M. Caramihai, © 2020

STRUCTURI DE DATE & ALGORITMI

CURS 1

Notiuni introductive

Scurt istoric (1)

- **Abordare matematica**
  - Leibniz (1674) dezvoltarea unui limbaj universal capabil descrie / resolve orice problema (orive rationament are la baza un set de reguli de baza)
  - Boole (~1800) – calcul binar
  - Hilbert (1928) – ideea unei metode “mecanice” de verificare a adevarurilor matematice
  - Turing (1930) masina Turing (Obs: nu poate rezolva problema lui Hilbert)

Scurt istoric (2)

- **Abordare inginereasca:**
  - Abacul (cca 3000 i.Ch.)
  - Bl. Pascal (1672) - realizarea de calcul mecanic, operatii de baza: +, -, x, :
  - Ch Babage (1837) – prima masina de calcul programabila (a fost realizata fizic in 1991)
  - Konrad Zuse (1938, Germania) primul calculator programabil (mecanic) bazat pe logica booleana si calculi in virgula mobila

Scurt istoric (3)

- **Abordare inginereasca (cont):**
  - Mark 1 (1943) – la universitatea Harvard, realizat de IBM: primul calculator universal (electromecanic). Avea 765.000 componente si cantarea 4.500kg
  - ENIAC (1946), Electronic Numerical Integrator and Computer: 5000 op/sec. Cantarea 30.000kg

Ciclul de viata al unui proiect (informatic)

- Analiza problemei
- Stabilirea cerintelor de rezolvare
- Definirea programului
- Proiectare program
- Implementare
- Testare
- Operare / mentenanta

Modelul algoritmic

- Ce este *Computer Science ?*
- Ce este programarea ?
- Algoritmi
  - Definitii
  - Proprietati
  - Descriere
  - Exemple
  - Componente

Ce este *Computer science* ?

- Nu este o stiinta in sensul traditional al cuvantului
- Se ocupa cu studiul algoritmilor si modul lor de utilizare (i.e. proprietati matematice, support hardware, descriere lingvistica, model de aplicare)
- NU se ocupa cu studiul calculatoarelor

- Ingineria software este importanta:
  - Analiza si proiectarea sistemelor
  - Programare
  - Testare si mentenanta

De ce ?

Strategii de rezolvare a problemelor

- Reverse engineering
- Prin analogie
- Prin rezolvarea de sub-probleme

Ce este *programarea* ?

- Programarea necesita doua abilitati:
  - Gandirea algoritmica
  - Cunoasterea sintaxei unui limbaj de programare
- A invata sintaxa unui limbaj de programare este problema cea mai simpla !

A invata *Computer science*

- Se recomanda utilizarea *pseudo-codului* pentru invatarea gandirii algoritmice.
- Aplicarea pseudo-codului in orice sintaxa este usoara!
- Nu este o solutie “modificarea & compilarea” codului !
- Multi programatori incearca sa programeze pe principiul “trial and error”.

Definirea algoritmilor

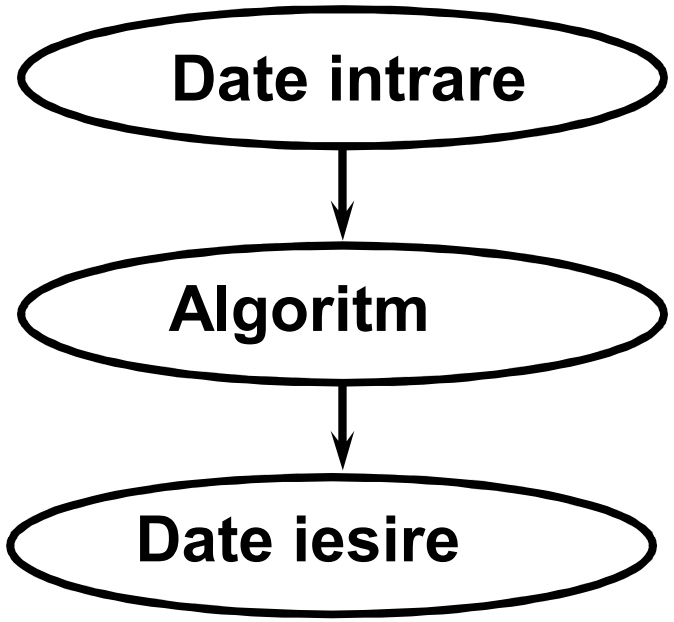
Definitie: descrierea logica a pasilor ce permit determinarea solutiei complete pentru o problema data, solutie ce trebuie implementata intr'un interval finit de timp

Termenul "algoritm" a fost definit in sec IX in Persia de Abdullah Jafar Muhammad ibn Musa Al-khowarizmi

Algoritmii contin:

- **Date**
- **Instructiuni**

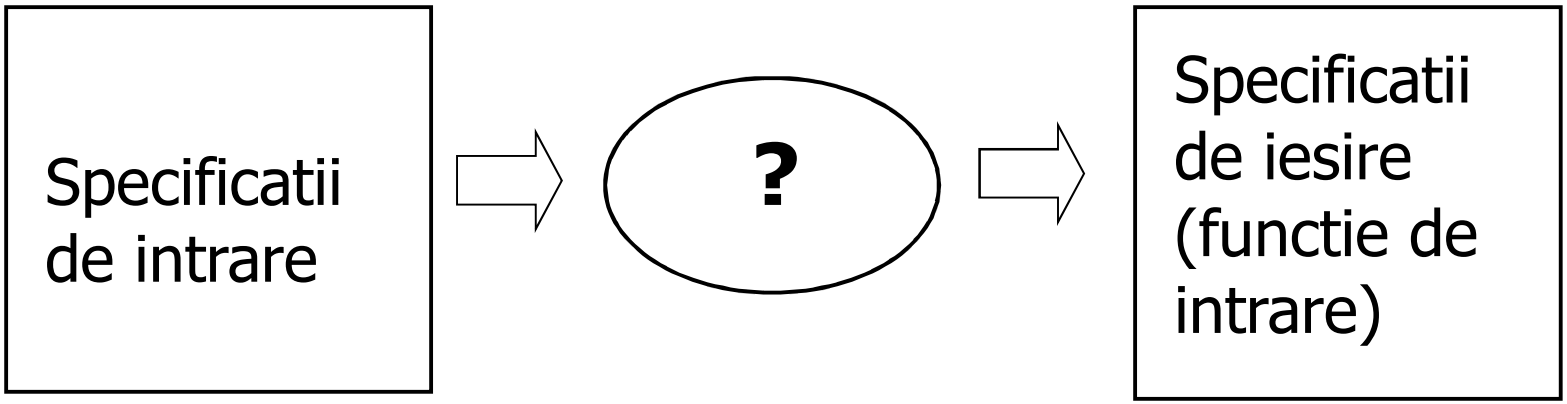
Algoritmi & Programe



Algoritm:
Pot fi implementati de oameni sau masini
Pot fi reprezentati in orice limbaj
Sunt abstracti

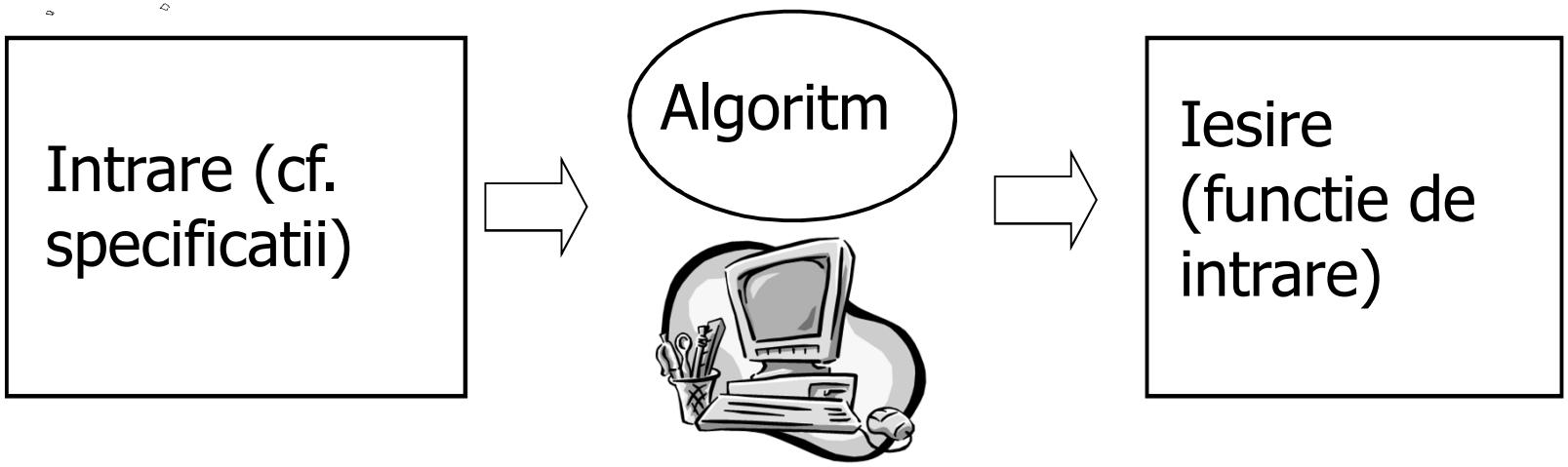
Program
Se implementeaza doar pe o masina
Trebuie sa fie exprimat intr'un limbaj de programare

Algoritmii: o abordare sistemica



- Poate exista un numar infinit de instante de intrare ce satisfac specificatiile

Solutia algoritmica



- Algoritmii descriu operatiile ce actioneaza asupra marimilor de intrare
- Pot exista mai multi algoritmi corecti (solutii) pentru o aceasi problema

Ce trebuie stiut despre algoritmi

- **Caracteristici functionale: timp de rulare, necesarul de memorie, complexitate, eficienta**

- **Caracteristici comportamentale :**
  - Trebuie sa aiba **un numar finit de intrari** si **cel putin o iesire**
  - Este **bine definit:** fiecare instructiune este clara si ne-ambigua
  - Este **finit:** algoritmul se finalizeaza intr'un numar finit de pasi

Abstractizarea

- Exemplul atlasului →scara de reprezentarea marimilor
- Definitie: procedurade mascarea detaliilorunuiobiectin scopul accentuarii / punerii in evidenta a altor aspecte / detalii / structuri
- Nivele de abstractizare
  - Cel mai de sus: comunitatea obiectelor ce trebuie sa interactioneze unele cu altele in vederea indeplinirii unui scop comun
  - Valabil in anumite limbaje: grup de obiecte ce lucreaza impreuna si care sunt grupate in *unit*-uri, d.e. *package* (Java), *name spaces* (C++) si *units* (Delphi)
  - Nivelul de interactiune intre 2 obiecte (interactiune *client / server, C/S*). In acestcaz exista 2 sub-nivele de abstractizare:
    - →Abstractizare client → server
    - →Abstractizare server → client

Abstractizare & calculatoare

- Se refera la gruparea logica a conceptelor si obiectelor
- Defineste / implementeaza idei generale
- Izoleaza detaliile
- Ajuta la imbunatatirea intelegerii si reproducerii algoritmilor

Abstractizare: exemple

- Algoritmi
- Nume de variabile
- Proceduri si functii
- Structuri de date
- Limbajul de calculator !

Ce este gresit in acest algoritm ?

Instructiuni de utilizare:

1. **Umeziti parul**
2. **Aplicati sampon pe par**
3. **Spalati**
4. **Clatiti**

Repetati (cel putin odata)

Ce inseamna un algoritm *bun* ?

Un algoritm *bun* este:

- **Precis**
- **Ne-ambiguu**
- **Complet**
- **Corect**
- **Simplu**
- **Contine nivele de abstractizare diferite**

Descrierea algoritmilor

- Limbajul natural
- Imagini
- Pseudo-cod sau un limbaj de programare

Componentele algoritmilor

Orice algoritm contine in general 5 elemente:

- Structuri de date pentru pastrarea datelor
- Instructiuni ce schimba valorile datelor
- Expresii conditionale pentru luarea deciziilor
- Structuri de control pentru actiuni
- Module permit managementul algoritmilor prin intermediul structurilor abstracte (i.e. gruparea componentelor)

Proprietatile algoritmilor

- Are 0 sau mai multe intrari
- Are cel putin o iesire.
- Fiecare instructiune este neambigua.
- Se termina intr'un numar finit de instructiuni.
- Fiecare pas este fezabil.

Clasificarea algoritmilor (1)

- *1*
  - Timp constant de rulare
  - Marea majoritate a instructiunilor dintr'un program sunt executate odata sau de un numar limitat de ori.
- log *N*
  - daca timpul de rulare al unui program este *logaritmic*, programul va rula ceva mai incet pe masura ce N creste.

Clasificarea algoritmilor (2)

- *N*
  - Timp *linear* de rulare.
  - Reprezinta situatia optimala pentru un algoritm ce trebuie sa proceseze N intrari (sau sa furnizeze N iesiri).
- *N* log *N*
  - are la baza “spargerea” unui program in probleme mai mici, rezolvarea lor independenta si “asamblarea” solutiei.

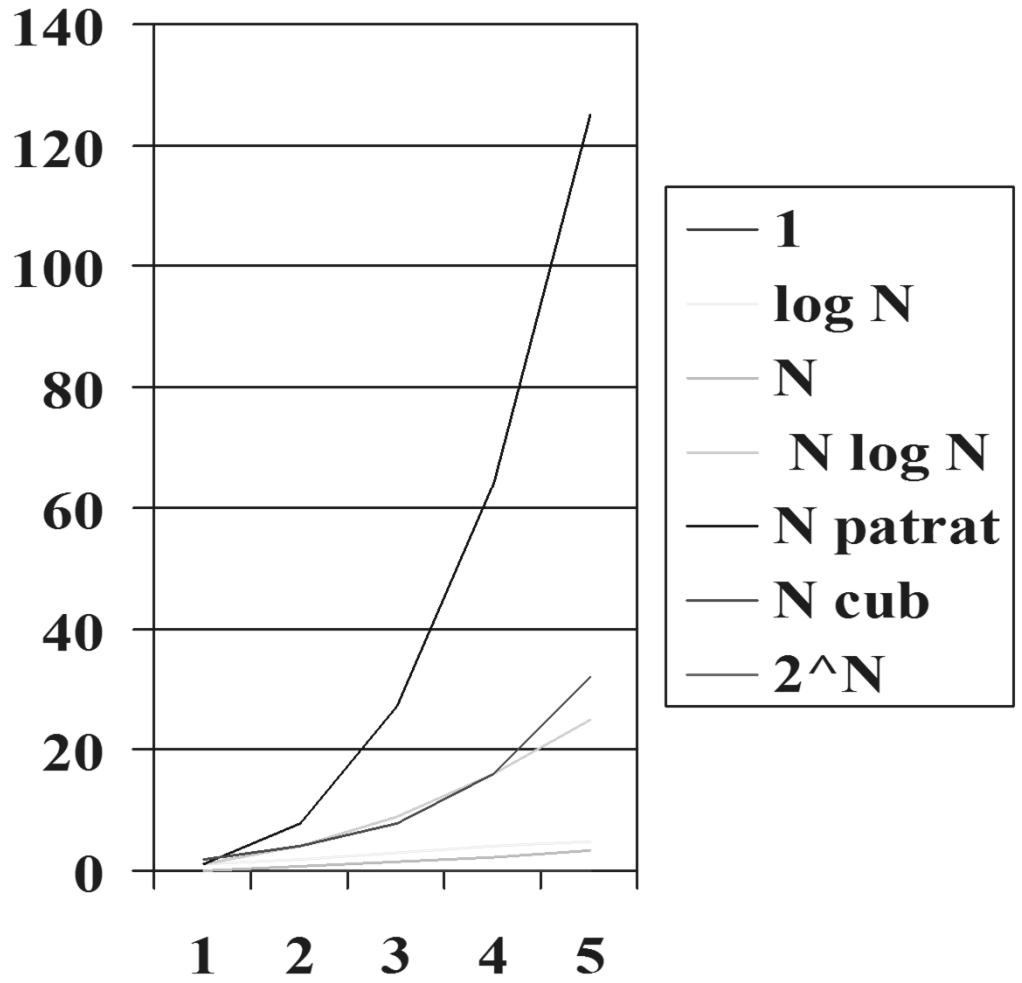
Clasificarea algoritmilor (3)

- N^2
  - Timp *patratic* de rulare pentru un algoritm
  - Util a fi utilizat in cazul problemelor de mici dimensiuni.
  - Este tipic algoritmilor ca proceseaza toate perechile de date de intrare (inclusiv bucle imbricate).
- N^3
  - un asemenea algoritm proceseaza date triple de intrare (inclusiv bucle triplu imbricate) si are timp cubic de rulare; recomandabil doar in cazul problemelor de dimensiuni foarte mici.

Clasificarea algoritmilor (4)

- 2^N
  - putini algoritmi de acest tip (timp exponential de rulare) sunt recomandati pentru utilizare practica; in general provin din algoritmi de tip “brute-force”.
  - cand N dubleaza, timpul de rulare este la patrat !

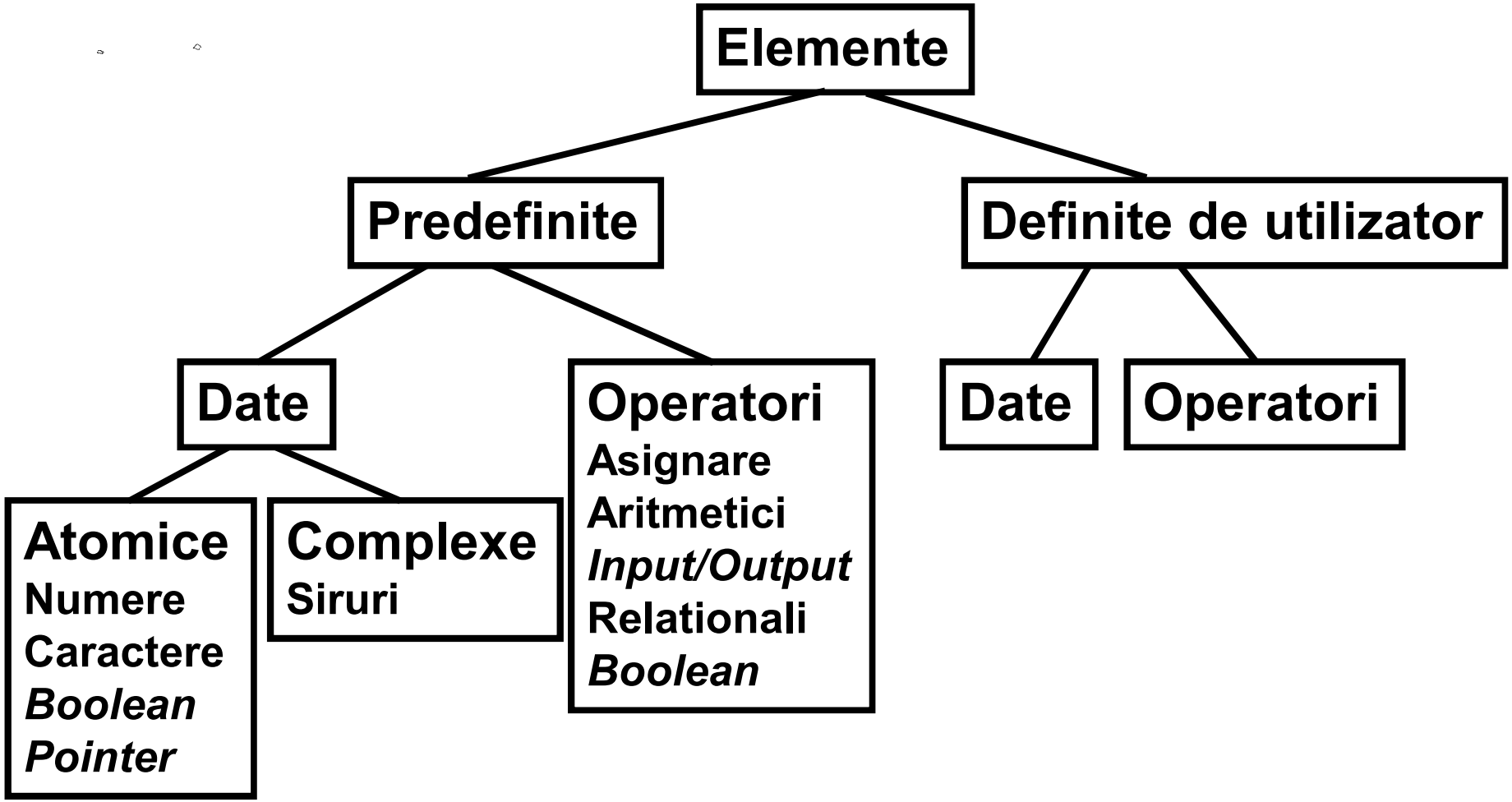
Clasificarea algoritmilor (5)



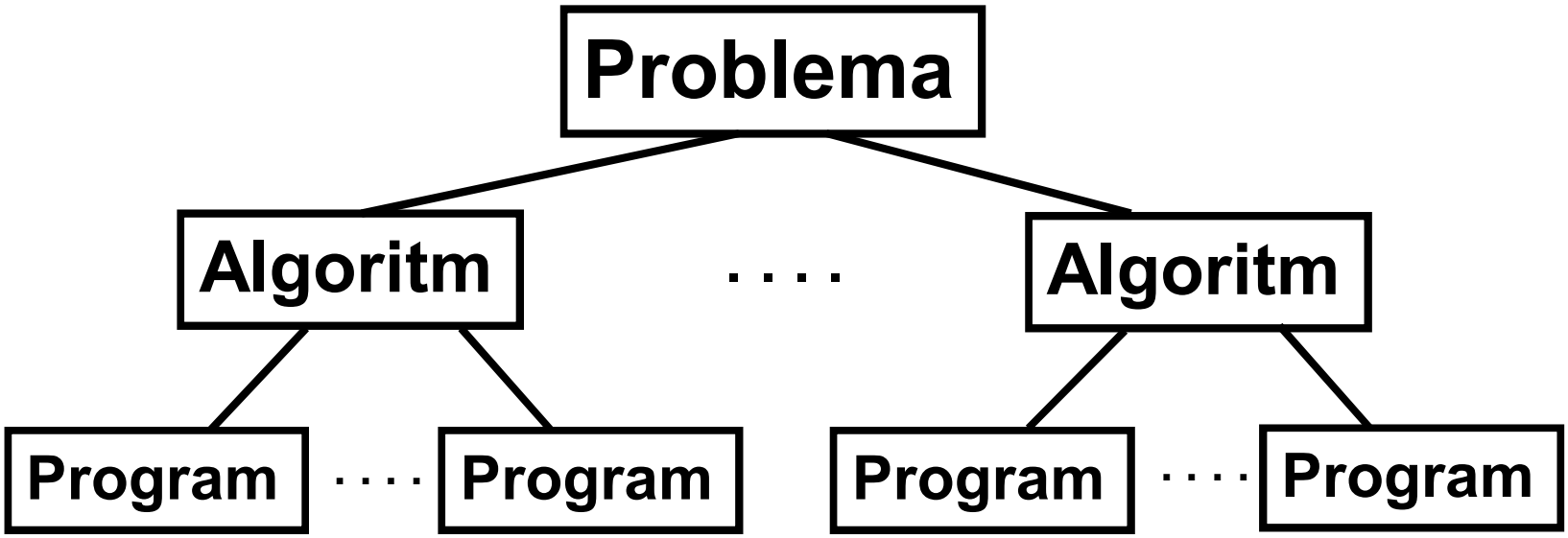
Limbaje de programare: generalitati

- Permit programatorului sa descrie algoritmii
- Sunt destinate
  - oamenilor
  - calculatoarelor (format binar)
- Consta in:
  - Elemente predefinite
  - Elemente definite de utilizator

Limbaj de programare



Relatia intre probleme / algoritmi / programe



Structuri de date

Structurile de date (SD) sunt elemente ce inmagazineaza date. Reprezinta o modalitate de organizare a datelor in vederea accesului la acestea.

Cea mai simpla SD este de tip “atomic” deoarece contine o singura valoare si nu poate fi divizata in elemente mai mici

Alte SD “complexe” pot contine mai multe elemente de tip date si sunt construite pe baza elementelor atomice.

Tipuri de date predefinite

4 tipuri atomice de date predefinite:

- **Caracter** (***char***)
- **Numar** (***num***)
- ***Boolean*** (***boolean***)
- ***Pointer*** (***ptr***)

1 tip complex de date predefinite:

- **Sir** (***string***)

Memoria

- Reprezentare sub forma de 1 si 0
- Abstractizarea detaliilor
  - Crearea unor spatii de memorie pentru anumite tipuri de informatii
  - Accesarea acestor locatii prin intermediul unor nume specifice