1. Asemănări și deosebiri între constructorul de copiere și operatorul de atribuire

Asemănări:

- Ambele sunt folosite pentru a copia obiectele unei clase.
- Ambele se ocupă de copierea atributelor unui obiect în altul.

Diferențe:

- **Constructorul de copiere** este apelat **automat** la crearea unui nou obiect pe baza altuia deja existent.
- **Operatorul de atribuire** este apelat **explicit** atunci când unui obiect existent i se atribuie alt obiect de același tip, **după ce acesta a fost deja creat**.

```
class Exemplu {
    int x;
public:
    // Constructor normal
    Exemplu(int val) : x(val) {}

    // Constructor de copiere
    Exemplu(const Exemplu& other) {
        x = other.x;
        cout << "Constructor de copiere apelat\n";
    }

    // Operator de atribuire
    Exemplu& operator=(const Exemplu& other) {
        if (this != &other) {  // Evităm auto-atribuirea
            x = other.x;
            cout << "Operator de atribuire apelat\n";
        }
        return *this;
    }
```

2. Operator unar în C++ dar nu și în C

:: (Operatorul de rezolutie)

2. Supradefiniti „operatorul de atribuire" pentru clasa Complex, Declarați clasa cu o structura minimala și definiti doar operatorul in cauza;

```
class Complex {
    double real, imag;
public:
    Complex(double r = 0, double i = 0) : real(r), imag(i) {}

    // Operator de atribuire
    Complex& operator=(const Complex& other) {
        if (this != &other) {
            real = other.real;
            imag = other.imag;
        }
        return *this;
    }
```

4. Dați un exemplu unde specificatorul virtual este necesar, aratand și motivarea.

Specificatorul `virtual` este necesar pentru **moștenire**, pentru a evita **problema metodei ascunse** (polimorfism dinamic).

```
class Baza {
public:
    virtual void afisare() { cout << "Afisare din Baza\n"; }
};

class Derivata : public Baza {
public:
    void afisare() override { cout << "Afisare din Derivata\n"; }
};

int main() {
    Baza* ptr = new Derivata();
    ptr->afisare();  // Apel corect la metoda din Derivata
    delete ptr;
    return 0;
}
```

Fără `virtual`, se apelează metoda din clasa de bază.

5. Ce reprezintă tipul referință?În ce situații putem întâlni noțiunea de referință?Exemplu unde se folosește referința

În C++, **referința** (&) este un alias pentru o variabilă existentă. Practic, o referință este un alt nume pentru aceeași zonă de memorie.

5.1. În ce situații putem întâlni noțiunea de referință?

1. **Transmisia parametrilor prin referință** – Permite modificarea valorilor fără a crea copii.
2. **Returnarea prin referință** – Permite întoarcerea unei variabile dintr-o funcție fără copiere.
3. **Referințe constante (`const`)** – Se folosește pentru a evita copierea datelor mari fără a le modifica.
4. **Referințe la obiecte din clase** – Se utilizează în OOP pentru eficiență.

5.2. Exemplu unde se folosește referința

5.2.1. Transmiterea parametrilor prin referință

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

void dubleaza(int& x) {  // x este referință, modifică direct variabila originală
    x *= 2;
}

int main() {
    int numar = 5;
    dubleaza(numar);
    cout << numar;  // Afișează 10
    return 0;
}
```

5.2.2. Returnarea prin referință

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

int& getValoare(int& x) {
    return x;  // Returnează referința către x
}

int main() {
    int a = 10;
    getValoare(a) = 20;  // Modifică direct a
    cout << a;  // Afișează 20
    return 0;
}
```

5.2.3. Referințe constante (`const`)

```cpp
void afiseaza(const int& val) {  // Nu permite modificarea lui val
    cout << val << endl;
}
```

6. Situație unde listă de inițializare este obligatorie

Când avem const, referințe sau obiecte ale unor clase fără constructor implicit.

1. **Membri const**

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

class Exemplu {
    const int x; // Variabilă constantă
public:
    Exemplu(int val) : x(val) {} // Obligatoriu, altfel eroare
    void afisare() { cout << x << endl; }
};

int main() {
    Exemplu obj(10);
    obj.afisare();  // Afișează 10
    return 0;
}
```

2: Referințe

```cpp
class Referinta {
    int& ref; // Referință la un întreg
public:
    Referinta(int& r) : ref(r) {} // Obligatoriu, altfel eroare
};
```

3: Clasă fără constructor implicit

```cpp
class FaraConstructorImplicit {
public:
    FaraConstructorImplicit(int x) {}
};

class Test {
    FaraConstructorImplicit obj; // Nu există constructor implicit!
public:
    Test(int val) : obj(val) {} // Obligatoriu, altfel eroare
};
```

Tipuri de polimorfism

- **Polimorfism parametric:** este mecanismul prin care putem defini o metoda cu acelasi nume in aceeasi clasa. De exemplu Template-uri: (**`template<typename T>`**).

- **Polimorfism ad-hoc:** Supraincarcarea operatorilor și funcțiilor.
- **Polimorfism prin moștenire:** `virtual` și suprascrierea metodelor.

8. Dați un exemplu de clasa, în limbajul C++ (la alegerea dv.). Exemplul va contine declaratia clasei, definires corpului metodelor (maxim 2.) și o utilizare simpla în funcția main().

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

class Masina {
    string marca;
public:
    Masina(string m) : marca(m) {}
    void afisare() { cout << "Marca: " << marca << endl; }
};

int main() {
    Masina m("BMW");
    m.afisare();
    return 0;
}
```

9. Ce putefi spune despre următorii operatori (exista/nu exista in C/C, denumire, rol, exemplu de utilizare)**

a. *-> Acces prin pointer la un membru al clasei(in cazul unei adrese de obiect)

b. :* Nu exista.

c. ::* Nu exista.