


Introducere

Contact: Mihai DOGARIU


mihai.dogariu@upb.ro

<https://mdogariu.aimultimedialab.ro/>

Ce este POO?

 Programarea Obiect-Orientată = paradigmă de programare în care entitățile sunt tratate ca obiecte care interacționează între ele

vs

 Programarea procedurală = paradigmă de programare în care programele sunt constituite dintr-o secvență de proceduri (funcții)

FAQ

Ce trebuie să știm

Q: Punctajul?

- Laborator: 60%
 - Test 1: 30%
 - Test 2: 30%
- Examen final: 40%
- Criterii promovare:
 - Maxim o absență laborator
 - Minim 50% din punctajul laboratorului
 - Minim 50% din punctajul cumulat (lab + examen)

Q: Care sunt principiile POO?

1. Încapsulare
2. Abstractizare
3. Ierarhizare
4. Polimorfism

Q: Ce limbaje de POO există?

- Python
- Java
- C++
- C#
- Lisp
- Perl
- Ruby
- Golang
- Smalltalk
- Swift

etc...

Q: Ce limbaje de POO există?

	Feb 2022	Feb 2021	Change	Programming Language		Ratings	Change
1		3	▲		Python	15.33%	+4.47%
2		1	▼		C	14.08%	-2.26%
3		2	▼		Java	12.13%	+0.84%
4		4			C++	8.01%	+1.13%
5		5			C#	5.37%	+0.93%
6		6			Visual Basic	5.23%	+0.90%
7		7			JavaScript	1.83%	-0.45%
8		8			PHP	1.79%	+0.04%
9		10	▲		Assembly language	1.60%	-0.06%
10		9	▼		SQL	1.55%	-0.18%

Q: De ce C++ și nu alt limbaj?

- Limbaj de programare general, compilat
- Rapid
- Aproape de limbajul mașină
- Rulează pe majoritatea microcontrollerelor
- Evoluție naturală de la C
- Utilizat intensiv în industrie: Google, Facebook, Amazon, Twitter, SpaceX, Tesla etc.

Q: Ce IDE folosim?

- Code::Blocks
- Visual Studio
- Visual Studio Code
- CLion
- Eclipse
- NetBeans
- QtCreator

C → C++

Cele mai vizibile diferențe

C → C++

funcții de citire/scriere → `#include <stdio.h>`

```
int main () {  
    șiruri de caractere → char cuvânt[20];  
                        → char fraza[20];  
    variabilă ce stochează  
    o valoare de adevăr → int flag = (1==1);  
    citire șir de caractere (cuvânt) → scanf("%s", cuvânt);  
    eliminare trailing newline ('\n') → getchar();  
    citire șir de caractere (până la '\n') → scanf("%[^\\n]", fraza);  
  
    if (flag){  
        afișare șir de caractere → printf("%s\\n", cuvânt);  
        afișare valoare reală  
        cu 2 zecimale → printf("%.2f\\n", f);  
    }  
  
    return 0;  
}
```

```
#include <iostream> ← funcții de citire/scriere  
#include <string>  
#include <iomanip>
```

```
using namespace std;
```

```
int main () {  
    string cuvânt; ← șiruri de caractere  
    string fraza; ← șiruri de caractere  
    float f = 1.23456789;  
    bool flag = (1==1); ← variabilă ce stochează  
    o valoare de adevăr  
  
    cin >> cuvânt; ← citire șir de caractere (cuvânt)  
    cin.ignore(); ← eliminare trailing newline ('\n')  
    getline(cin, fraza); ← citire șir de caractere (până la '\n')  
  
    if (flag){  
        cout << cuvânt << endl; ← afișare șir de caractere  
        cout << fraza << '\\n' ; ← afișare șir de caractere  
        cout << fixed; ← afișare valoare reală  
        cout << setprecision(2); ← afișare valoare reală  
        cout << f << endl; ← afișare valoare reală  
        cu 2 zecimale  
    }  
  
    return 0;  
}
```

C → C++

```
#include <stdio.h>
```

```
int main () {  
    char cuvant[20];  
    char fraza[20];  
    float f = 1.23456789;  
    int flag = (1==1);  
  
    scanf("%s", cuvant);  
    getchar();  
    scanf("%[^\\n]", fraza);  
  
    if (flag){  
        printf("%s\\n", cuvant);  
        printf("%s\\n", fraza);  
        printf("%.2f\\n", f);  
    }  
  
    return 0;  
}
```

```
#include <iostream>  
#include <string>  
#include <iomanip>
```

```
using namespace std;
```



```
int main () {  
    string cuvant;  
    string fraza;  
    float f = 1.23456789;  
    bool flag = (1==1);  
  
    cin >> cuvant;  
    cin.ignore();  
    getline(cin, fraza);  
  
    if (flag){  
        cout << cuvant << endl;  
        cout << fraza << '\\n' ;  
        cout << fixed;  
        cout << setprecision(2);  
        cout << f << endl;  
    }  
  
    return 0;  
}
```

Domeniul de vizibilitate

def

Domeniu de vizibilitate (scope) = porțiunea din cod în care poate fi folosit numele unui element, e.g., variabilă, clasă, funcție.

Niveluri ale domeniului de vizibilitate:

1. bloc de cod
2. parametri de funcție
3. namespace
4. clasă
5. enumerație
6. parametri template
7. punctul de declarație

Domeniul de vizibilitate

1. Domeniul de vizibilitate la **nivel de bloc de cod**: domeniul de vizibilitate al unui nume începe în punctul declarației și se termină la sfârșitul blocului de cod

```
int main() {  
    int x = 0; // incepe domeniul de vizibilitate al lui x exterior  
    {  
        int x = 1; // incepe domeniul de vizibilitate al lui x interior  
        // domeniul de vizibilitate al lui x exterior este suspendat  
    } // sfarsit bloc de cod, sfarst domeniu de vizibilitate al lui x interior  
    // se reia domeniul de vizibilitate al lui x exterior  
}
```

```
int main() {  
    for (int i=0; i < 5; i++){ // incepe domeniul de vizibilitate al lui i  
        std::cout << i << std::endl;  
    } // se termina domeniul de vizibilitate al lui i  
    i = 10; // eroare: i nu a fost declarat in acest domeniu de vizibilitate  
}
```

Domeniul de vizibilitate

2. Domeniul de vizibilitate la **nivel de parametri de funcție**: domeniul de vizibilitate al unui nume începe în punctul declarației și se termină la sfârșitul corpului funcției.

```
int n = 3; // incepe domeniul de vizibilitate al var globale n

void f(int n){ // incepe domeniul de vizibilitate al parametrului functiei n
               // domeniul de vizibilitate al var globale n este suspendat
    ++n;
} // sfarsit domeniu de vizibilitate al parametrului functiei n
// se reia domeniul de vizibilitate al var globale n
```

Domeniul de vizibilitate

3. Domeniul de vizibilitate la **nivel de namespace**: domeniul de vizibilitate al unui nume începe în punctul declarației din namespace și se termină la sfârșitul namespace-ului. Numele din namespace își extind vizibilitatea la orice alt domeniu de vizibilitate în care sunt introduse (cu directiva `using`) din momentul introducerii, până la sfârșitul domeniului.

4. Domeniul de vizibilitate la **nivel de clasă**: domeniul de vizibilitate al unui nume începe în punctul declarației din clasă și acoperă tot restul clasei și corpurile funcțiilor membre ale clasei.

Namespace

def

namespace = gruparea unor entități (obiecte, clase, funcții) sub un singur domeniu de vizibilitate

Declarare:

```
namespace firstnamespace{  
    int a = 1;  
    namespace secondnamespace{  
        int b = 2;  
    }  
}
```

Utilizare:

```
int c = 0;  
c += firstnamespace::a;  
c += firstnamespace::secondnamespace::b; // c = 3
```

:: - operatorul de rezoluție – oferă acces la toate entitățile din namespace-ul care îl precede.

Namespace

```
namespace firstnamespace{  
    int a = 1;  
    namespace secondnamespace{  
        int b = 2;  
    }  
}
```

Doar variabila a din firstnamespace poate fi accesată fără operatorul de rezoluție.

```
using firstnamespace::a;  
int c = 0;  
c += a;  
c += firstnamespace ::secondnamespace::b; // c = 3
```

Doar variabila b din secondnamespace poate fi accesată fără operatorul de rezoluție.

```
using firstnamespace::a;  
using firstnamespace::secondnamespace::b;  
int c = 0;  
c += a;  
c += b; // c = 3
```

secnamespace este un alias pentru firstnamespace::secondnamespace.

```
using firstnamespace::a;  
namespace secnamespace = firstnamespace::secondnamespace;  
int c = 0;  
c += a;  
c += secnamespace::b; // c = 3
```

Namespace

```
namespace firstnamespace{  
    int a = 1;  
    namespace secondnamespace{  
        int b = 2;  
    }  
}
```

```
using namespace firstnamespace;  
int c = 0;  
c += a;  
c += secondnamespace::b; // c = 3
```

Toate entitățile din interiorul `firstnamespace` pot fi accesate fără operatorul de rezoluție.

```
using namespace firstnamespace;  
using namespace secondNamespace;  
int c = 0;  
c += a;  
c += b; // c = 3
```

Toate entitățile din interiorul `secondnamespace` pot fi accesate fără operatorul de rezoluție.

```
using namespace secondnamespace;  
using namespace firstnamespace;  
int c = 0;  
c += a;  
c += b; // c = 3
```

Greșit! Nu se cunoaște cine este `secondnamespace`, deoarece nu a fost accesat `firstnamespace` apriori.

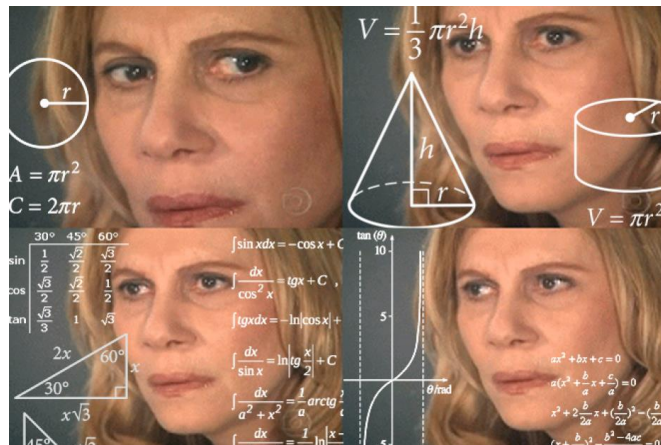
Namespace

```
namespace firstnamespace{  
    int a = 1;  
    namespace secondnamespace{  
        int b = 2;  
    }  
}
```

```
using namespace firstnamespace;  
using namespace secondnamespace;  
int c = 0;  
int a = 100;  
c += a;  
c += b; // c = ???
```

Coliziune de nume. Foarte ușor de scăpat din vedere!

c=102



Ce variantă folosim???

C → C++

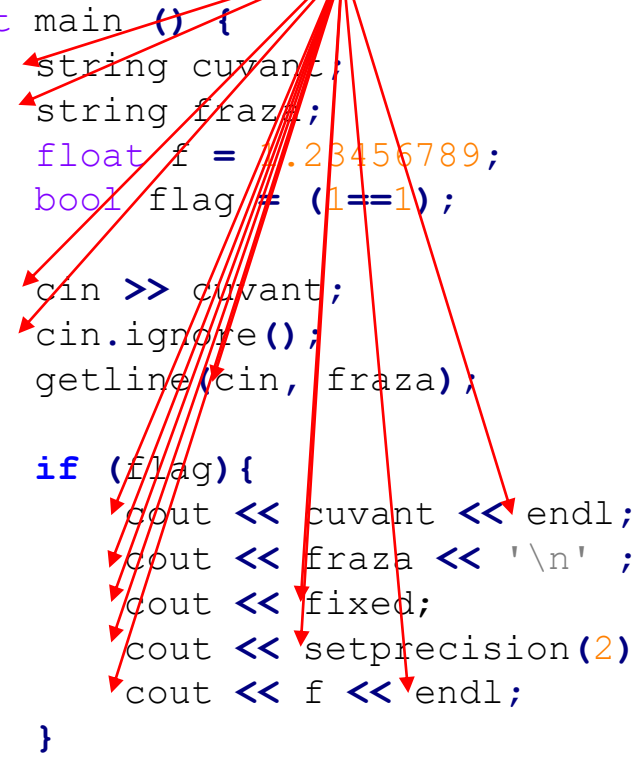
```
#include <stdio.h>
```

```
int main () {  
    char cuvant[20];  
    char fraza[20];  
    float f = 1.23456789;  
    int flag = (1==1);  
  
    scanf("%s", cuvant);  
    getchar();  
    scanf("%[^\\n]", fraza);  
  
    if (flag){  
        printf("%s\\n", cuvant);  
        printf("%s\\n", fraza);  
        printf("%.2f\\n", f);  
    }  
  
    return 0;  
}
```

```
#include <iostream>  
#include <string>  
#include <iomanip>
```

```
using namespace std;
```

```
int main () {  
    string cuvant;  
    string fraza;  
    float f = 1.23456789;  
    bool flag = (1==1);  
  
    cin >> cuvant;  
    cin.ignore();  
    getline(cin, fraza);  
  
    if (flag){  
        cout << cuvant << endl;  
        cout << fraza << '\\n' ;  
        cout << fixed;  
        cout << setprecision(2);  
        cout << f << endl;  
    }  
  
    return 0;  
}
```



C → C++

```
#include <stdio.h>
```

```
int main () {  
    char cuvant[20];  
    char fraza[20];  
    float f = 1.23456789;  
    int flag = (1==1);  
  
    scanf("%s", cuvant);  
    getchar();  
    scanf("%[^\\n]", fraza);  
  
    if (flag){  
        printf("%s\\n", cuvant);  
        printf("%s\\n", fraza);  
        printf("%.2f\\n", f);  
    }  
  
    return 0;  
}
```

```
#include <iostream>  
#include <string>  
#include <iomanip>
```

```
int main () {  
    std::string cuvant;  
    std::string fraza;  
    float f = 1.23456789;  
    bool flag = (1==1);  
  
    std::cin >> cuvant;  
    std::cin.ignore();  
    getline(std::cin, fraza);  
  
    if (flag){  
        std::cout << cuvant << std::endl;  
        std::cout << fraza << std::endl;  
        std::cout << std::fixed;  
        std::cout << std::setprecision(2);  
        std::cout << f << std::endl;  
    }  
  
    return 0;  
}
```