Algoritm de Verificare Compilare C++ - POO

🔍 PASUL 1: VERIFICĂRI SINTACTICE RAPIDE (10 secunde)

Căutare erori evidente:

- **Paranteze/acolade** - sunt toate închise?
- **Punct și virgulă** - lipsește după declarații de clase sau variabile?
- **Include-uri** - sunt toate necesare incluse?
- **Namespace** - `using namespace std;` este prezent?

🧬 PASUL 2: VERIFICĂRI CONSTRUCTORI/DESTRUCTORI (20 secunde)

2.1 Constructor Default

cpp

```cpp
// PROBLEMĂ FRECVENTĂ: Constructor cu parametri șterge constructorul default
class A {
    A(int x) {} // ⚠️ Nu mai avem constructor default!
};
A arr[10]; // ❌ NU COMPILEAZĂ - array-ul are nevoie de constructor default
```

Ce să cauți:

- Există array-uri de obiecte? → Verifică dacă există constructor default
- Constructor cu parametri fără valori default? → Posibilă problemă

2.2 Variabile Statice

cpp

```cpp
class A {
    static int x; // ⚠️ Declarată dar nu inițializată
};
// Lipsește: int A::x = 0;
```

Ce să cauți:

- `static` în clasă → Verifică inițializarea în afara clasei
- **Excepție:** variabilele statice NU pot fi în lista de inițializare!

🔗 PASUL 3: VERIFICĂRI MOȘTENIRE (30 secunde)

3.1 Accesibilitate Moștenire

cpp

```cpp
class D : B { // ⚠️ Moștenire PRIVATĂ implicit!
   // B devine inaccesibilă
};
B* p = new D; // ❌ NU COMPILEAZĂ
```

Regulă: Întotdeauna verifică dacă lipsește `public` în moștenire!

3.2 Acces la Membri Protected

cpp

```cpp
class B {
protected:
   int x;
};
class D : public B {
   int f(B ob) {
      return ob.x; // ❌ NU COMPILEAZĂ - x este protected
   }
};
```

Regulă: `protected` = accesibil în clasa derivată, dar NU prin alte obiecte!

🎯 PASUL 4: VERIFICĂRI POINTERI/REFERINȚE (20 secunde)

4.1 Referințe la Obiecte Temporare

cpp

```cpp
void f(problema& o) {} // Parametru prin referință
problema temp() { return problema(6); }
f(temp()); // ❌ NU COMPILEAZĂ - funcția returnează valoare, nu referință
```

4.2 Pointeri Derivată/Bază

cpp

```cpp
D* p = new B; // ❌ NU COMPILEAZĂ - pointer derivată nu poate primi bază
B* p = new D; // ✅ OK - pointer bază poate primi derivată
```

🛠️ PASUL 5: VERIFICĂRI FUNCȚII/OPERATORI (30 secunde)

5.1 Ambiguitate în Funcții

cpp

```cpp
class Base {
    int f();
    int f(string);
};
class Derived : public Base {
    int f(); // ⚠️ Ascunde TOATE funcțiile f din Base!
};
Derived d;
d.f("hello"); // ❌ NU COMPILEAZĂ - f(string) este ascunsă
```

5.2 Operatori cu Template

cpp

```cpp
template<class T> T f(T x, T y) { return x + y; }
int f(int x, int y) { return x - y; }

int* a = new int(3);
int b = 5;
f(a, b); // ⚠️ Verifică ce funcție se apelează - f(int*, int) sau template?
```

5.3 Funcții Const

cpp

```cpp
class A {
    int& f() const { return x; } // ❌ NU COMPILEAZĂ
    // Funcția const nu poate returna referință non-const
};
```

🔄 PASUL 6: VERIFICĂRI VIRTUALI/POLIMORFISM (20 secunde)

6.1 Destructori Virtuali

cpp

```cpp
class B {
    ~B() {} // ⚠️ Destructor NU virtual
};
class D : public B {
    ~D() {}
};
B* p = new D;
delete p; // ⚠️ Se apelează doar ~B(), nu ~D()
```

6.2 Funcții Virtuale Pure

cpp

```cpp
class Abstract {
    virtual void f() = 0; // Funcție virtuală pură
};
Abstract obj; // ❌ NU COMPILEAZĂ - clasă abstractă
```

⚠️ PASUL 7: VERIFICĂRI SPECIALE (15 secunde)

7.1 Scope de Variabile

cpp

```cpp
if (condition) {
    A obj;
} else {
    B obj;
}
obj.method(); // ❌ NU COMPILEAZĂ - obj nu există în acest scope
```

7.2 Conversii Implicite

cpp

```cpp
class A {
    A(int x) {} // Constructor de conversie
};
class B {
    operator A() {} // Operator de conversie
};
```

7.3 Copy Constructor vs Operator=

cpp

```cpp
class D {
    int* ptr;
    // Lipsește copy constructor sau operator=
};
D d1, d2;
d1 = d2; // ⚠️ Possible shallow copy → double delete
```

🚀 STRATEGIA DE LUCRU RAPIDĂ

Pentru fiecare exercițiu (MAX 2 minute):

1. **10s:** Scan vizual pentru erori sintactice
2. **20s:** Verifică constructori, statice, moștenire
3. **30s:** Verifică accesibilitatea membrilor
4. **20s:** Verifică pointeri, referințe, funcții
5. **15s:** Verifică scope-uri și conversii
6. **15s:** Verifică outputul dacă compilează

🎯 INDICATORI ROȘII (Red Flags):

- `class D : B` (fără public)
- `static int x;` (fără inițializare)
- `A(int x) {}` + `A arr[10];`
- `void f(T& ref)` + apel cu valoare temporară
- `protected` + acces prin alt obiect
- Funcții cu același nume în ierarhie

- `const` + modificări
- Pointeri + conversii bază/derivată

🔧 MODIFICĂRI COMUNE:

1. Adaugă `public` în moștenire
2. Adaugă `int Class::x = 0;` pentru statice
3. Adaugă constructor default sau valori default
4. Schimbă referință în valoare pentru parametri
5. Adaugă `virtual` pentru destructori
6. Folosește `Class::function()` pentru ambiguități