "/" << numitor << endl;
    }
};

int main() {
    Fractie fractie1(1, 2);
    Fractie fractie2(1, 3);

    Fractie::aduna(fractie1, fractie2);

    return 0;
}
```

4 Teme

Pentru laboratorul L1, implementați două probleme:

- o problemă dintre *4.1. Puncte* sau *4.2. Complex*;
- problema *4.3. User*.

Denumiți proiectele realizate de voi cu denumirea problemei și numele vostru. Exemplu: ***puncte_Bold Nicolae***.

4.1 Puncte

Implementați un proiect C++ care să conțină clasa *Punct*, cu următoarele caracteristici:

- atribute:
  a. *x*, de tip ***int***, care va reține abscisa unui punct într-un sistem de coordonate xOy;
  b. *y*, de tip ***int***, care va reține ordonata unui punct într-un sistem de coordonate xOy.
- clasa va conține următorii constructori și destructori:
  a. un constructor implicit, un constructor cu parametri și un constructor de copiere;
  b. un destructor.
- clasa va conține metodele:
  a. metoda *afiseaza*, care va afișa coordonatele punctului;
  b. metoda *getX*, getter pentru abscisă;
  c. metoda *getY*, getter pentru ordonată;
  d. metoda *getXY*, getter pentru coordonatele unui punct;
  e. metoda *setXY*, setter pentru coordonatele unui punct;
  f. metoda *dist*, implementată static și nestatic, care calculează distanța între două puncte A și B folosind următoarea formulă:

$$d(A,B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$$

  g. metoda *peDreapta*, care verifică dacă un punct se află pe o dreaptă a cărei ecuație e dată ca parametru (*a* și *b* dintr-o ecuație de tipul $ax + b = 0$).

Cerință: să se construiască două obiecte de tip *Punct* și să se aplice operații conform metodelor *dist* și *peDreapta* asupra lor.

4.2 Complex

Un număr complex este un concept matematic care constă dintr-o parte reală și o parte imaginară. Reprezentarea generală a unui număr complex este de forma:

$$z = a + bi$$

unde *a* este partea reală, *b* este partea imaginară, iar *i* este unitatea imaginară, definită ca $i^2 = -1$.

În cadrul unui proiect, implementați clasa *Complex*, cu următoarele caracteristici:

- atribute:
  a. *a*, de tip ***int***, care va reține partea întreagă a numărului complex;
  b. *b*, de tip ***int***, care va reține partea fracționară a numărului complex.
- clasa va conține următorii constructori și destructori:
  a. un constructor implicit, un constructor cu parametri și un constructor de copiere;
  b. un destructor.
- metode:
  a. metoda *afiseaza*, care va afișa numărul complex sub forma $a + bi$;
  b. *getA*, care returnează care returnează partea întreagă a numărului;
  c. *getB*, care returnează care returnează partea fracționară a numărului;
  d. *setAB*, care setează părțile numărului;
  e. *getAB,* care returnează părțile numărului;
  f. *adunare*, care adună două numere complexe, după formula $z_1 + z_2 = (a_1 + a_2) + (b_1 + b_2) \times i$;
  g. *modul*, implementată static și nestatic, de tip *double*, care returnează modulul unui număr complex *r*, după formula $r = \sqrt{a^2 + b^2}$.

Cerință: să se construiască două obiecte de tip *Complex* și să se calculeze operații de adunare și aflare a modulului asupra lor.

4.3 User

Implementați un proiect C++ care să conțină clasa *User*, cu următoarele caracteristici:

- clasa va avea următoarele atribute:
  a. *nume*, de tip ***char***, care va fi alocat dinamic;
  b. *parola*, de tip ***char***, care va fi alocat dinamic;
  c. *an_nastere*, de tip ***int***.
- clasa va conține următorii constructori și destructori:
  a. doi constructori: un constructor implicit și un constructor cu parametri;
  b. un destructor, cu rol de a elibera memoria alocată.
- clasa va conține cinci metode:
  a. metoda *afiseaza*, care va afișa datele utilizatorului;
  b. metoda *verifica_parola*, care verifică dacă parola îndeplinește două condiții: numărul minim de caractere este 8 și parola conține cel puțin o cifră;

c. metoda *getAn*, care returnează anul de naștere al utilizatorului;
d. metoda *setAn*, care actualizează anul de naștere al unui utilizator.
e. metoda *varsta*, care calculează vârsta unui utilizator și care va fi definită în afara clasei, fiind definit doar prototipul în cadrul clasei.

Cerință: să se construiască un obiect de tip *User* și să se aplice operații conform metodelor asupra lui.