2. Ce reprezinta pointerul this? Explicați cum poate fi utilizat (exemple).

`this` este un pointer implicit în C++ care se referă la **obiectul curent** pe care îl invocă o metodă a unei clase. Este folosit pentru a accesa membri ai obiectului din interiorul metodei respective.

```
class Exemplu {
private:
    int x;
public:
    Exemplu(int val) : x(val) {}

    Exemplu& seteazaX(int val) {
        this->x = val;
        return *this;  // Returnează obiectul curent
    }

    void afiseaza() {
        cout << "Valoarea x este: " << x << endl;
    }
};
```

2. Supradefiniti operatorul de incrementare (varianta pre si post fixata) pentru o posibila clasa Test;
Declarati clasa cu o structura minimala si definiti doar operatorii in cauza;

```
class Test {
private:
    int valoare;
public:
    Test(int val) : valoare(val) {}

    // Operator de incrementare pre-fixat
    Test& operator++() {
        valoare++;  // Incrementează valoarea
        return *this;  // Returnează obiectul curent
    }
```

3. Dati un exemplu de o clasa polimorfa. Care este utilitatea ei?

O **clasă polimorfă** în C++ este o clasă care conține cel puțin o **metodă virtuală** (de obicei, o metodă virtuală pură), care poate fi suprascrisă în clasele derivate.

Clasa polimorfă permite:

- Crearea de funcționalități comune în clasele de bază și derivate.
- Utilizarea de **pointeri sau referințe la clasa de bază** pentru a manipula obiecte de diferite tipuri derivate, aplicând polimorfismul.

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

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;
}
```

4. Ce reprezintă tipul static in C ++ (data sau metoda)? Enumerati principalele proprietăți.

`static` este un modificator care face ca variabilele sau metodele să fie **partajate** între toate instanțele unei clase sau să aibă o **durată de viață globală**, dar un **scop local**.

Proprietăți ale `static`:

1. **Variabile statice**:
    - Partajate între toate instanțele unei clase.
    - Au o singură copie în memorie.
    - Se inițializează o singură dată și sunt disponibile pe toată durata programului.
2. **Metode statice**:
    - Pot fi apelate fără a crea un obiect.
    - Nu pot accesa membri non-statici ai clasei.

3. **Durata de viață**:
    - Membrii statici există pe toată durata execuției programului, indiferent de instanțele create.
4. **Scopul**:
    - Variabilele și metodele statice sunt vizibile doar în fișierul în care sunt definite (dacă sunt folosite global).

6. Utilitatea constructorilor de conversie de tip. Cum ai recunoastem? Un scurt exemplu.

Constructorii de conversie de tip sunt folosiți pentru a converti un obiect de un tip într-un obiect de alt tip, fără a fi nevoie să apelăm un constructor explicit.

Cum recunoaștem constructorii de conversie de tip:

- Constructorii de conversie de tip nu au un **tip de returnare** specificat (sunt de obicei constructori care primesc un argument de un tip diferit de tipul clasei).
- Sunt **invocați implicit** de compilator pentru a efectua conversia între tipuri, fără a fi necesar apelul direct al unui constructor.

```
class Test {
private:
    int valoare;
public:
    // Constructorul de conversie de tip
    Test(int val) {
        valoare = val;
    }

    void afiseaza() {
        cout << "Valoare: " << valoare << endl;
    }
};

int main() {
    Test t1 = 10;  // Constructorul de conversie de tip va fi apelat aici
    t1.afiseaza();  // Afișează: Valoare: 10
```

În acest exemplu, `Test(int val)` este un constructor de conversie de tip, care permite conversia unui **int** într-un obiect de tip `Test`.

9. **Pentru urmatoarele concepte stabiliti ierarhia (structura) de clase:**

- **O aplicatie lucreaza cu trei concepte: Componenta electronica, tranzistor, capacitor iar**
- **In interiorul aplicatiei vom spune ca in magazie exista N componente iar afisarea lor se face prin metoda Print**

Componenta electronica: Clasa de bază, care conține atribute și metode comune pentru toate componentele electronice.

Tranzistor: Clasă derivată din `ComponentaElectronica`, care adaugă atribute și comportamente specifice tranzistorilor.

Capacitor: Clasă derivată din `ComponentaElectronica`, care adaugă atribute și comportamente specifice capacitorilor.

```
ComponentaElectronica
     /       \
Tranzistor  Capacitor
```