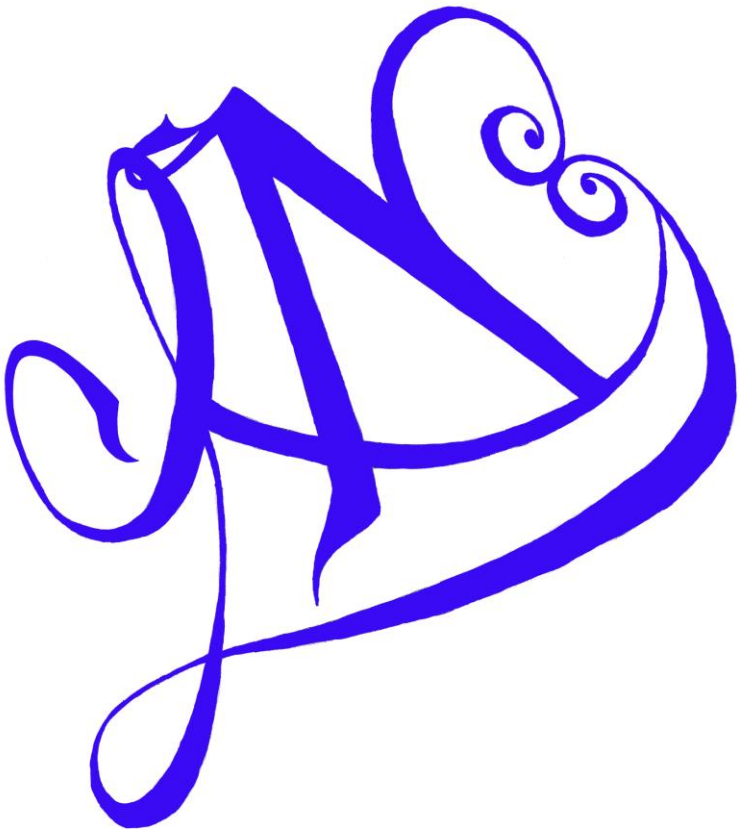
Conţinut



I. Structuri de bază în elaborarea algoritmilor.

Programarea structurată este un tip de programare care disciplinează realizarea algoritmilor reducând numărul structurilor de control utilizabile.

Structura este o modalitate de îmbinare a operaţiilor cu care lucrează algoritmii.

Unul din principiile de bază ale programării structurate rezultă din teorema de structură a lui *Böhm şi Jacopini*, care arată că orice algoritm se poate construi folosind doar trei tipuri de structuri fundamentale:

- liniară(secvenţială),
- alternativă (decizia), (simplă)
- repetitivă (iteraţia) (cu număr necunoscut de paşi condiţionată anterior).

II. Reprezentarea algoritmilor în pseudocod.

Pentru ca o secvenţă de operaţii să constituie un algoritm, ea trebuie să fie clară, adică la orice moment operaţia care urmează a fi executată trebuie să fie unic determinată, definită şi realizabilă.

Limbajul de tip pseudocod este un ansamblu de convenţii, respectate în mod sistematic, care permite folosirea unor instrucţiuni care conţin structurile de bază, uşurând astfel atât scrierea cât şi înţelegerea algoritmilor.

Prin program în limbaj de tip pseudocod înţelegem o succesiune de operaţii permise; acestea pot fi declaraţii sau instrucţiuni efective.

1. **Structura liniară (secvenţială)** cuprinde:

1.a. Declararea variabilelor:

Sintaxă: variabilă tip;

La începutul oricărui algoritm, vom preciza datele de intrare (in), datele de ieşire (out), eventualele date de manevră (dm), precum şi tipul acestora. Înainte de a utiliza orice variabilă, o vom declara, precizând numele şi tipul ei.

Exemplu:

x real; // am declarat o variabilă cu numele x de tip real

c caracter; // am declarat o variabilă cu numele c de tip caracter

i întreg; // am declarat o variabilă cu numele i de tip întreg

a,b,c real; // am declarat trei variabile cu numele a,b,c de tip real

1.b. Operaţia de citire:

Sintaxă: **Citeşte** v_1, v_2, ...,v_n;

Efect: se citesc n variabile; prin operaţia de citire se preiau succesiv valori de la tastatură/fişier şi se asociază, în ordine, variabilelor specificate.

1.c. Operaţia de scriere:

Sintaxă: **Scrie** e_1, e_2, ...,e_n;

Efect: se scriu expresiile respective; operaţia de scriere presupune evaluarea în ordine a expresiilor specificate şi afişarea pe ecran a valorilor lor pe aceeaşi linie.

1.d. Operaţia de atribuire:

Sintaxă: variabilă←expresie;

Efect: se evaluează expresia, apoi se atribuie valoarea expresiei variabilei din membrul stâng.

1.e. Instrucţiunea vidă:

Sintaxă: ;

Orice instrucţiune se termină cu caracterul ;.

1.f. Instrucţiunea compusă:

Sintaxă:

```
{
        instrucţiune_1;
        instrucţiune_2;
        ...
        instrucţiune_n;
}
```

Efect: se efectuează în ordine instrucţiunile specificate; este utilă când sintaxa permite executarea unei singure instrucţiuni, dar este necesară efectuarea mai multor operaţii.

2. **Structura alternativă (decizia)** cuprinde:

a. Instrucţiunea de selecţie simplă: cu format redus şi cu format complet.
b. Instrucţiune de selecţie multiplă.

2.a. Instrucţiunea de selecţie simplă, cu format redus:

Sintaxă:

Dacă (expresie)
    Atunci
        instrucţiune;

Efect: se evaluează (expresie); dacă valoarea expresiei este `adevărată`, atunci se execută instrucţiune; în caz contrar, se trece la următoarea linie de program.

2.a. Instrucţiunea de selecţie simplă, cu format complet:

Sintaxă:

Dacă (expresie)
    Atunci
        instrucţiune_1;
    Altfel
        instrucţiune_2;

Efect: se evaluează (expresie);

dacă valoarea expresiei este `adevărată`, atunci se execută instrucţiune_1;

dacă valoarea expresiei este `falsă`, atunci se execută instrucţiune_2.

2.b. Instrucţiunea de selecţie multiplă:

Sintaxă:

Alege E **dintre**
C_1: S_1
C_2: S_2
.....
C_n: S_n
rest: S_{n+1}

Efect: evaluarea expresiei E şi executarea secvenţei de instrucţiuni S_i sau S_{n+1} după cum E este egală cu constanta C_i sau nu.

3. Structura repetitivă (iteraţia) cuprinde:

a. Instrucţiunea repetitivă cu număr cunoscut de paşi.
b. Instrucţiunea repetitivă cu număr necunoscut de paşi:
- cu test iniţial (condiţionată anterior)
- cu test final (condiţionată posterior)

3.a. Instrucţiunea repetitivă cu număr cunoscut de paşi.

Sintaxă: **Pentru** contor←val_{init}, val_{fin} **execută**
instrucţiuni_S;

Efect: execuţia repetată a secvenţei de instrucţiuni_S pentru fiecare valoare a contorului cuprinsă între val_{init} şi val_{fin}; pentru ca secvenţa de instrucţiuni_S să se execută măcar o dată trebuie ca $val_{init} \leq val_{fin}$, în acest caz numărul de repetări este val_{fin}-val_{init}+1; valoarea variabilei contor se incrementează (se măreşte cu 1).

3.b. Instrucţiunea repetitivă cu număr necunoscut de paşi:
- *cu test iniţial (condiţionată anterior)*

Sintaxă: **Cât-timp** (expresie) **execută**
instrucţiune;

Efect:
p1: se evaluează expresia;
p2: dacă valoarea expresiei este `fals`, se iese din instrucţiunea cât-timp;
dacă valoarea espresiei este `adevărat`, se execută instrucţiunea, apoi se revine la p1.

3.b. Instrucţiunea repetitivă cu număr necunoscut de paşi:

- *cu test final (condiţionată posterior)*

Sintaxă:
Execută
instrucţiune;
cât-timp (expresie);

Efect:

p1: se execută instrucţiunea;

p2: se evaluează expresia;

p3: dacă valoarea expresiei este `fals` se iese din instrucţiunea repetitivă;
dacă valoarea expresiei este `adevărat`, se revine la p1.

III. Echivalenţa structurilor

Cât-timp (E**) execută** instr;	**Dacă (E)** **atunci** **Execută** instr**;** **cât-timp** (E);
Execută instr**;** **cât-timp** (E);	instr; **Cât-timp (**E**) execută** instr;
Pentru c←val_{init}, val_{fin} **execută** instr;	c ← val_{init}; **Cât-timp** c<=val_{fin} **execută** **{** instr; c ← c + 1; }
Pentru c←val_{init}, val_{fin} **execută** instr;	c ← val_{init}; **Dacă** c<=val_{fin} **atunci** **Execută** **{** instr; c ← c + 1; } **cât-timp** c<=val_{fin};