PROGRAMARE ORIENTATĂ OBIECT (C++)

Conf.univ.dr. Ana Cristina DĂSCĂLESCU

cristina.dascalescu@prof.utm.ro

Universitatea Titu Maiorescu

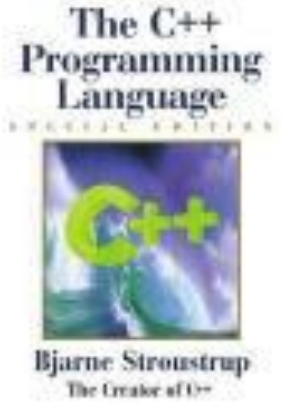
2023 – 2024

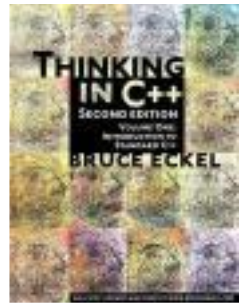
Conținutul tematic

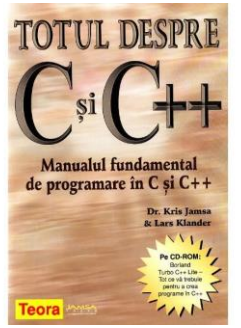
- Introducere în Programarea orientată pe obiecte
- Clase şi obiecte
- Supraîncărcarea operatorilor
- Moştenirea. Polimorfismul
- Biblioteca STL - Standard Template Library

Bibliografie

- Bjarne Stroustrup, *The C++ Programming Language*, Adisson-Wesley, 3nd edition, 1997
- Bruce Eckel, *Thinking in C++ ,* 3nd Edition, Prentice Hall 2016
- Liviu Negrescu: *Limbajele C și C++ pentru începători*, Vol. II, Editura Albastră, Cluj-Napoca, 2013
- Kris Jamsa, Lars Klander: *Totul despre C si C++, Manualul fundamental de programare in C si C++*, Editura Teora, 2015



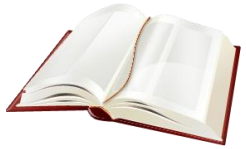




CONCEPTE DE BAZĂ ALE PROGRAMĂRII ORIENTATE OBIECT

➢Detalii organizatorice

➢Evaluare



- 60% Evaluare finală în sesiune (minim nota 5)
- 40% Proiect

Puncte slabe ale Programării Procedurale

➢**Nu se poate reutiliza cu ușurință un cod creat anterior!!!** 😟

- funcțiile pot avea referințe către alte funcții
- tipul parametrilor poate să fie diferit
- un program poate să conțină variabile globale

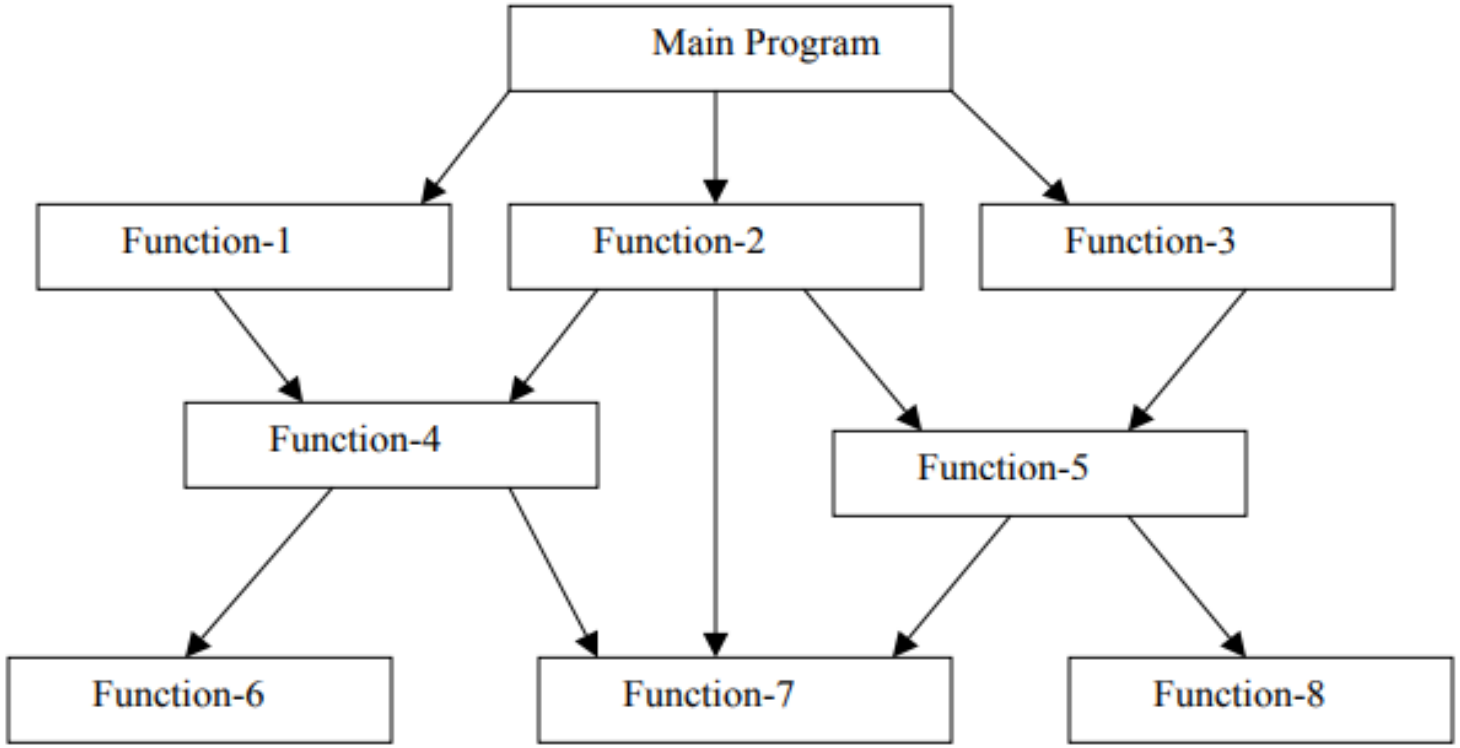
➢**Nu oferă un grad înalt de abstractizare!!!** 😟

➢**Datele sunt tratate separat de rutinele de programare!!!** 😟

Paradigme de programare

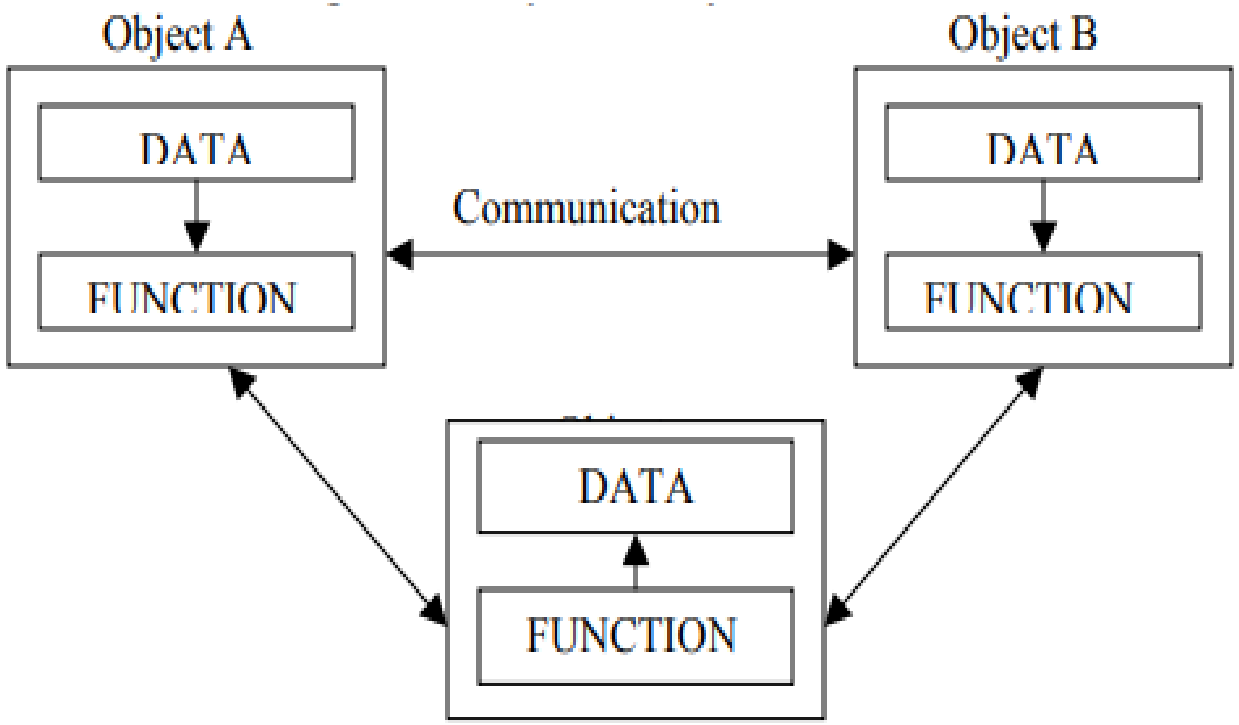
- **Paradigma programării structurate**: un program este privit ca o mulțime ierarhică de blocuri și proceduri (Pascal, C, C++)

Algoritm + Structuri de date = Program (Niklaus Wirth)



Paradigme de programare

- **Paradigma programării orientate obiect**: un program este privit ca o mulțime de obiecte care interacționează între ele. 😮

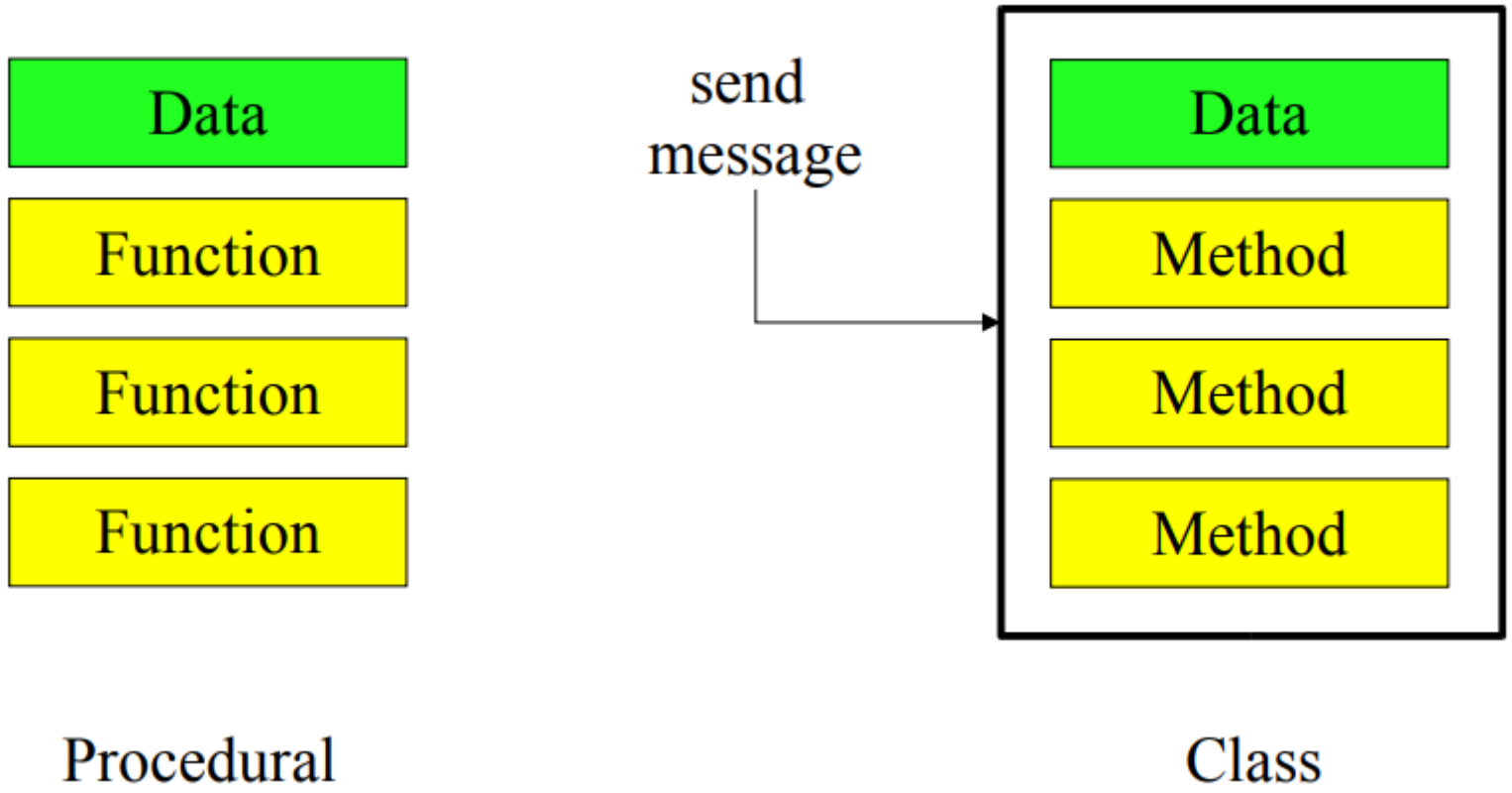


Concepte de bază POO

- Încapsularea
- Abstractizarea
- Ierarhizarea
- Polimorfismul

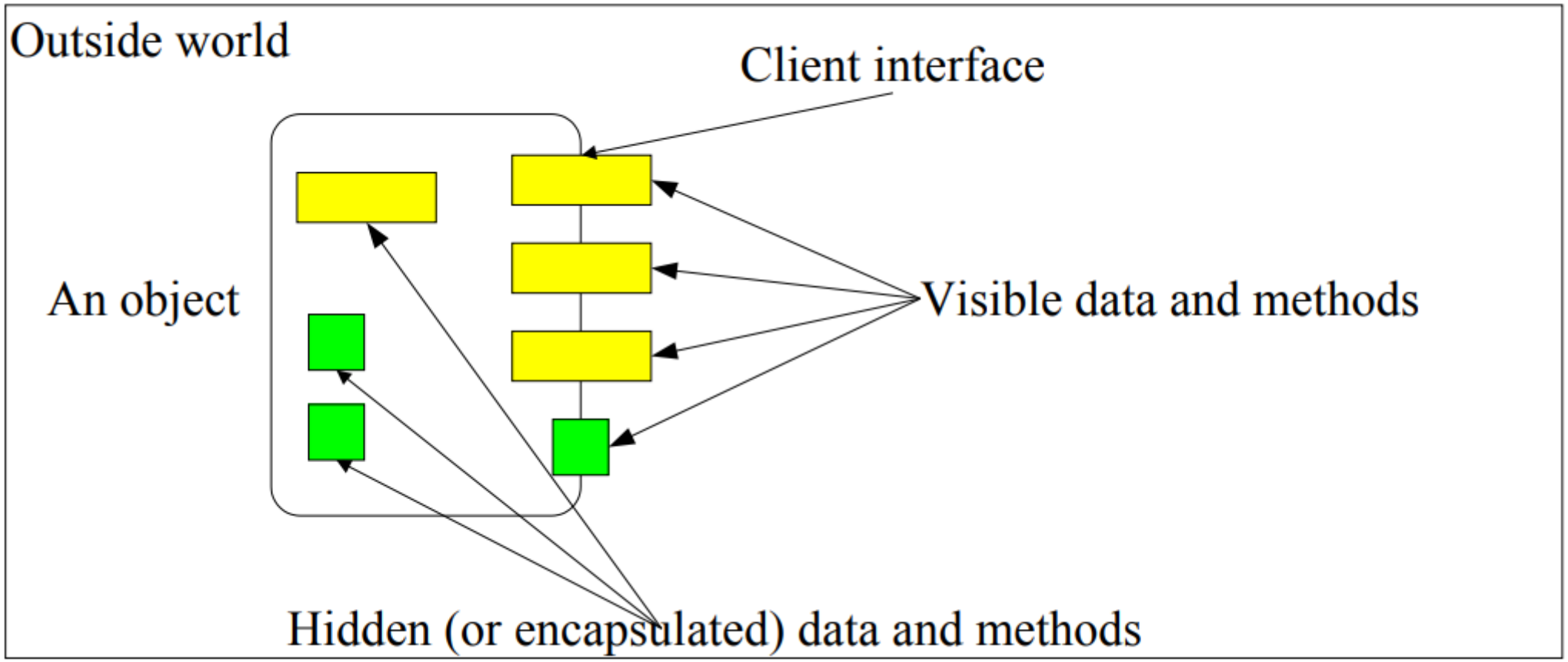
Încapsularea

- Mecanismul prin care datele și operațiile sunt înglobate sub forma unui tot unitar (obiect)



Încapsularea

- Datele pot fi accesate din afara entității (obiectului) numai prin intermediul operațiilor (funcțiilor/metodelor) publice!!!



Abstractizarea

- Procesul prin care se identifică datele și operațiile relevante pentru un concept din lumea reală.

Tipul Persoana - pentru aplicația Recensământ	Tipul Persoana – pentru aplicația Calcul Intreținere
▪ Nume ▪ prenume ▪ varsta ▪ localitate	▪ nume ▪ prenume ▪ suprafataLocuita ▪ NrPersoaneIntretinere
▪ numara ▪ afisare ▪ statictica	▪ calculIntretinere ▪ deduceri ▪ Afisare

Abstractizarea

➢ **Primul pas spre abstractizare...**

Tipul abstract de date „Persoana"

```
typedef struct {
char varsta;
char localitate;
} Persoana;
```

Structura de date

Instanţierea tipului abstract „Persoana"

```
Persoana s={"Popescu Emil”,23,„Rm Valcea"};
```

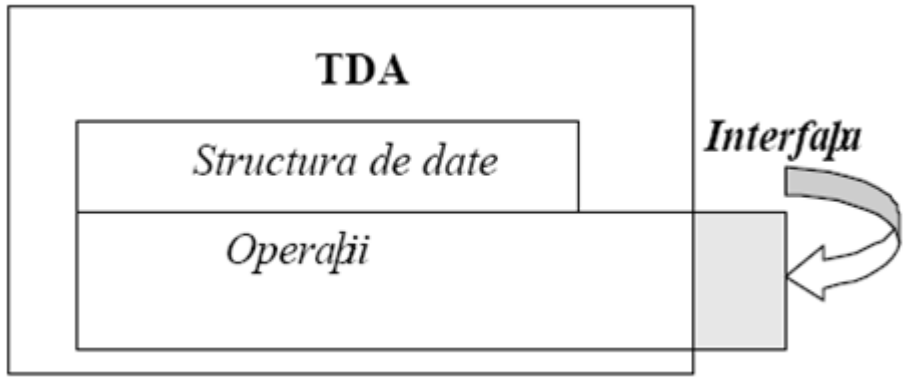
Abstractizarea

➢ **Urmatorul pas spre abstractizare!!!**

Date
nume;
varsta;
localitate;

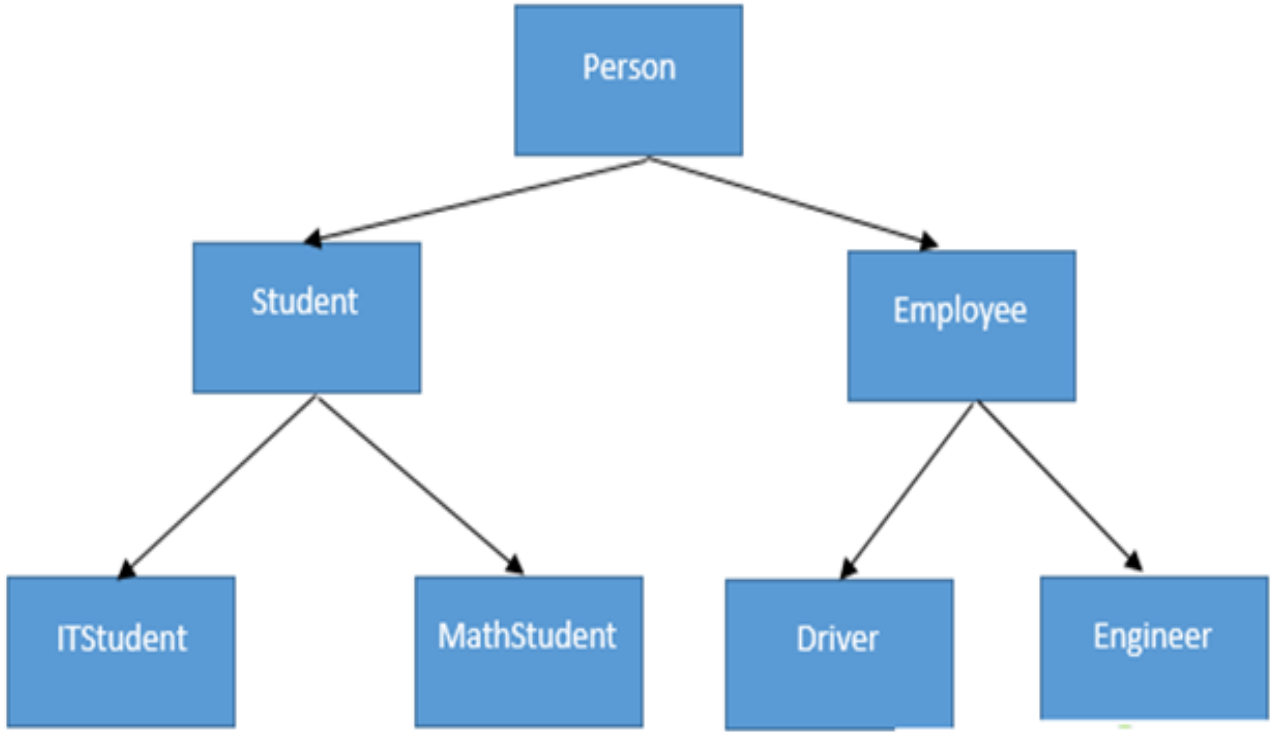
Funcții
inițializare
afisare
calculIntretinere

Tip Abstract de date (TAD)



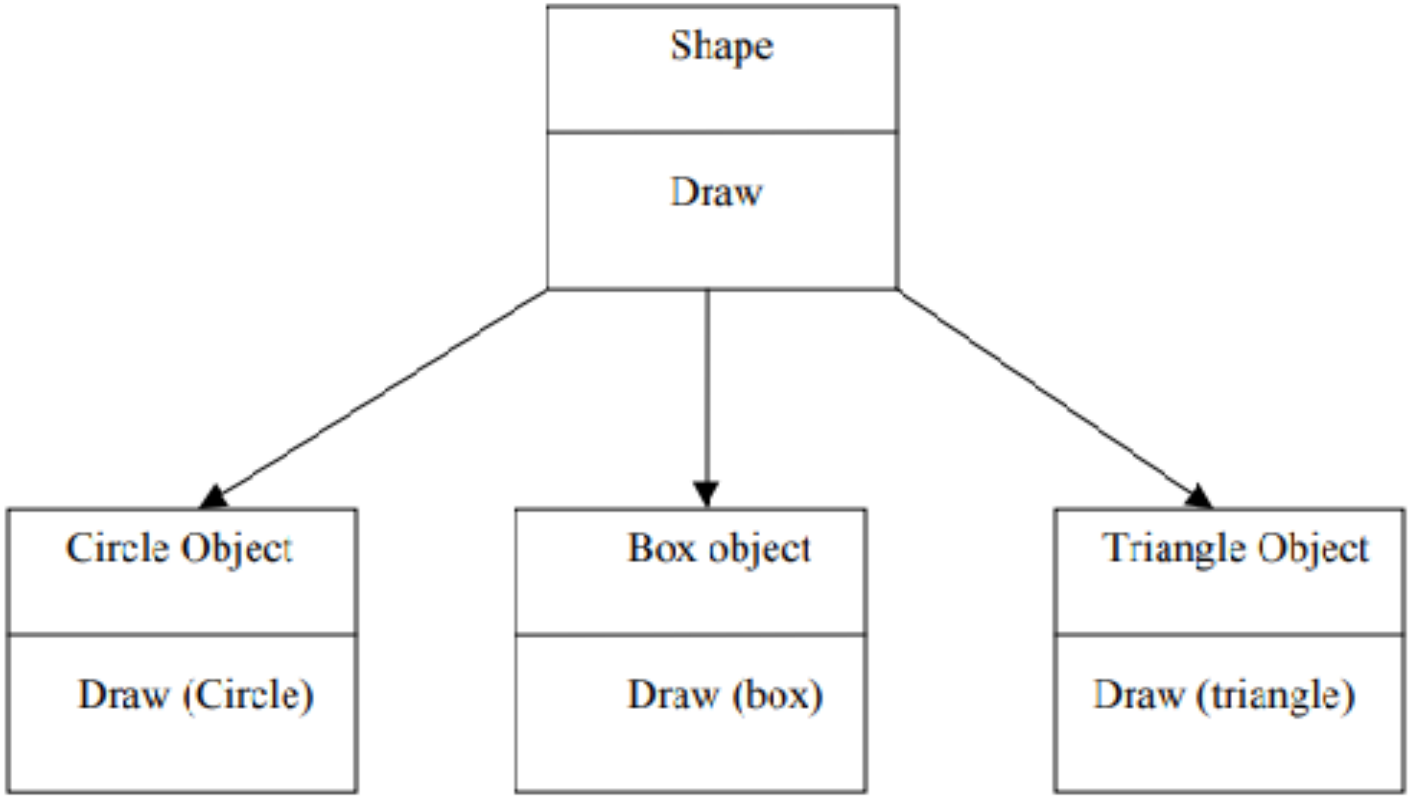
Moștenire

- Reprezintă un mecanism de reutilizare a unui cod creat anterior!!!
- Anumite tipuri abstracte de date pot fi create din alte TDA, numite TDA de bază. Noile TDA se numesc TDA derivate.



Polimorfism

- Proprietatea unei enități (obiect) de a avea comportament diferit, în funcție de context.



Limbajul C++

- Limbajul C++ a fost elaborat de către **Bjarne Stroustrup** în anul 1980 în laboratoarele Bell, New Jersey.
- Inițial a fost denumit **C cu clase**, iar apoi **în anul 1983** a fost denumit C++.
- De la apariția sa a trecut prin mai multe revizii, în 1985, 1989, 1994 etc.
- În anul 1994 este standardizat ANSI.
- Actualul standard ISO C++ 23 (iulie 2022)

Carcteristici ale limbajului C++

- Este un limbaj poliparadigmă: suportă programarea orientată pe obiecte și programarea procedurală
- Este o extensie a limbajului C
- Este un limbaj compilat
- Este dependent de platformă
- Introduce concepte precum:
  - Clase și obiecte
  - Funcții virtuale
  - Supraîncărcarea operatorilor
  - Moștenirea multiplă
  - Șabloane (template)
  - Tratarea excepțiilor

Operații de intrare/ieșire pentru tipuri predefinite

- Sunt bazate pe fluxuri (stream).
- **Stream**: o entitate logică în care/din care se introduc/extrag informații

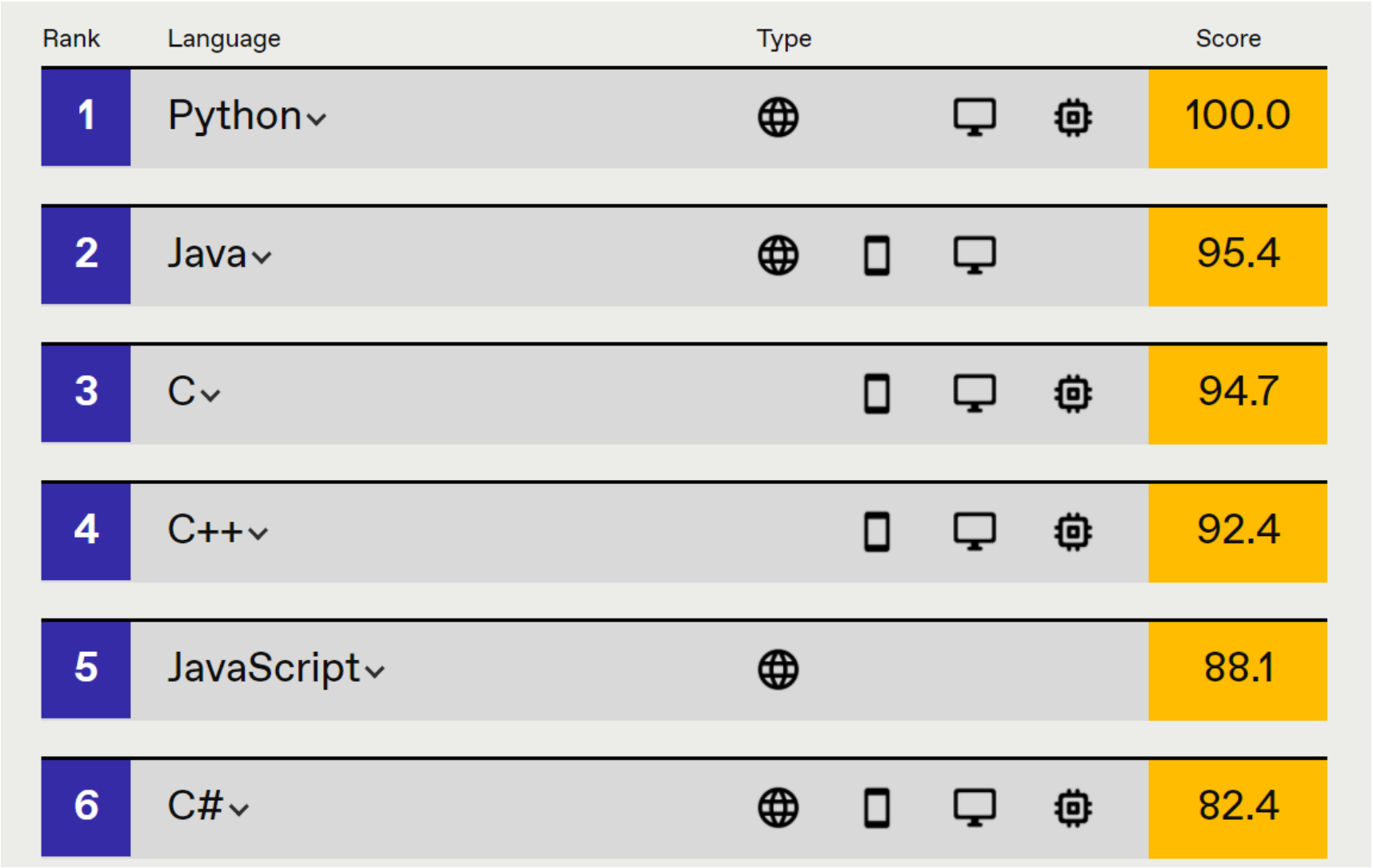
- Strem-uri standard:
  - **cin**: intrare standard – tastatură
  - **cout:** ieșire standard - ecran
  - **cerr:** ieșire standard pentru eroare

- Operaţiile de I/O din C++ se efectuează folosind funcţiile operator de inserţie >> şi operator de extragere <<.

- Fișierul iostream.h definește biblioteca de clase necesare fluxului de intrare/ieșire standard.

Top IEEE (Institute of Electrical and Electronics Engineers) Spectrum

https://spectrum.ieee.org/top-programming-languages/



Rank	Language	Type	Score
1	Python		100.0
2	Java		95.4
3	C		94.7
4	C++		92.4
5	JavaScript		88.1
6	C#		82.4

Operații de intrare/ieșire

➢Sintaxa operaţiei de intrare (citire):

```
cin >> var_1 >> var_2 >> … >> var_n;
```

- Funcția **get(char *sir, int dim_buffer)**
  tratează spațiile albe la fel ca orice alt caracter

➢Sintaxa operaţiei de ieşire (scriere, afișare):

```
cout << expr_1 << expr_2 << … << expr_p;
```

- endl: trimite la o nouă linie fluxul de ieșire

- Manipulatori pentru formatarea afișării datelor **iomanip.h**:
  - setprecision (int nr_cifre): setează numărul de zecimale
  - setw(int dim): stabilește dimensiunea câmpului
  - setfill(char c): setează caracterul de umplere

Operatori specifici limbajului C++

➢Operatorul de rezoluție (de apartenență) ::

- Permite accesul la o variabilă globală, redefinită ca o variabilă locală

```
int var_globala=1;
int main()
{
    int var_globala=2;
    cout<<var_globala;
    cout<<::var_globala;
}
```

Se va afișa 2

Se va afișa 1

Alocarea dinamică a memoriei

➢Operatorul de alocare dinamică a memoriei: **new**

- Alocarea unei variabile

```
Tip *var=new Tip;
Tip *var=new Tip(valoare);
```

- Alocarea unui tablou de date

```
Tip *v=new Tip[dim];
```

➢Operatorul de eliberare a zonei de memorie: **delete**

```
delete var;
delete []v;
```

Tipul referință

- O referinţă (*reference*) este un nume alternativ al unui obiect (variabilă)
- Pentru a crea o referință se utilizează operatorul de referențiere **&**

Sintaxa

```
tip var;
tip& alias=var;
```

Exemplu

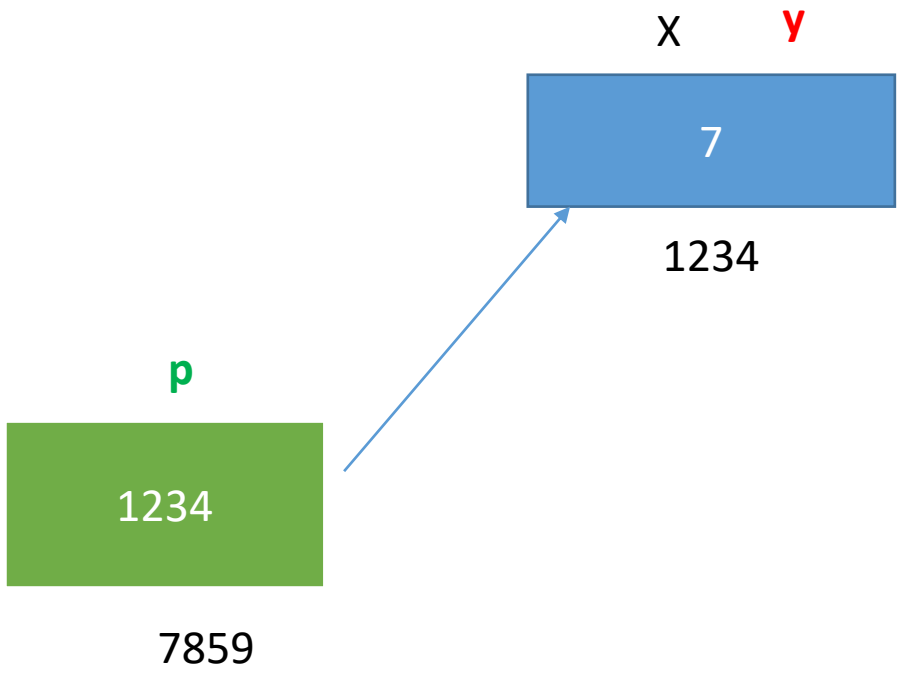
```
int x = 1;
int& r = x; // r şi x se referă la aceeaşi zonă de memorie
int y = r; // y = 1
r++; // x = 2
```

Tipul referință

```
int x=7;
```

```
int *p = &x;
```

```
int &y=x;
```



Apelul prin referință

- O referinţă este utilizată ca argument pentru o funcţie care poate să modifice permanent valoarea acestuia.

Exemplu

```
void incr(int x)
{
   x++;
}
void f()
{
   int i = 1;
   incr(i);
   cout<<i;
}
```

```
void incr(int* x)
{
     (*x)++;
}
void f()
{
   int i = 1;
   incr(&i);
   cout<<i;
}
```

```
void incr(int& x)
{
   x++;
}
void f()
{
     int i = 1;
     incr(i);
     cout<<i;
}
```