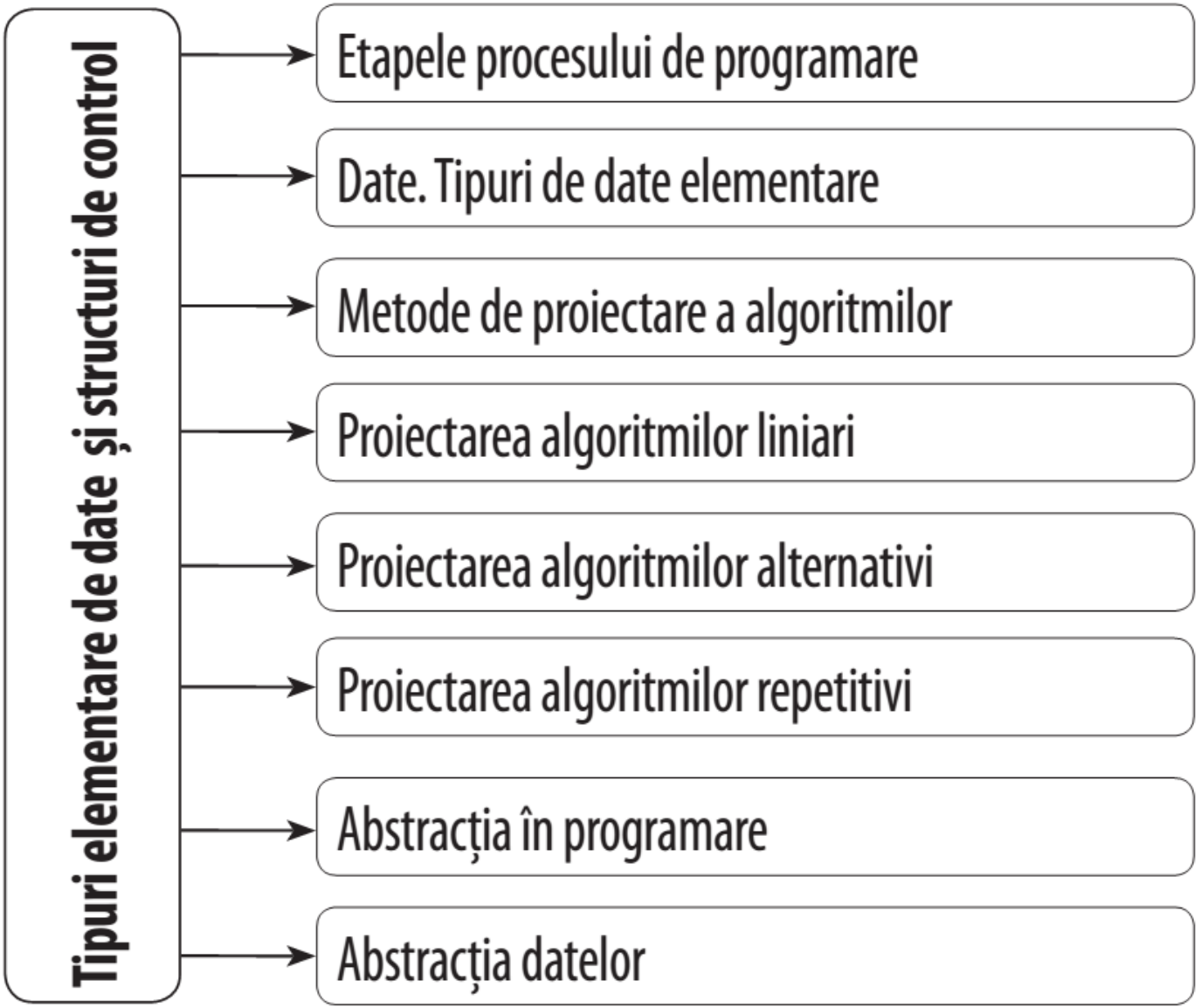
Bazele programării I

Despre mine

- Moglan Diana, dr., conf. univ.
- Adresa e-mail: *mogdiana@gmail.com*
- Catedra de matematică și informatică, bl. I, aula 145, tel. 0 231 52 488

Structura modulului inițial



Clasele de situaţii

Clasa de situații	Exemple de situații
Elaborarea algoritmilor liniari	Introducerea unui set de numere naturale, întregi sau reale de la tastatură și realizarea operațiilor aritmetice cu ele; Introducerea unui set de caractere de la tastatură și realizarea operațiilor cu ele.
Elaborarea algoritmilor cu structură alternativă	Identificarea valorii maxime (minime) dintr-un set de numere introduse de la tastatură; Realizarea diferitor prelucrări ale datelor în funcție de îndeplinirea unor condiții; Realizarea diferitor calcule în funcție de valoarea de adevăr a unei expresii logice.
Elaborarea algoritmilor repetitivi cu un număr cunoscut de repetări	Prelucrarea unui șir din *n* date elementare introduse de la tastatură; Generarea primilor *n* termeni ale unei progresii aritmetice (geometrice); Identificarea valorii maxime (minime) dintr-un șir de *n* numere; Identificarea divizorilor unui număr.

Clasele de situaţii

Elaborarea algoritmilor repetitivi cu un număr necunoscut de repetări	Prelucrarea unui şir de date elementare introduse de la tastatură până la îndeplinirea unei condiţii; Identificarea valorii maxime (minime) dintr-un şir de date elementare citite de la tastatură până la îndeplinirea unei condiţii; Realizarea programelor-meniu.
Elaborarea procedurilor fără parametri	Afișarea unui desen static.
Elaborarea procedurilor cu parametri de intrare	Afișarea ariei unei figuri geometrice; Afișarea rezultatului prelucrării unui sir de numere.
Elaborarea procedurilor cu parametri de ieșire	Identificarea rezultatelor prelucrării unui şir de date citite de la tastatură; Generarea și identificarea rezultatelor prelucrării unui şir de date.
Elaborarea funcţiilor	Elaborarea funcţiilor logice; Identificarea rezultatului prelucrării unui şir de date.

Cerinţe

- Repartizarea orelor: teorie – 44 ore (22 perechi), laboratoare – 46 ore (23 perechi).
- Prezența la laborator și teorie: minim 70% din ore.
- Rezolvarea însărcinărilor și problemelor prevăzute la laborator.
- Studierea materialului teoretic.

Evaluarea

- Evaluarea va fi realizată sub formă dinamică (pe parcursul semestrului) şi finală (la finele semestrului).
- Pe parcursul semestrului trebuie să fie susţinute 3 lucrări de control (teste) la calculator.
- Evaluarea finală se realizează sub formă de test la calculator.
- Nota la disciplină este calculată ca media ponderată a notei de la examen (40%) şi a mediei notelor de la lucrările de laborator şi de control (60%).

Bibliografie

1. Pătrășcoiu, Octavian; Marian, Gheorghe; Mitroi, Nicolae. *Elemente de grafuri și combinatorică. Metode, algoritmi și programe*. București: Ed. All, 1994. 224 pag.

2. Thomas H., Cormen; Charles E., Leiserson; Ronald R., Rivest. *Introducere în algoritm*. Cluj: Ed. Libris Agora, 2000. 880 pag.

3. Deinego, Nona; Cabac, Valeriu. *Bazele programării. Tipuri elementare de date şi structuri de control*. Vol. 1. Bălți: Presa universitară Bălţeană, 2013. 222 pag.

Bibliografie

1. Дейнего, Нона; Кабак, Валерий; Моглан, Диана. *Основы программирования. Элементарные типы данных и управляющие структуры*. Бэлць: Бельцкая университетская пресса, 2016. 234 стр.

2. Райли Д. *Использование языка Модула-2. Вводный курс*. Москва: Изд-во Мир, 1993. 606 стр.

3. Никлаус Вирт. *Алгоритмы и структуры данных*. СПб: Изд-во Невский диалект, 2001. 352 стр.

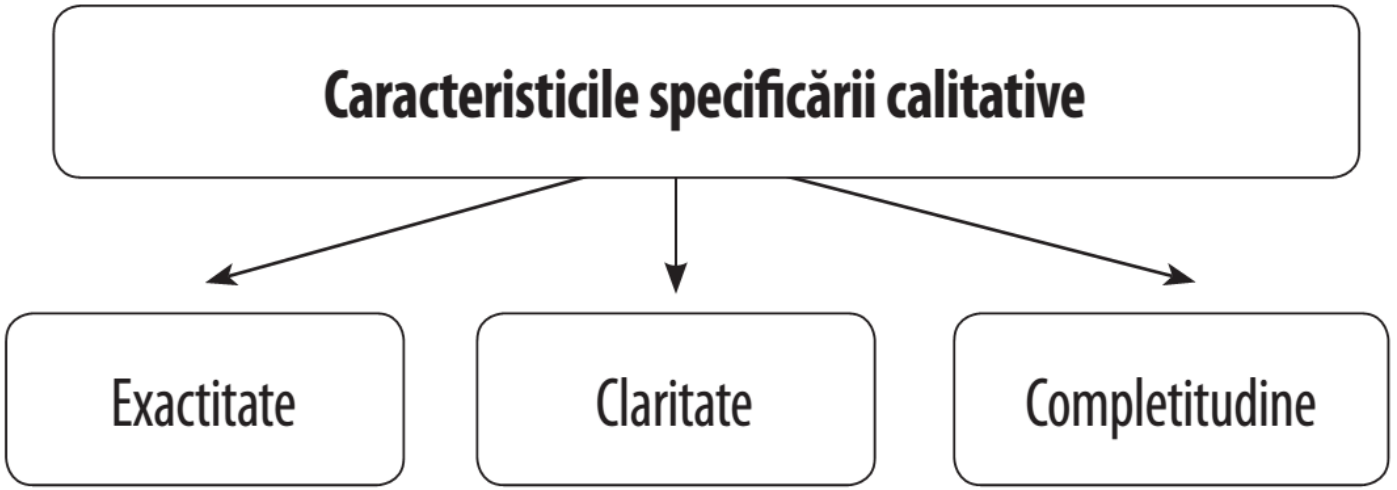
4. Иванова, Г. *Основы программирования*. Москва: Изд-во МГТУ им. Н.Э. Баумана, 2002. 416 с.

Etapele procesului de programare

- *specificarea problemei* (precizarea completă a problemei de rezolvat);
- *proiectarea algoritmului de rezolvare a problemei*;
- *codificarea algoritmului* (programarea propriu-zisă);
- *exploatarea şi întreţinerea programului*.

Specificarea problemei

- Specificarea problemei are funcţia de contract dintre client (cel ce comandă elaborarea unui program) şi programator. La această etapă se analizează cerinţele clientului referitoare la funcționalitatea viitorului program.
- Analiza cerinţelor permite programatorului să specifice funcţiile şi performanţele produsului de dezvoltat, să stabilească interfaţa lui.



Modelul specificării problemei

- denumirea problemei;
- descrierea problemei;
- introducerea datelor iniţiale;
- afişarea rezultatelor;
- descrierea erorilor;
- exemplu.

Exemplu: *Introducerea a trei numere şi afişarea lor ordonată*.

Exemplu de specificare a problemei

Denumirea problemei

Sortarea a trei numere întregi.

Descrierea problemei

De la tastatură se introduc trei numere întregi care apoi se afișează în ordine crescătoare.

Introducerea datelor inițiale

Se introduc trei numere întregi, fiecare din rând nou.

Afișarea rezultatelor

Se afișează numerele introduse într-un rând în ordine crescătoare.

Exemplu de specificare a problemei

Erori

- *dacă nu au fost introduse toate cele trei numere, programul va aștepta următoarea introducere;*
- *dacă au fost introduse mai mult de trei numere, atunci numerele introduse în plus se vor ignora;*
- *dacă într-un rând au fost introduse mai multe numere, atunci programul afișează un mesaj de eroare şi se termină.*

Exemplu

Introduceți 3 numere întregi, fiecare din rând nou:
3
-2
1
Numerele ordonate: -2 1 3

Schema etapei specificării problemei

Noţiunea de algoritm

Exemplu: *Să presupunem că trebuie să mergem la magazin să cumpărăm un produs*. Ce trebuie să facem?

Pas 1: luăm banii necesari;

Pas 2: ne îndreptăm către magazin;

Pas 3: solicităm produsul;

Pas 4: plătim;

Pas 5: venim cu produsul acasă.

Noţiunea de algoritm

Este un algoritm:

- care conține 5 etape (deci un număr finit de operații);
- care are scrise etapele în ordinea în care trebuie executate (deci sunt ordonate);
- în care fiecare etapă este explicată în cuvinte (deci este complet definită);
- care pornind de la ceva (în cazul nostru bani) obţinem ceea ce dorim (un produs).

Proiectarea algoritmului

- La această etapă specificarea problemei se transformă în algoritm.

Prin **algoritm** se înţelege o succesiune determinată de operaţii precise, care permite rezolvarea problemelor dintr-o clasă dată într-un număr finit de paşi.

- Un algoritm de calcul este o mulțime de operații determinate care se execută într-o ordine bine stabilită asupra unor date de intrare şi conduc într-un timp finit la un set de date de ieşire.

Proprietăţile algoritmului

Generalitate

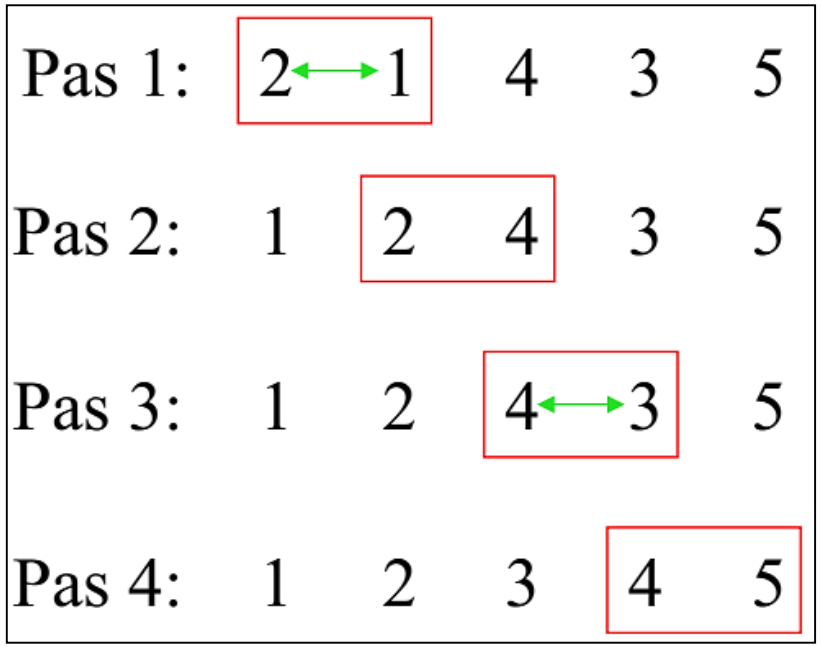
Algoritmul trebuie conceput nu pentru rezolvarea unei probleme particulare, ci pentru soluționarea unei clase de probleme căreia îi aparține problema respectivă (ecuații de gradul I, suma, produsul).

Exemplu:

Sa considerăm problema ordonării (sortării) crescătoare a unui șir de valori numerice.

Generalitate

Metoda



Descriere

- Compară primele două elemente; dacă nu sunt în ordinea dorită se interschimbă;
- Compară al doilea cu al treilea și aplică aceeași strategie….;
- Continuă procesul până la ultimele două elemente din secvență.

Generalitate

Este acest algoritm suficient de general? Asigură ordonarea crescătoare a oricărui șir de valori?

Răspuns: NU

Contraexemplu:

3 2 1 4 5
2 3 1 4 5
2 1 3 4 5
2 1 3 4 5

Metoda nu poate fi considerată un algoritm general de sortare. Pentru a realiza sortarea completă e necesară reluarea procesului de parcurgere a secvenței:

2 1 3 4 5 -> 1 2 3 4 5 -> 1 2 3 4 5 -> 1 2 3 4 5 -> 1 2 3 4 5

Proprietăţile algoritmului

Finitudine

Operațiile algoritmului trebuie astfel concepute încât să se termine într-un număr finit de pași.

Exemplu:

Pas 1: Atribuie 1 lui x;

Pas 2: Adaugă 2 la x;

Pas 3: Dacă x=10 atunci STOP;
altfel afișează x; se reia de la Pasul 2.

Finitudine

Pas 1: Atribuie 1 lui x;

X=1

Pas 2: Adaugă 2 la x;

X=3 X=5 X=7 X=9 X=11 …

Pas 3: Dacă x=10 atunci STOP;
altfel afișează x; se reia de la Pasul 2.

Afișează numere impare dar nu se oprește niciodată!

Finitudine

Generarea numerelor impare mai mici decât 10

Pas 1: Atribuie 1 lui x;

X=1

Pas 2: Adaugă 2 la x;

X=3 X=5 X=7 X=9

Pas 3: Dacă x<=10 atunci STOP;
altfel afișează x; se reia de la Pasul 2.

Proprietăţile algoritmului

Determinare

Algoritmul trebuie să prevadă modul de soluționare a tuturor situațiilor care pot apărea în rezolvarea problemei respective, fără ambiguităţi sau neclarităţi.

Exemplu:

Pas 1: Atribuie 1 lui x;

Pas 2: **Fie** incrementează x cu 1

Fie decrementează x cu 1

Pas 3: Dacă $x \in [1,10]$ atunci se reia de la Pasul 2; altfel Stop.

Determinare

Exemplu:

Pas 1: Atribuie 1 lui x;

Pas 2: Aruncă o monedă

Pas 3: Daca se obține aversul monedei
atunci incrementează x cu 1
altfel decrementează x cu 1

Pas 4: Dacă $x \in [1,10]$ atunci se reia de la Pasul 2; altfel Stop.

Proprietăţile algoritmului

Eficiență

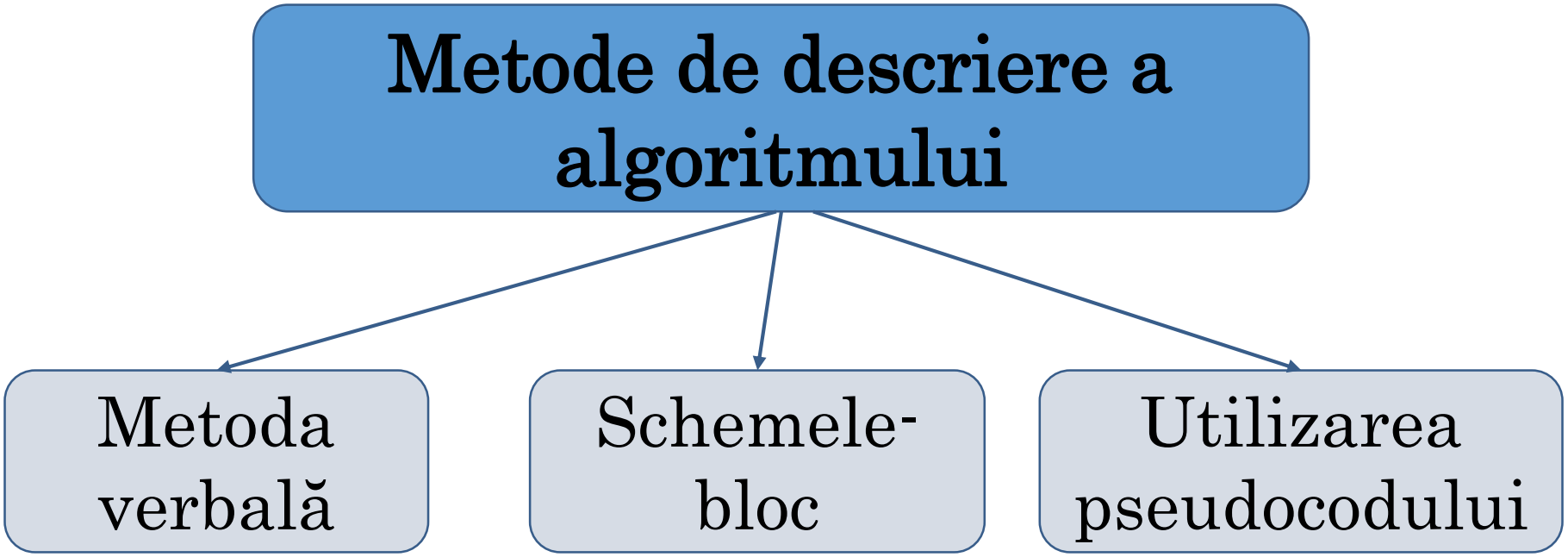
Operațiile algoritmului trebuie concepute astfel, încât durata lui de execuție să fie minimă.

Un algoritm trebuie să folosească un volum rezonabil de resurse de calcul: memorie și timp de calcul.

Elementele algoritmului

- **Setul de obiecte** ce formează totalitatea datelor iniţiale posibile, rezultatelor intermediare şi finale;
- Succesiunea de operaţii care realizează **introducerea** datelor iniţiale;
- Succesiunea de operaţii care realizează **prelucrarea** nemijlocită a datelor iniţiale în vederea obţinerii datelor finale;
- Succesiunea de operaţii care realizează **vizualizarea** (afişarea) datelor finale.

Metodele de descriere a algoritmilor



Metoda verbală

Descrierea verbală a algoritmului conține enumerarea pașilor algoritmului într-un limbaj uman.

Specificare a problemei: *De la tastatură se introduc două numere întregi A și B. Să se afişeze suma acestora.*

Algoritm:

Pas 1: Citirea de la tastatură a numărului **A**;

Pas 2: Citirea de la tastatură a numărului **B**;

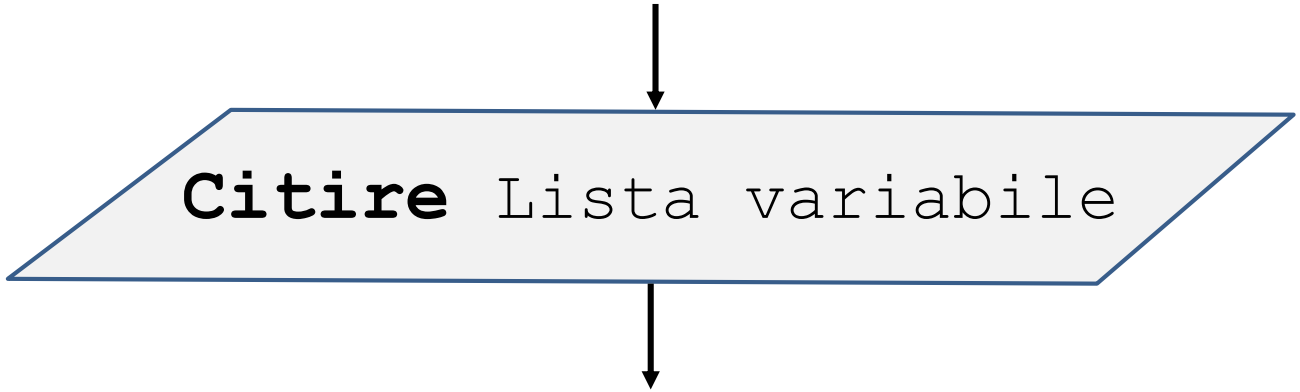
Pas 3: Calcularea rezultatului **S** – suma numerelor **A** și **B**;

Pas 4: Afișarea rezultatului **S**.

Schemele-bloc

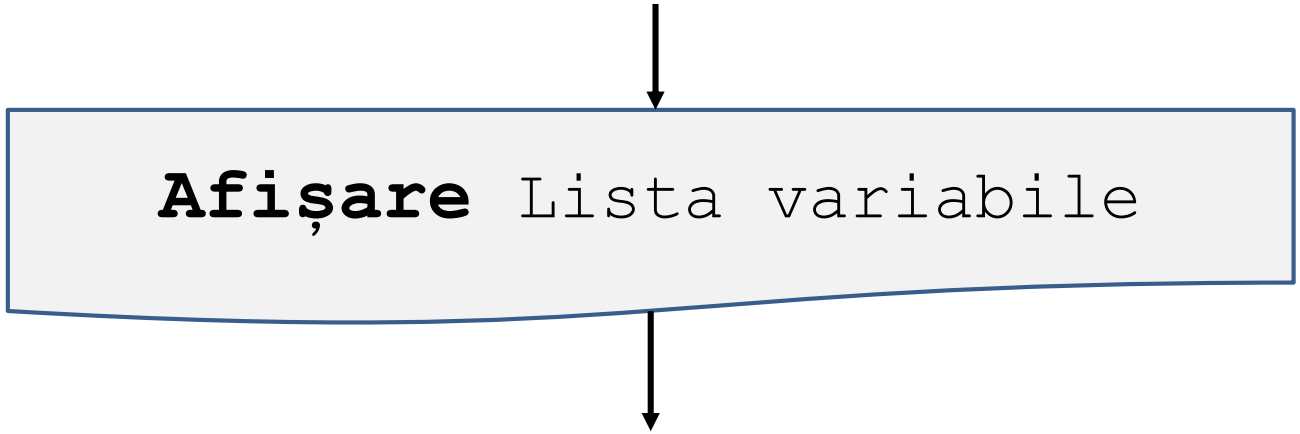
- Operațiile algoritmului se reprezintă cu ajutorul unor blocuri grafice.
- Schemele logice utilizează săgeţi de legătură între diferite forme geometrice care simbolizează acţiunile ce urmează a fi executate.
- Scheme logice = diagrame de blocuri.

Bloc pentru introducerea datelor (bloc de citire)



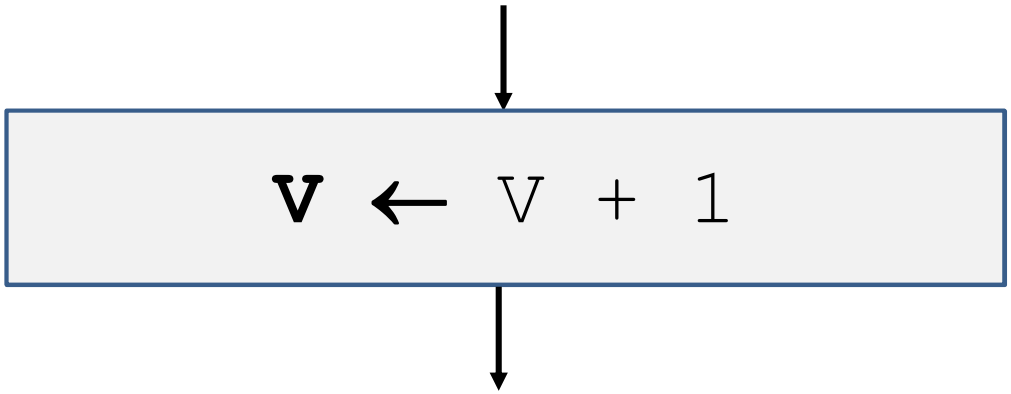
- Lista variabilelor cuprinde numele simbolice ale variabilelor cărora li se asociază valori citite.
- **Variabile** sunt zone de memorie care își schimbă valoarea și care se caracterizează printr-un nume unic.

Bloc de extragere a rezultatelor (bloc de scriere)



- Bloc de scriere (operaţia de ieşire) presupune afișarea valorilor unor variabile pe ecran în ordine specificată pe aceeaşi linie.

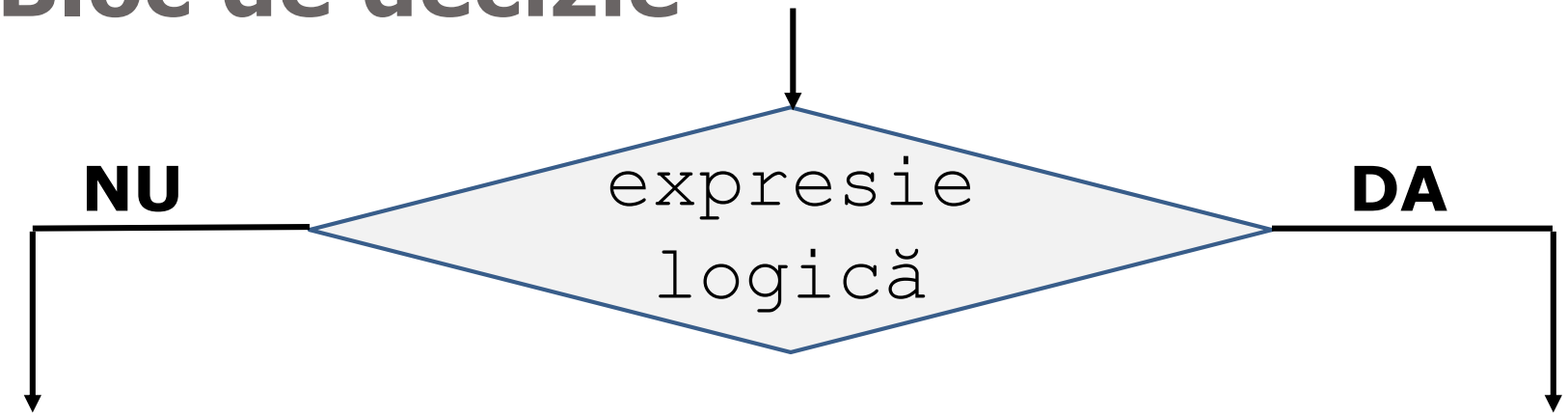
Bloc de calcul (bloc de atribuire)



Un astfel de bloc indică următoarea succesiune de operații:

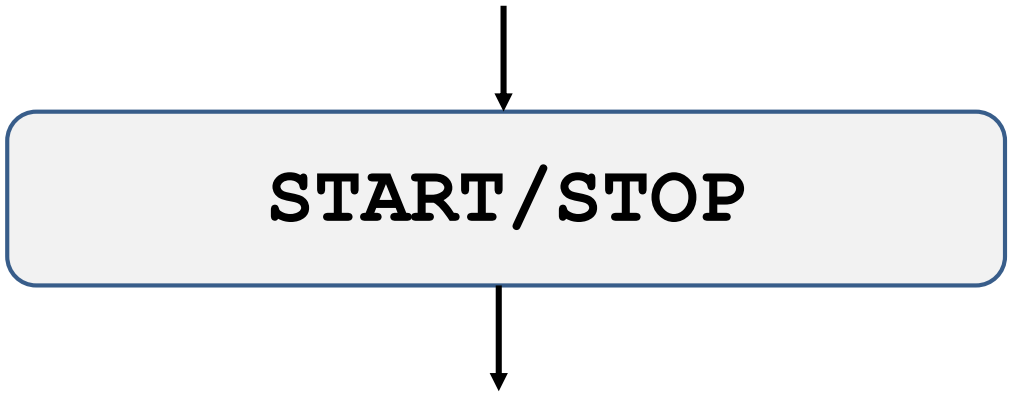
1. Se calculează expresia din partea dreaptă;
2. Se atribuie variabilei din partea stângă valoarea calculată anterior (**v** reprezintă numele variabilei).

Bloc de decizie



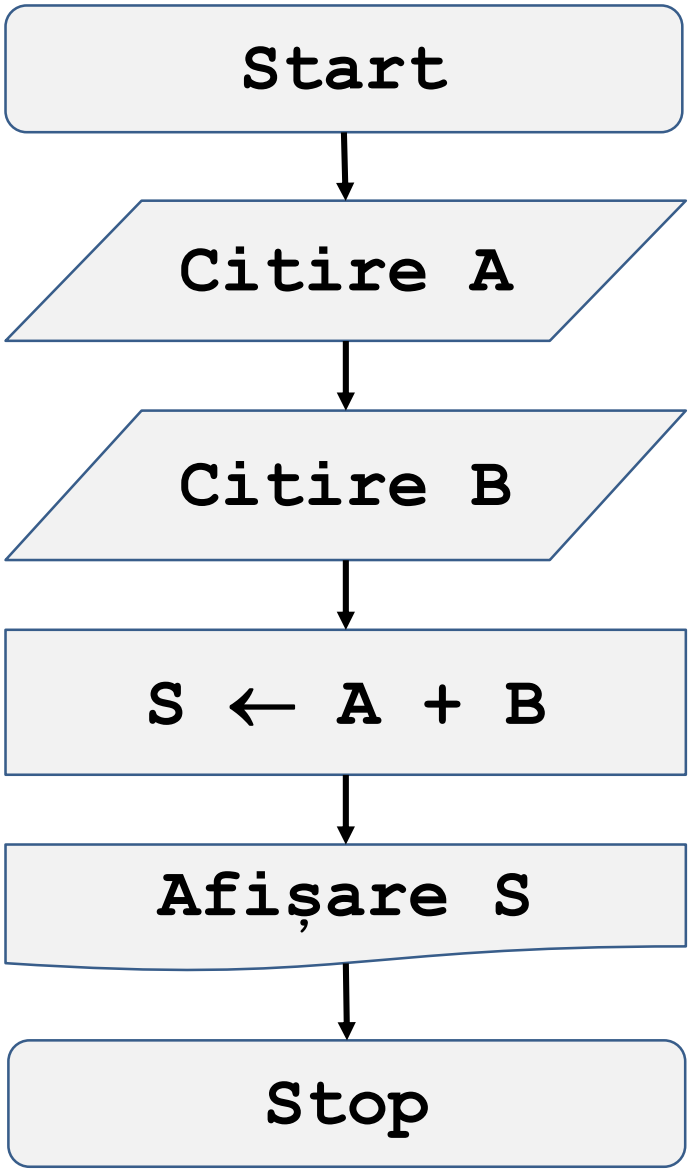
- expresie logică (condiția) poate avea valoarea “adevărat” sau “fals”.
- algoritmul se va executa după acest bloc **doar pe una** din ramurile sale: fie pe ramura cu DA sau fie pe ramura cu NU.
- decizia de a alege o ramură sau pe cealaltă se face pe baza evaluării **condiției** din bloc.

Bloc de început/sfârșit



- START – Indică începutul algoritmului.
- STOP – Indică sfârșitul algoritmului.
- sunt unice în cadrul unei scheme logice.

Schema-bloc a algoritmului



Utilizarea pseudocodului

Pseudocodul reprezintă o notație textuală ce permite exprimarea logicii programelor într-un mod formalizat fără a fi necesare reguli de sintaxă riguroase ca în limbajele de programare.

În calitate de pseudocod se utilizează construcţiile limbajului de programare în care va fi codificat algoritmul, fără respectarea strictă a regulilor de sintaxă.

Exemplu:

```
Citeşte A
Citeşte B
S←A+B
Scrie S
```

```
Read(A)
Read(B)
S:= A+B
Write(S)
```

Sarcini pentru lucrul independent

Fiind date trei numere naturale x, y și z, citite de la tastatură, să se determine și să se afișeze pe ecran media aritmetică a acestora.

Se cer:

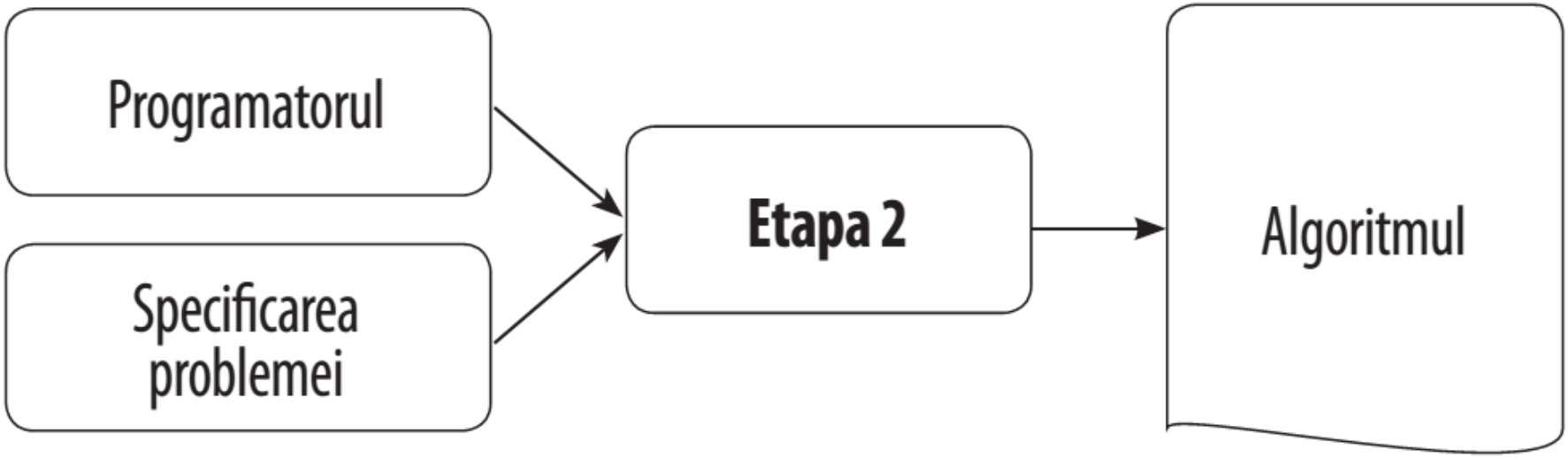
a) specificarea problemei;

b) algoritmul;

c) schema logică;

d) pseudocodul.

Schema etapei de proiectare a algoritmului



Codificarea algoritmului

- La această etapă algoritmul este transformat în program.
- **Limbajul de programare** reprezintă un sistem formal de notații, destinat descrierii algoritmilor într-o formă permisă de calculator.
- Limbajul de programare definește setul de **reguli lexicale**, **sintactice** și **semantice**, utilizate la scrierea unui program.

Codificarea algoritmului

Lexicul definește totalitatea cuvintelor unui limbaj şi semnificația acestor cuvinte.

```
Begin
End
Var
Procedure
...
```

Codificarea algoritmului

```
Program Exemplu;
Begin
 Writeln('********');
 Writeln('Hello world');
 Write('********');
End.
```

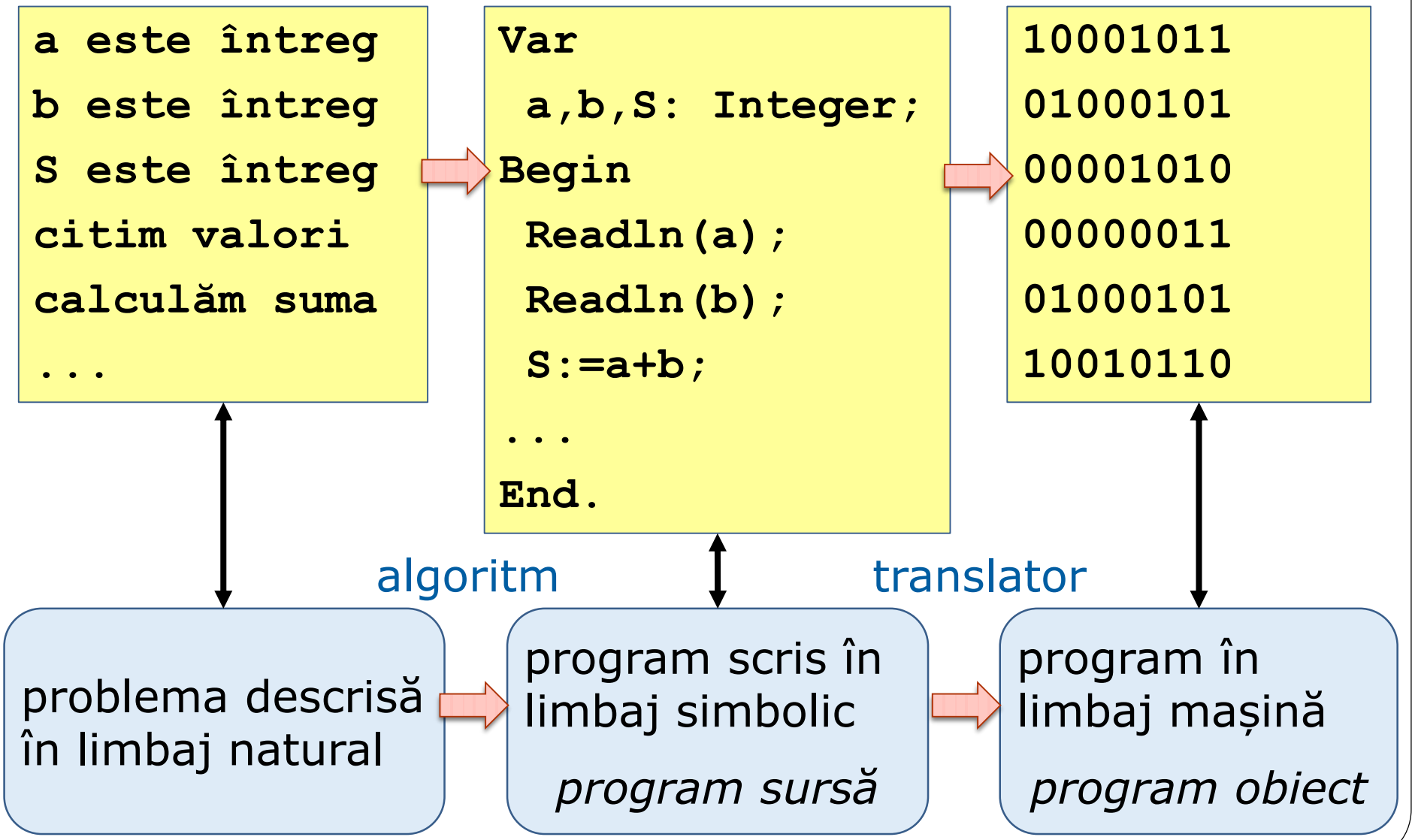
Sintaxa unui limbaj de programare reprezintă mulțimea regulilor care descriu modul de alcătuire al programului.

Semantică reprezintă mulțimea regulilor care definesc înțelesul fiecărui program.

Codificarea algoritmului

- O problemă de calcul este întotdeauna descrisă inițial de către programator într-un **limbaj natural** (apropiat de cel uman).
- Rezolvarea acesteia cu ajutorul calculatorului implică traducerea algoritmului într-un **limbaj simbolic** ce poate fi interpretat de sistemul de calcul.
- Programul scris în limbaj simbolic este în final tradus în **limbaj mașină** pentru a putea fi înțeles şi executat de sistemul de calcul.

Codificarea algoritmului

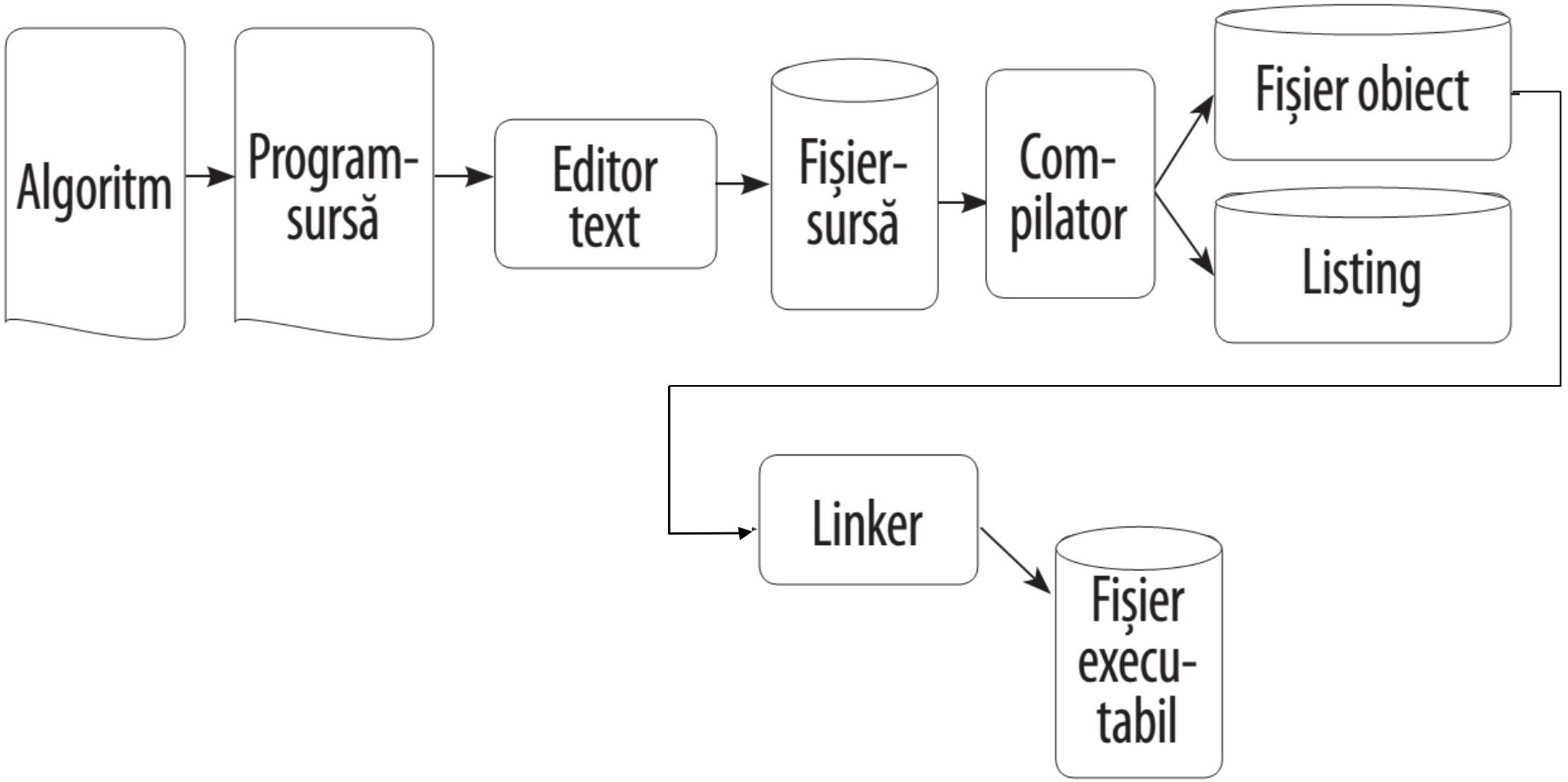


Translatoare

Program translator = este un program special care face conversia între limbajul simbolic şi limbajul mașină.

- **asamblor**: translatorul unui *limbaj de asamblare* în cod mașină, operația de traducere poartă numele de asamblare (mnemonica *ADD* indică operația de adunare),
- **compilator**: translatorul unui *program sursă* scris într-un limbaj de nivel înalt, în cod mașină sau într-un limbaj apropiat de acesta (limbajul de asamblare "assembler"),
- **interpretor**: translatorul unui *program sursă* scris într-un limbaj de nivel înalt într-un cod intermediar mai eficient care este executat imediat.

Tehnologia elaborării programelor executabile



Tipuri de erori în programe

Erorile sintactice apar la nerespectarea regulilor sintactice ale limbajului de programare în care este codificat algoritmul. Acest tip de erori este depistat de compilator. Toate erorile sintactice sunt descrise în listing.

Erorile de executare (run-time). Pot apărea la executarea programului în cazul apariției unor evenimente inadmisibile.

Erori logice. În aceste cazuri, programul se termină neavariat, dar rezultatele furnizate sunt incorecte.

Schema etapei de codificare a algoritmului

Exploatarea şi întreţinerea programului

Pot fi cazuri când programul nu corespunde specificării problemei și programatorul trebuie să refacă programul.

De asemenea, la cererea clientului, în program pot fi introduse funcționalități noi sau să fie modificate cele existente.

Sarcini pentru lucrul independent

1. Elaboraţi precizarea completă a problemei “Calcularea suprafeţei camerei”.
2. Elaboraţi precizarea completă a problemei “Determinarea apartenenţei unui număr la un interval dat”.
3. Elaboraţi specificarea problemei care va calcula cât trebuie de plătit pentru grădiniţă, dacă copilul a frecventat grădiniţa N_zile şi plata lunară este S lei.