Dr. ing. Nedelcu Dorian

Programarea calculatoarelor în Turbo Pascal

- 2002 -

Referenţi ştiinţifici:

Prof. dr. ing. Ioan Jurca, Universitatea POLITEHNICA TIMIŞOARA
Dr. ing. Cornelia Anghel, Universitatea EFTIMIE MURGU Reşiţa
Dr. ing. Berinde Florin, Universitatea EFTIMIE MURGU Reşiţa

Tehnoredactarea computerizată: **Nedelcu Dorian**
Grafică copertă: **Diana SZEBEBYI**

Descrierea CIP a Bibliotecii Naţionale a României
NEDELCU DORIAN
Programarea calculatoarelor în TurboPascal/
Dorian Nedelcu – Timişoara : Eurostampa, 2002
210 pag. 25 cm
I.S.B.N.: 973-687-013-8

004.43 TURBO PASCAL

Editura EUROSTAMPA
Timişoara, B-dul Revoluţiei din 1989, nr. 26
Tel./Fax.: 056-20.48.16

Tiparul executat la ***Tipografia EUROSTAMPA***
Mai 2002

PREFAȚĂ

Lucrarea "*Programarea calculatoarelor in Pascal*" se adresează în primul rând studenților din anul I care au în planul de învățământ o disciplină de introducere în programare şi are un conținut adecvat acestui scop.

În primul capitol al lucrării sunt prezentate conceptual arhitectura şi structura internă a unui sistem de calcul, accentul fiind pus pe funcționalitatea acestuia şi nu pe descrierea tehnică detaliată. În acest fel, cititorul neavizat ia cunoştință de principalele componente ale unui calculator şi de modalitățile de interacțiune cu acesta. Capitolul se finalizează cu noțiuni generale despre limbajele de programare.

Al doilea capitol este rezervat algoritmilor. Sunt expuse caracteristicilor acestora şi prezentate modalitățile de reprezentare a algoritmilor sub forma schemelor logice şi prin limbaj pseudocod, moduri de reprezentare utilizate în lucrare. Pentru fiecare dintre acestea, sunt detaliate instrucțiuni şi restricții specifice. Capitolul continuă cu prezentarea elementelor programării structurate: secvența, decizia, repetiția. După expunerea noțiunilor teoretice, capitolul include un set de exemple de algoritmi reprezentativi, materializați prin prezentarea comparativă a schemelor logice cu a formatului pseudocod, însoțite de descrieri teoretice ce fundamentează explicativ noțiunile prezentate; în baza unor exemple sugestive sunt abordați algoritmii de sumare, calcul produs şi contorizare, algoritmi de bază în domeniul programării.

Capitolul trei oferă o apropiere descriptivă de mediul de programare TurboPascal. Sunt expuse, prin limbaj şi imagini, elemente ale interfeței TurboPascal: ferestre de lucru şi de dialog, taste de apel, editorul de texte, precum şi opțiunile meniului TurboPascal, detaliind funcțiunile acestora.

Capitolul patru defineşte conceptele fundamentale ale limbajului TurboPascal: vocabular, notații, tipuri de date, structura generală a unui program în format TurboPascal şi se finalizează cu o enumerare a etapelor ce trebuie parcurse la rezolvarea unei probleme utilizând acest limbaj.

Capitolul cinci este rezervat instrucțiunilor de bază ale limbajului TurboPascal: declararea variabilelor şi constantelor, instrucțiunea de atribuire, instrucțiunea compusă, instrucțiunea vidă, precum şi cu instrucțiuni de intrare-ieşire. Aceste instrucțiuni sunt exemplificate printr-un pachet de patru exemple semnificative, amplu comentate prin figuri, valori numerice, explicații teoretice şi simboluri explicative.

Instrucțiunile decizionale constituie subiectul celui de-al şaselea capitol, substanța acestuia fiind concretizată printr-un grup de nouă aplicații specifice, care includ printre altele: ecuația de grad I şi II, minim dintre două respectiv patru numere, sistem de două ecuații cu două necunoscute, acestea făcând apel la instrucțiunile de decizie IF–THEN-ELSE şi/sau selecția multiplă prin CASE. Pentru multe dintre aceste exemple sunt prezentate rezultate ale execuției pe variante ale datelor de intrare, materializând astfel numeric efectul ramificațiilor instrucțiunilor decizionale.

Capitolul şapte este dedicat structurilor repetitive WHILE, REPEAT şi FOR. După sistematizarea noțiunilor şi explicarea acestora din punct de vedere sintactic, pachetul format din patrusprezece aplicații, concretizate prin programe TurboPascal, fundamentează practic utilizarea instrucțiunilor repetitive. Sunt incluse exemple de algoritmi descrişi în capitolul unu prin scheme logice şi limbaj pseudocod, pentru care prezentarea comparativă a programului

permite evidențierea mijloacelor specifice de exprimare ale limbajului TurboPascal. Capitolul se încheie prin prezentarea procedurilor asociate ciclurilor BREAK şi CONTINUE.

Instrucțiunea de salt necondiționat GOTO şi generarea numerelor aleatoare, sunt subiectele rezervate capitolului opt, acestea fiind însoțite de exemple edificatoare de utilizare.

Tipurile de date utilizator (enumerare, interval, structurate) sunt abordate în capitolul nouă, acestea fiind exemplificate printr-un set de aplicații specifice, care includ manipularea vectorilor şi matricilor, prelucrări frecvent utilizate în limbajul TurboPascal. Pentru fiecare exemplu, accentul este pus pe explicarea modalităților limbajului de accesare a componentelor acestor tipuri de date, modalitățile grafice utilizate în acest sens constituind un suport vizual ajutător.

Capitolul zece se opreşte asupra şirurilor de caractere şi asupra disponibilităților limbajului de accesare, prelucrare şi utilizare a acestor tipuri de date, exemplificarea fiind concretizată prin intermediul a patru aplicații TurboPascal.

Ultimul capitol al lucrării este dedicat subprogramelor, prin descrierea noțiunile conceptuale asociate acestora. Sunt abordate probleme specifice procedurilor şi funcțiilor TurboPascal: domenii de vizibilitate, comunicarea între programul apelant şi subprograme, declarația anticipată, tipuri procedurale. Exemplele analizate scot în evidență mecanismele de operare specifice subprogramelor TurboPascal, insistând asupra modului de interacțiune cu programele apelante.

Lucrarea se încheie cu enumerarea bibliografică şi cu anexe utile cititorului.

Pe parcursul a peste două sute de pagini autorul introduce cititorul în domeniul programării calculatoarelor, bazat pe limbajul de programare TurboPascal, metoda principală utilizată fiind expunerea noțiunilor teoretice urmate de exemplificarea comentată a acestora prin programe sugestive, uşor de asimilat. În interiorul lucrării, sunt utilizate mijloace vizuale de o deosebită expresivitate pentru evidențierea noțiunilor, structurilor şi tehnicilor specifice limbajului TurboPascal. Asocierea fiecărui exemplu cu rezultate numerice concrete, uneori chiar şi pentru mai multe variante de execuție, constituie o altă modalitate inedită de concretizare a limbajului, de o veritabilă utilitate în învățarea acestuia.

Prin maniera de expunere şi conținutul informațional de o certă accesibilitate, lucrarea poate fi utilizată în aceeaşi măsură de studenți, elevi şi cadre didactice ca un instrument de lucru în asimilarea cunoştințelor de programare bazate pe utilizarea limbajului de programare TurboPascal, motiv pentru care o recomandăm cu căldură celor interesați.

Prof. Dr. Ing. Ioan Jurca
Universitatea “Politehnica” Timişoara

Structurată pe 11 capitole, cu un bogat arsenal de titluri bibliografice de referinţă în domeniu, precum şi 3 anexe care concretizează şi sintetizează noţiunile prezentate, lucrarea are un puternic caracter didactic, conţinând toate elementele pentru studiu individual, fiind de un real folos studenţilor din anul I de facultate, precum şi începătorilor în domeniul programării calculatoarelor, dar poate fi utilizată în egală măsură şi de cei mai avansaţi datorită gradului înalt de accesibilitate.

Parcurgerea lucrării este uşoară, nivelul asimilării de cunoştinţe crescând gradual de la noţiunile de bază şi componenţa calculatorului, concepte de bază ale algoritmilor, prezentarea mediului de programare, până la instrucţiuni de program şi exemple imediate în programe specifice, prezentate comparativ prin scheme logice, limbaj pseudocod şi program TurboPascal.

Tehnoredactarea la un înalt nivel, exemplele prezentate sunt însoţite de exemple grafice, explicaţii detaliate suplimentare, precizări importante, constituind un ajutor preţios în acumularea, înţelegerea, verificarea şi materializarea rezultatelor algoritmilor şi/sau programelor.

Dr. Ing. Cornelia Anghel
Univ. EFTIMIE MURGU Reşiţa

Manualul se adresează în primul rând unei categorii de persoane care nu au experienţă în lucrul cu calculatorul, fiind un curs de iniţiere. Este structurat pe 11 capitole şi 3 anexe. Studenţilor li se pune la dispoziţie un manual accesibil, concis, utilizabil şi pentru studiul individual, cu multe exemple bine documentate. Lucrarea este gândită pentru a dezvolta deprinderi de raţionament algoritmic.

În Capitolul 1, intitulat „SISTEME DE CALCUL”, se prezintă un scurt istoric al calculatorului, arhitectura generală a unui sistem de calcul, structura internă a unui calculator, evidenţiindu-se rolul funcţional al principalelor sale componente. Se evidenţiază, de asemenea, şi rolul limbajelor de programare şi se face o clasificare a acestora. În Capitolul 2, intitulat „ALGORITMI” se prezintă noţiunea de algoritm, modul de exprimare a algoritmilor în limbaj natural, moduri de reprezentare a algoritmilor prin scheme logice şi pseudocod, elementele programării structurate şi se încheie cu numeroase exemple comentate. În Capitolul 3, intitulat „MEDIUL DE PROGRAMARE Turbo Pascal” se prezintă în amănunt elementele de comandă ale sistemului de programe care constituie „Mediul de programare Turbo Pascal”. În capitolele următoare se prezintă în mod sistematic, simplu, cu multe exemple comentate, elementele fundamentale ale limbajului Turbo Pascal.

Conţinutul este puternic marcat de aspectul său educativ, fiind atractiv şi incitant. Este nu numai un manual bun, dar şi o lectură tehnică plăcută. Lucrarea este scrisă folosind o tehnoredactare deosebită, care poate sta ca model, din punctul de vedere al tehnicilor folosite şi pentru alţi autori de manuale, în general. Se remarcă multitudinea şi calitatea sugestivă a figurilor folosite. Limbajul este accesibil unui om neinstruit în domeniul tehnicii de calcul. Complexitatea cunoştinţelor prezentate creşte gradat. Se prezintă soluţii alternative la unele probleme, în acest fel, folosindu-se din plin procedeul pedagogic al învăţării prin comparaţie. Manualul, aşa cum este scris, oglindeşte o muncă susţinută şi plină de pasiune depusă de autor, pentru elaborarea lui.

Dr. Ing. Berinde Florin
Univ. EFTIMIE MURGU Reşiţa

CUVÂNT INTRODUCTIV AL AUTORULUI

Lucrarea se adresează celor care doresc să asimileze limbajul de programare Turbo Pascal, punând la dispoziția acestora fundamentele limbajului, concretizate prin noțiuni teoretice însoțite de aplicații amplu comentate.

Concepția lucrării este axată pe următoarele elemente principiale:

- ***accesibilitate** – lucrarea se adresează cu precădere începătorilor în însuşirea cunoştinţelor de programare, fiind concepută astfel încât să ofere un instrument de lucru eficient şi accesibil ca nivel de cunoştinţe; această caracteristică nu este însă strict restrictivă, deoarece lucrarea poate constitui un ajutor şi celor cunoscători şi/sau avansaţi, prin sistematizarea noţiunilor şi repertoriul exemplelor comentate;*
- ***ierarhizarea nivelului informațional** – pe parcursul lucrării noțiunile sunt prezentate gradual din punct de vedere al complexității, urmând o cale logică ce trebuie parcursă la apropierea de acest domeniu: noțiuni principiale despre calculator, asimilarea conceptelor de bază ale algoritmilor, prezentarea mediului de programare, detalierea instrucțiunilor în paralel cu exemplificarea acestora prin algoritmi şi programe specifice;*
- ***exemplificarea graduală** – considerând exemplul ca fiind metoda cea mai eficientă de asimilare, fiecare noțiune nou expusă este concretizată prin exemple şi aplicații, a căror complexitate creste gradat, fără a depăşi însă o barieră normală de accesibilitate; primele exemple sunt prezentate comparativ prin schemă logica, limbaj pseudocod şi program TurboPascal, urmând ca apoi sa fie prezentat numai programul TurboPascal; de asemenea, sunt exemple şi aplicații reluate în mai multe variante, permițând astfel accentuarea comparativă a tehnicilor de programare; Anexa 1 oferă memoratorul exemplelor din lucrare;*
- ***documentarea exemplelor** – fiecare exemplu este însoțit de ample şi detaliate explicații teoretice suplimentare cu rol de ajuta cititorul în însuşirea noțiunilor exemplificate şi formarea unei logici orientată spre programare; nivelul de detaliere descreşte gradat de la început spre final, evitând reluarea aceloraşi idei şi luând în considerare acumularea de cunoştinţe realizată prin parcurgerea materialului; de asemenea, exemplele şi aplicațiile sunt însoțite de rezultate numerice concrete, constituind un prețios ajutor la înțelegerea, verificarea şi materializarea rezultatelor unui algoritm şi/sau program; indicatori grafici atenționează vizual asocierea instrucțiunilor cu rezultatul numeric şi evidențiază diferitele tipuri de structuri prin simboluri specifice;*
- ***conținut didactic** – lucrarea are un puternic caracter didactic, conținând toate elementele necesare unei parcurgeri prin studiu individual, chiar si in lipsa calculatorului; în acest sens s-a apelat la exemple didactice şi reprezentative, ce pot fi înțelese şi asimilate uşor şi sunt accesibile ca logică.*

Lucrarea se adresează studenților din anul I care parcurg pe durata unui semestru materia "Programarea calculatoarelor", dar poate fi utilizată în egală măsură şi de elevii de nivel preuniversitar. Lucrarea nu îşi propune să epuizeze întreg domeniul limbajului Turbo Pascal, ci se doreşte a fi un instrument călăuzitor celor care doresc să se inițieze în acest domeniu.

Reşița, *Dr. ing. Dorian Nedelcu*
Aprilie, 2002

CUPRINS

1. SISTEME DE CALCUL

1.1 SCURT ISTORIC

Timp de 2500 de ani oamenii au utilizat diverse dispozitive care să-i ajute în calcule, cel mai răspândit fiind abacul. Încă din sec. XVII-lea sistemul binar a atras atenția marelui matematician George Friederich Leibnitz, care a sesizat principalul avantaj: având numai două cifre 0 şi 1, acest sistem se pretează cel mai bine la descrierea fenomenelor cu două stări posibile: DA şi NU, EXISTENȚĂ şi INEXISTENȚĂ, ALEGERE sau RESPINGERE. Un astfel de sistem asigură o mare simplitate în calcule. Dificultatea a constat în a găsi un sistem fizic care să poată simula un astfel de calcul şi care să permită efectuarea rapidă de calcule complicate.

În anul 1642 matematicianul şi filozoful Blaise Pascal pune la punct o maşină mecanică de calculat, dar în sistem zecimal, care funcționează cu angrenaje cu roți dințate şi cu viteză de calcul foarte mică. În anul 1671 matematicianul german Gottfried Wilhelm von Leibnitz inventează prima maşină de calcul pentru operația de înmulțire.

În fața oamenilor de știință au apărut probleme care implică volume mari de calcul. Astfel, spre sfârşitul secolului trecut calculele pentru ridicarea hărții topografice a Indiei au necesitat munca unui colectiv numeros vreme de 2 ani de zile.

În anul 1833 englezul profesorul Charles Babbage la Cambridge University a imaginat o maşină de calcul a cărei concepție se regăseşte în calculatoarele de azi, o uriaşă maşină analitică, care funcționa pe principiul roților dințate, includea un sistem de intrare cu cartele perforate, memorie, o secțiune de calcul şi un dispozitiv de ieşire. Maşina era controlată de program, Babbage fiind ajutat în această privință de Ada Byron, fiica lordului Byron. Tehnologia mecanică nu a putut asigura condițiile de realizarea acestei maşini.

La sfârşitul secolului al XIX-lea Marconi construia primul aparat de radio, care conținea un element de bază: TUBUL ELECTRONIC, prin care se putea realiza CIRCUITUL BISTABIL, care are numai două stări posibile: ÎNCHIS şi DESCHIS. La primirea unui impuls acest circuit trece dintr-o stare în alta, constituind modalitatea de a simula sistemul de numerație binar. Astfel se pot crea circuite care să reproducă operații aritmetice cu 2 numere.

În anul 1954, o echipă americană a creat primul calculator electronic de uz general, care se numea ENIAC (Electronic Numerical Integrator And Calculator), cântărea cca. 30 tone şi conținea 18.000 tuburi, din care trebuiau înlocuite în medie 10/oră. Performanța acestuia era de 300 înmulțiri pe secundă şi putea memora 20 de numere.

Matematicianul John Von Neumann a introdus ideea codificării informațiilor în sistemul binar, precum şi a memorării atât a datelor, cât şi a secvenței de instrucțiuni care controlează operațiile efectuate de calculator şi a definit structura de bază a unui calculator, valabilă şi astăzi, prin care un calculator trebuie să conțină următoarea componență:

- dispozitive de intrare - care să preia operanzi şi instrucțiuni;
- memoria – din care se preiau operanzi şi instrucțiuni şi se depun rezultatele;
- secțiune de calcul – care execută operații aritmetice şi logice asupra operanzilor citiți din memorie;
- dispozitive de ieşire – care permit transferul rezultatelor spre un suport fizic;
- unitate de comandă – capabilă de interpretarea instrucțiunilor/comanda execuției acestora.

Tabelul 1.1 prezintă evoluția calculatoarelor electronice:

Tabel 1.1

Generația	Ani apariție	Tehnologia de bază	Număr instrucțiuni/sec.	Durata de funcționare
I	50	Tub electronic	1.000	Ore
II	60	Tranzistoare	100.000	Sute de ore
III	70	Circuite integrate	10.000.000	Mii de ore
IV	80	Microprocesoare	100.000.000	Ani
V	90	Inteligență artificială	Mult superioară	Mult superioară

1.2 ARHITECTURA GENERALĂ A UNUI SISTEM DE CALCUL

Calculatorul *reprezintă un sistem fizic de echipamente care prelucrează, pe baza unui program, datele introduse într-o formă prestabilită şi furnizează rezultatele într-o formă accesibilă utilizatorului*, fig. 1.1.

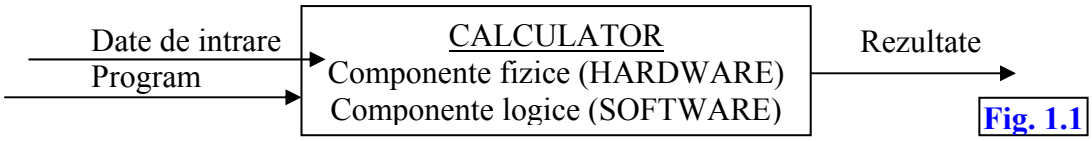


Fig. 1.1

Organizarea de bază a unui calculator (componența hardware) este dată în fig. 1.2.

Fig. 1.2

Unitatea aritmetică şi logică (UAL) execută toate calculele, comparațiile şi luări de decizii. Datele sunt transferate din memorie pentru prelucrare în UAL. Nici o prelucrare nu se face în memorie.

Unitatea de comandă şi control (UCC) acționează ca un sistem nervos central pentru celelalte componente ale calculatorului. Fără a executa prelucrare propriu-zisă a datelor, UCC dirijează funcționarea întregului sistem.

Unitatea de memorie (UM) se împarte în:

a) *memorie internă (MI) sau principală*

- memorează datele de prelucrat (date de intrare), programele de prelucrare (în curs de execuție), rezultatele prelucrării (date de ieşire); există 4 zone ale MI: zona datelor de intrare, zona de lucru, zona datelor de ieşire, zona programului;
- poate fi de două feluri: **ROM** (Read Only Memory) – poate fi doar citită, nu şi scrisă, memorează programe din fabricația calculatorului fără posibilitatea de ştergere sau modificarea acestora; **RAM** (Random Acces Memory) – memorie de tip volatil, în care sunt memorate datele şi programele în curs de execuție, permițând citirea,

scrierea sau ştergerea informaţiilor; în momentul opririi calculatorului conţinutul memoriei RAM se pierde;

b) *memorie externă (ME) sau auxiliară* – permite memorarea datelor pe o perioadă nedeterminată de timp; poate fi conectată direct la UCP (hard disk, unităţi de disc flexibil, CD-ROM) sau în afara acesteia (hârtie, benzi magnetice, etc.).

În momentul prelucrării, datele şi programele sunt transferate în memoria internă.

Echipamentele de intrare/ieşire se mai numesc şi e**chipamente periferice sau periferice**, deoarece, deşi nu fac parte din UCP sunt localizate în apropierea ei. Ansamblul de periferice conectate la unitatea centrală reprezintă **configuraţia unui calculator**.

Toate unităţile funcţionale se concretizează din punct de vedere tehnic în ECHIPAMENTE, care, interconectate între ele constituie, *arhitectura unui sistem de calcul*.

Echipamentele de intrare sunt destinate:

- introducerii datelor de intrare într-o formă specifică utilizatorului;
- convertirea acestora într-o formă specifică maşinii;
- transmiterea datelor UCP.

Exemple de echipamente de intrare: tastatura, mouse, joystick, scanner, etc.

Echipamentele de ieşire sunt destinate:

- să preia rezultatele prelucrărilor de la UCP în cod maşină;
- să convertească aceste rezultate într-o formă accesibilă oamenilor.

Exemple de echipamente de ieşire: monitor, imprimantă, plotter, etc.

1.3 STRUCTURA INTERNĂ A UNUI CALCULATOR

Evoluţia calculatoarelor este marcată de inventarea în anii 70 a microprocesorului, care oferă posibilitatea concentrării într-un singur circuit integrat a funcţiilor propuse de Neumann (& 1.1). Structura standard a unui calculator PC este dată în fig. 1.3.

Calculatoarele PC sunt formate din module specializate în execuţia anumitor operaţii, care se interconectează prin magistrala de sistem BUS. Interfaţa BUS este un dispozitiv care serveşte la interconectarea între UCP (microprocesor) şi periferia sistemului.

1.3.1 Microprocesorul

În interiorul său, microprocesorul conţine nişte zone sau locaţii în care se pot memora date de lungimi foarte mici, locaţii care poartă numele de regiştrii, fiecare având un nume special. Între aceştia, există unul special, denumit IP (INSTRUCTION PROGRAM) utilizat pentru memorarea instrucţiunilor programului de executat.

Din punct de vedere al performanţelor, un microprocesor este caracterizat de următorii parametrii: viteza de lucru, capacitatea de memorie pe care o poate adresa, setul de instrucţiuni pe care le poate executa.

a) **Viteza de lucru** – este determinată de mai mulţi factori:

1. *frecvenţa ceasului intern* – ceasul intern al unui PC este un oscilator care trimite în calculator pulsuri la intervale de timp bine determinate; toate activităţile calculatorului sunt coordonate de aceste impulsuri periodice, deci şi funcţionarea microprocesorului. Frecvenţa cu care sunt generate aceste pulsuri poartă numele de *frecvenţa ceasului intern*,

cu unitate de măsură Hz şi multiplii acestuia (1KHz=1000 Hz, 1 MHz=1.000.000 Hz); de exemplu o frecvenței de 16 MHz înseamnă 16 milioane de pulsuri pe secundă.

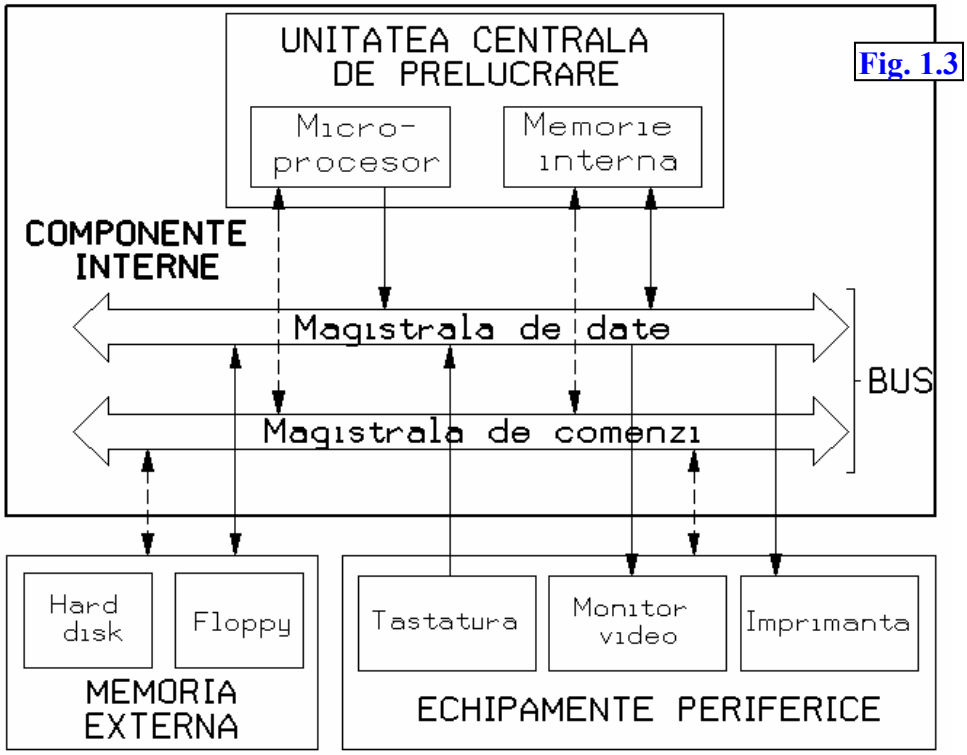


Fig. 1.3

2. *dimensiunea regiştrilor şi a magistralei de date* – acestea sunt exprimate în biți şi multiplii ai acestuia (puteri ale lui 2):

1 byte (octet) = 2^3 biți = 8 biți
1 word (cuvânt) = 2^5 biți = 32 biți
1 double word (dublu cuvânt) = 2^6 biți = 64 biți
1 Kilo Bytes (KB) = 2^{10} bytes = 1024 bytes
1 Mega Bytes (MB) = 2^{20} bytes = 1024 KB
1 Giga Bytes (GB) = 2^{30} bytes = 1024 MB
1 Terra Bytes (TB) = 2^{40} bytes = 1024 GB

Dimensiunea uzuală a regiştrilor interni sunt de 8, 16, 32 biți, mărimile mari generând viteze de lucru mai mari, deoarece cu cât dimensiunea regiştrilor este mai mare cu atât numărul de operații de transfer cu memoria internă este mai mic. Dimensiunea magistralei de date este de importanță, în sensul că cu cât aceasta este mai mare cu atât fluxul de date ce circulă pe magistrală este mai mare.

3. *tipul de procesor* – tipuri de procesoare ale firmei Intel: 8086, 80286, 80386, 80486, Pentium, Pentium PRO, Pentium MMX, Pentium II, Pentium III, Pentium IV. Alte firme care fabrică procesoare: AMD, Cyrix şi IBM.
4. *dimensiunea memoriei cache* - acest tip de memorie este concepută pentru a fi legată mai direct de procesor decât MI, evitându-se astfel operațiile intermediare.

b) Capacitatea memoriei pe care o poate adresa

Un procesor nu poate adresa o cantitate infinită de memorie, ci numai cea maximă impusă de construcția acestuia. Această caracteristică este importantă deoarece:

- procesorul lucrează mai repede cu memoria internă (MI) decât cu cea externă (ME) (discuri magnetice);

- un program nu poate fi executat decât dacă se află încărcat în MI; deci un program complex care ocupă multă MI, nu poate fi pornit dacă MI nu este suficient de mare.

c) **Setul de instrucţiuni pe care le poate executa**

În general setul de instrucţiuni este legat direct de tipul microprocesorului. Numărul de instrucţiuni din set creşte pe măsura evoluţiei procesoarelor.

1.3.2 Memoria internă

Memoria internă este o parte fiabilă a calculatorului, fiind testată la fiecare pornire a acestuia. Orice eroare detectată la testul de memorie provoacă oprirea sau în timpul lucrului provoacă oprirea calculatorului. Forma acesteia este asemănătoare cu circuitele integrate şi este alcătuită din părţi de dimensiune egală, denumite **locaţii de memorie**. Locaţiilor de memorie li se atribuie un număr de ordine, începând cu valoarea zero (0,1,2,3,…), numere care se numesc **adrese de memorie**. Accesul la locaţiile de memorie se realizează prin intermediul adreselor. Locaţiile de memorie au o anumită lungime, exprimată în biţi. Cea mai mică locaţie este **octet**-ul sau **byte**-ul (1 byte=8 biţi), fiind folosit la memorarea unui caracter. Pentru a avea acces şi la alte date decât caractere, se utilizează locaţii formate din succesiuni grupate în octeţi:

- locaţii de lungime fixă:
  - semicuvânt = 2 octeţi = 16 biţi
  - cuvântul = 4 octeţi = 32 biţi
  - dublu-cuvântul = 8 octeţi = 64 biţi.
- locaţii de memorie de lungime variabilă (denumite **succesiuni de caractere**).

Memoria internă a unui PC este caracterizată de 2 parametrii:

- **dimensiunea MI** – este legată de procesorul folosit şi este limitată de tipul acestuia;
- **timpul maxim de răspuns** – se exprimă prin intervalul de timp scurs din momentul în care MI primeşte comanda de la microprocesor pentru a citi sau scrie date şi momentul în care le depune pe magistrala de date.

În varianta clasică, memoria este împărţită în următoarele zone:

- **memoria de bază** ce cuprinde 640 KO şi este la dispoziţia programatorului
- **memoria rezervată** ce cuprinde 384 Ko şi este la dispoziţia sistemului de operare SO (in care se încarcă partea rezidentă a SO şi programe de comandă a diferitelor echipamente periferice)
- **memoria extinsă** – memoria peste 1 MB.

1.3.3 Echipamente periferice

Toate echipamentele diferite de procesor sau de MI se numesc **periferice** şi se pot clasifica în:

- **echipamente periferice de intrare** – permit introducerea de date, programe sau comenzi (tastatura, mouse, scanner, etc.);
- **echipamente periferice de ieşire** – oferă posibilitatea de vizualizare a rezultatelor prelucrărilor efectuate de calculator şi transmiterea acestora altor calculatoare (monitorul video, imprimanta, plotter, proiectoare multimedia);

- **echipamente periferice de memorare (memoria externă)** – dispozitive capabile să transmită şi să recepționeze date (hard disc, floppy disk, compact discuri, bandă magnetică).

1.3.3.1 Tastatura

Tastatura reprezintă perifericul standard al oricărui PC, apăsarea fiecărei taste având ca efect transmiterea în interiorul calculatorului a unui cod diferit. Modelele uzuale sunt cu 101, 102 taste sau 104 taste pentru tastatură de tipWindows. Dispoziția tastelor este prezentată în fig. 1.4.

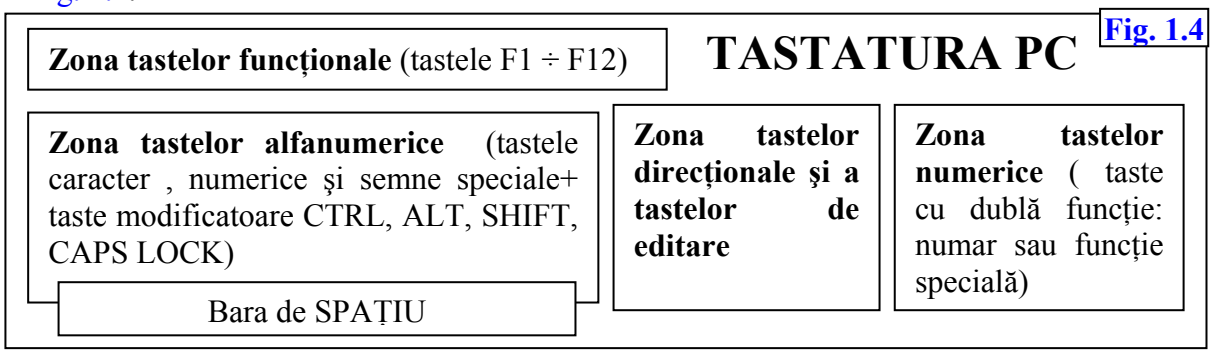


Fig. 1.4

Semnificația tastelor speciale este următoarea:

- tastele **SHIFT**, **CTRL**, **ALT** – nu lucrează individual ci întotdeauna impreună cu alte taste suplimentare, extinzând funcțiile acestora; aceste taste sunt dublate în stânga şi dreapta zonei tastelor alfanumerice; operarea decurge astfel: se apasă una din tastele speciale **SHIFT**, **CTRL**, **ALT** şi se menține apăsată în timpul apăsării tastei suplimentare asociate;
- **ENTER** – care marchează sfârşitul unei comenzi;
- Bara **SPAȚIU** – generează un caracter de tip spațiu, are o formă alungită şi uşor accesibilă;
- **NumLock** – poziționată în zona superioară a tastelor numerice şi care activează funcția de număr sau de funcție specială a tastelor din această zonă;
- Tastele direcționale ← , → , ↑ ,↓ - permit mişcarea cursorului cu o poziție în sensul indicat (caracter stânga/dreapta respectiv rând sus/jos)
- Taste pentru mişcări mai ample **PgUp**, **PgDn**, **Home**, **End** – afişare ecran precedent/următor pentru informații care depăşesc nivelul unui ecran respectiv poziționare cursor la început/sfârşit de pagină curentă sau rând;
- Tasta **INSERT** - comutare între modurile de scriere *inserare* / *suprascriere*; modul *inserare* provoacă includerea noilor caractere între cele existente, iar modul *suprascrirere* provoacă înlocuirea succesivă a caracterelor existente cu cele noi introduse.
- Taste de corecție:
  - **DELETE** – ştergerea caracterului pe care se află poziționat cursorul;
  - **BackSPACE** – ştergere caracter din stânga cursorului;
- Tastele **Start** şi **Application** –pentru tastatură de tip Windows, care au efectul deschiderii meniului **Start** respectiv înlocuire buton dreapta mouse.
- Tasta **Pause** – tastă de pauză pentru întrerupere temporară;
- **Print Screen** – generează copierea ecranului la imprimantă;
- **Scroll Lock** – oprirea afişajului derulant de pe ecran.

Apăsarea un timp mai îndelungat a unei taste generează transmiterea către calculator în mod repetat a aceluiaşi caracter, proces care trebuie evitat, exceptând situația în care se dorește în mod expres execuția acestei acțiuni.

1.3.3.2 Mouse

Este o completare a tastaturii şi funcționează pe următorul principiu: mişcarea dispozitivului pe suprafață este transmisă prin rotirea unei bile şi înregistrată de senzori, ceea ce va genera deplasarea pe ecran a unui cursor special, denumit *cursor de mouse*. Prin intermediul cursorului de mouse se pot selecta mai rapid comenzi prin intermediul butoanelor mouse (care pot fi 2 sau 3 butoane).

1.3.3.3 Monitorul video (display)

Este perifericul standard de ieşire al PC-urilor şi permite afişarea de texte şi imagini grafice. Monitoarele se cuplează la calculator prin intermediul unui dispozitiv denumit adaptor video sau placă grafică. Monitorul are următoarele caracteristici:

- **definiția** - diametrul unui pixel; valoarea tipică a definiției este de 0,28 mm;
- **rezoluția –** numărul de pixeli de pe o linie respectiv coloană a monitorului, valori uzuale: 640x480, 800x600, 1024 x768;
- **numărul de culori** – monocrome şi color;
- **dimensiunea diagonalei** – se exprimă în inch (1 inch=25,4 mm); valori uzuale: 14”, 15”, 17”, 19”, 21”, 24”, valoarea monitorului crescând cu dimensiunea diagonalei;
- **gradul de periculozitate al radiațiilor** – noile tipuri sunt cu radiație scăzută (**low radiation**), care nu sunt dăunătoare, cu condiția folosirii lor max. 6 ore/zi; norme care impun standarde ale nivelului radiației sunt MPR-II sau TCO;
- **rata de** numărul de cadre posibil de afişat într-o secundă; cu cât sunt mai mari cu atât creşte calitatea imagini;
- modul de lucru **întrețesut sau neântrețesut** – în modul *neântrețesut* ecranul este parcurs dintr-o singură trecere, iar în modul *întrețesut (interlaced)* din două treceri: prima pentru liniile impare, a doua pentru liniile pare, timpul de baleiaj fiind acelaşi în ambele variante. Calitatea este mai bună pentru modul neântrețesut.

1.3.3.4 Imprimantele

Sunt echipamente destinate imprimării rezultatelor sub formă textuală sau grafic. Caracteristicile unei imprimante sunt:

- **rezoluția** – exprimată în număr de puncte/inch;
- **viteza de tipărire** – exprimată în caractere /secundă, pagini/secundă;
- **dimensiunea maximă a hârtiei** – format A4, A3, etc.;
- **memoria imprimantei** – de ordinul MB.

După principiul de funcționare, imprimantele uzuale pot fi:

- **imprimante cu jet de cerneală** – caracterele se obțin dintr-o matrice fină de puncte, prin depunerea pe hârtie a unor picături fine de cerneală;

- **matriceale (cu ace)** – imprimarea unui caracter se face prin intermediul unei matrici de ace, care lovesc o bandă tuşată; există imprimante cu 9 sau 24 ace care au viteze de ordinul sutelor de caractere/secundă;
- **imprimante laser** – imprimarea se face pe baza principiului de funcţionare al xerox-urilor: o rază laser polarizează electrostatic gradat un cilindru special, pe care se agaţă toner-ul (praf de cărbune) care se va depune pe hârtie.

1.3.3.5 Hard disk-ul

Este un suport de stocare reutilizabil, realizat fizic sub forma unui pachet de discuri coaxiale (din aluminiu sau material plastic), cu feţele acoperite de un strat subţire de material magnetic (de ordinul micronilor).

. **Tehnologia de înregistrare** – tehnologia are la bază principiul înregistrării magnetice (ca şi la benzile audio). Suprafaţa de înregistrare este tratată ca o succesiune de puncte ce primesc un echivalent magnetic 0 sau 1. Poziţiile acestor punct sunt precis identificabile prin *marcatori*. Operaţia de formatare stabileşte poziţia acestor marcatori.

Metoda de acces la date – hard discul are o mişcare de rotaţie, de ordinul miilor de rotaţii /minut. Citirea şi scrierea datelor se face prin intermediul capetelor de citire-scriere montate pe un cărucior mobil. Mişcarea capetelor este radială, fig. 1.5.

Combinaţia dintre mişcarea circulară a suprafeţei discului şi mişcarea lineară a capului determină accesul rapid la informaţia de pe disc (de ordinul 10-50 milisecunde).

Organizarea suprafeţei hard-disk-ului – suprafaţa unei feţe de disc este împărţită în cercuri concentrice numite *piste* (*tracks*), numerotate de la 01 la n-1, n fiind numărul pistelor, prima pistă fiind cea de la exterior, fig. 1.6.

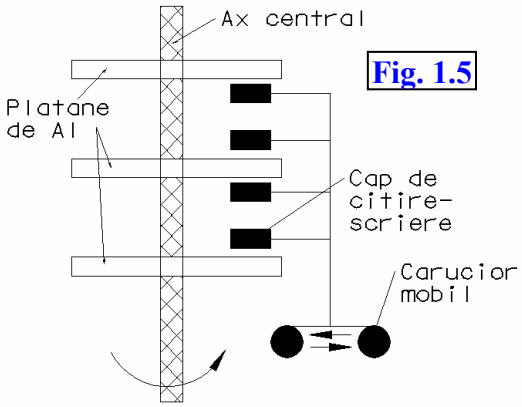


Fig. 1.5

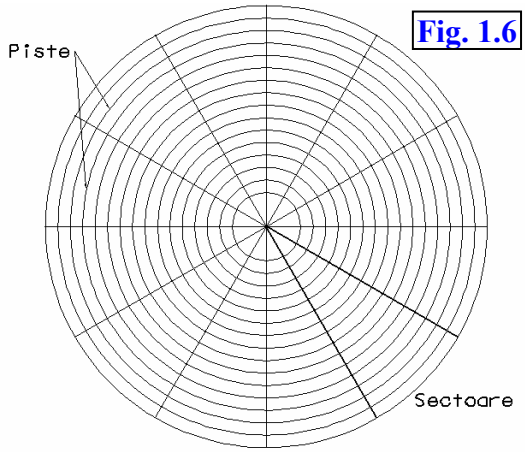


Fig. 1.6

Toate pistele cu acelaşi diametru de pe feţele discurilor formează un *cilindru*, numărul de cilindrii fiind egal cu cel al pistelor. Scrierea-citirea informaţiilor se face pe cilindrii. Fiecare pistă se împarte în sectoare, valoarea standard fiind de 512 B/sector.

Capacitatea de memorare se poate calcula din relaţia:

nr. bytes/sector x **nr. sectoare/pistă** x **nr. piste/faţă** x **nr. feţe.**

1.3.3.6 Discul flexibil (floppy disc)

Constructiv este format dintr-o singură folie magnetică, acoperită cu strat magnetic şi protejat de un înveliş. Tehnologia de înregistrare şi organizarea datelor este similară hard-discului. Modul de acces la date se realizează prin capete de citire-scriere, timpul de acces fiind de 1/6 s.

1.4 LIMBAJE DE PROGRAMARE

Vom înțelege prin PROGRAM forma finală în care se prezintă calculatorului un algoritm pentru rezolvare.

Singurele programe care pot fi înțelese direct de către unitatea centrală de prelucrare (UCP) sunt cele care folosesc un set de bază de instrucțiuni (cod maşină) specifice tipului de procesor aflat în UCP.

Primele limbaje de programare (1940) numite **LIMBAJE COD MAŞINĂ** erau scrise în cod binar. Ele permiteau scrierea programelor sub formă de comenzi, fiecărei operații i se asocia un cod. Astfel, fiecare instrucțiune trebuia să conțină: codul operației, adresa primului operand, adresa celui de-al doilea operand, adresa rezultatului.

O expresie aritmetică trebuia deci descompusă în operații cu doi operanzi, rezultatele intermediare devenind operanzi în operațiile următoare. De asemenea trebuiau cunoscute adresele datelor de intrare şi de ieşire, ceea ce transforma programarea într-o operație laborioasă, cu consum de timp şi energie umană.

Următoarea generație de limbaje sunt **LIMBAJELE DE ASAMBLARE**, cu următoarele facilități:

- în locul codului binar, instrucțiunile sunt scrise sub forma unor mnemonice (ex. ADD sau STO de la STORE)
- referințele locațiilor de memorie sunt apelate sub formă de nume;
- numerele nu mai sunt reprezentate binar, ci zecimal sau hexazecimal (sistem de numerație cu baza 16);
- grupurile de instrucțiuni care se repetă în cadrul algoritmului, pot fi grupate sub formă de macrou şi apelate prin nume.

Translatarea programului din limbaj de asamblare în cod maşină se face prin intermediul programelor asambloare, ce traduc PROGRAMUL SURSĂ instrucțiune cu instrucțiune, transformându-l în PROGRAM COD OBIECT, ce poate fi rulat pe sistemul de calcul.

Dezavantajele limbajelor de asamblare sunt:

- fiecărui tip de procesor îi sunt asociate un set diferit de instrucțiuni şi un limbaj de asamblare aparte, deci programele scrise în limbaje de asamblare nu sunt portabile (nu pot fi rulate pe alte procesoare);
- limbaj de asamblare sunt orientate maşină, adică se fac referințe la regiştrii din unitatea aritmetică şi logică (UAL), programatorul fiind concentrat mai mult pe aspectele fizice legate de maşină, decât de logica programului;
- programarea în limbaj de asamblare este consumatoare de timp, fiecare operație trebuie programată la nivel de detaliu.

Avantajul programării în limbaj de asamblare este utilizarea eficientă a maşinii, ceea ce se traduce printr-o viteză de execuţie mai mare a programelor. De aceea aceste limbaje se folosesc pentru scrierea sistemelor de operare, care necesită o viteză ridicată de execuţie.

Pentru depăşirea acestor inconveniente, s-au elaborat limbaje evoluate, numite **LIMBAJE SUPERIOARE (DE NIVEL ÎNALT)**. O instrucţiune scrisă într-un limbaj de nivel înalt este echivalentă cu una sau mai multe instrucţiuni în limbaj cod maşină.

Avantajele folosirii limbajelor de nivel înalt:

- permit scrierea programelor într-o formă mai apropiată de limbajul natural (utilizând limba engleză), dar sunt supuse unor reguli mai stricte;
- sunt teoretic portabile de pe un calculator pe altul, cu mici diferenţe, care necesită însă modificări minore;
- specializarea limbajelor pe tipuri de aplicaţii, ceea ce conduce la creşterea productivităţii programatorilor; exemple de limbaje de programare de nivel înalt:
  - FORTRAN (FORmula TRANslator) - a fost primul limbaj de nivel înalt, creat în anul 1955 şi utilizat pentru scopuri ştiinţifice şi matematice;
  - ALGOL – dezvoltat în anul 1958 şi este destinat metodelor de rezolvare a problemelor de mare complexitate;
  - COBOL – limbaj dezvoltat în anii ‘60-’61 pentru procesarea datelor;
  - BASIC - dezvoltat în anul 1965 în scopuri didactice, pentru a fi uşor de înţeles şi de asimilat de către studenţi, fiind şi în prezent unul din cele mai răspândite limbaje de programare;
  - PASCAL – este un limbaj de programare puternic structurat, dezvoltat la începutul anilor ’60; aplicaţia este divizată în mai multe module, fiecare modul executând o anumită operaţie logică;
  - PROLOG – limbaj utilizat în aplicaţii de inteligenţă artificială.

Forma programului scrisă în limbaj de nivel înalt se numeşte **PROGRAM SURSĂ**.

Programul sursă nu poate fi executat de către calculator, ci trebuie tradus în cod maşină, adică transformat în **PROGRAM COD OBIECT**, pentru a putea fi rulat de către calculator, operaţie care se face, prin intermediul unor programe translatoare, în două feluri:

- prin **INTERPRETARE** - fiecare linie a programului este tradusă, prin intermediul unui program interpretor, în cod obiect şi executată imediat;
- prin **COMPILARE** - caz în care are loc traducerea întregului program, printr-un program compilator, după care programul obiect obţinut este executat, fără a mai fi nevoie de programul sursă; de asemenea, în această formă programul este mai greu de înţeles şi modificat. Programele compilate nu rulează decât după eliminarea tuturor erorilor de sintaxă.

Programele interpretate lucrează mai încet decât cele compilate, deoarece traducerea în cod maşină se face la fiecare execuţie, pentru fiecare linie, dar sunt mai accesibile pentru începătorii în programare.

2. ALGORITMI

2.1 ALGORITMI. CARACTERISTICI.

Caracteristica principală a proceselor de prelucrare automată a informației este natura lor algoritmică şi efectuarea lor cu ajutorul maşinilor de calcul. Calculatoarele sunt capabile să rezolve probleme doar dacă acestea sunt descompuse în secvențe de calcul elementare, precizându-se totodată şi succesiunea acestora.

Conceptul de algoritm este fundamental pentru informatică, fiind menționat prima oară în cartea "*Liber Algorithmi*" a învățatului Abu Jafar Mohammed Ibn Musa al Khovarzmi, ce a trăit în Asia Centrală în jurul anului 850 după Hristos.

Ca şi alte noțiuni din matematică sau alte domenii, noțiunea de algoritm nu este explicată printr-o definiție riguroasă, unic acceptată. Lipsa unei definiții nu împiedică însă utilizarea acestora în domeniul informatic. În consecință, conceptul de algoritm se va introduce prin evidențierea caracteristicilor sale fundamentale.

Algoritmul (ALG) este un sistem de reguli care transformă informația inițială (*datele de intrare*) într-o informație finală (*date de ieşire*), trecând printr-un şir de informații intermediare, fig. 2.1.

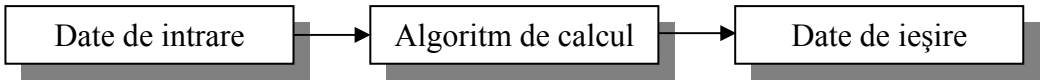


Fig. 2.1

Caracteristicile algoritmilor:

- **generalitate** – algoritmul nu trebuie să rezolve numai o problemă, ci toate problemele din aceeaşi clasă;
- **finitudine** – numărul de transformări intermediare aplicate asupra informației inițiale pentru a obține informația finală trebuie să fie finit; această caracteristică ne asigură de faptul că algoritmul se termină într-un timp finit;
- **unicitate** – toate transformările intermediare făcute asupra informației inițiale sunt unic determinate de regulile algoritmului;
- **corectitudine** – informația finală trebuie să corespundă unei rezolvări corecte a problemei, adică legătura dintre informația inițială şi cea finală trebuie să fie cea impusă de problema pentru care s-a elaborat algoritmul.

Informația de care dispunem la un moment dat se exprimă printr-un limbaj specializat prin intermediul literelor, cuvintelor şi simbolurilor matematice.

După elaborarea algoritmului, el poate fi executat de orice persoană sau maşină capabilă să efectueze operațiile prevăzute de algoritm, chiar dacă nu cunoaşte fundamentarea matematică a problemei şi nici măcar problema în sine. Acest fapt stă la baza executării algoritmilor de către calculator. Altfel spus, algoritmul este un set de reguli aplicate mecanic datelor de intrare pentru a genera datele de ieşire.

În consecință, un calculator trebuie să fie capabil să îndeplinească următoarele acțiuni:

- să poată primi (să citească) datele de intrare şi algoritmul de rezolvare, memorate în memoria internă MI;

- să poată executa operaţiile prevăzute de algoritm, care se realizează prin unitatea aritmetică şi logică UAL;
- să poată comunica (scrie) utilizatorului datele de ieşire, prin intermediul echipamentelor de ieşire.

Forma finală în care se prezintă calculatorului un algoritm se numeşte **program**. Un program este format din **instrucţiuni**, forma acestora şi modul în care pot fi de asamblate pentru a produce un program variază funcţie de **limbajul de programare**.

Oricărei probleme care admite o formulare matematică i se poate asocia un algoritm de rezolvare, care însă nu trebuie confundat nici cu formularea matematică a acesteia şi nici cu codificarea sub formă de program exprimat într-un limbaj de programare.

Scopul final în cadrul procesului de programare este elaborarea unui program într-un limbaj de programare, proces în care elaborarea algoritmului nu este o etapă neapărat obligatorie, dar totuşi recomandabilă, mai ales în fazele de început ale apropierii de domeniul programării calculatoarelor. Odată cu câştigarea experienţei, exprimarea algoritmică devine tot mai concisă şi mai schematică.

O ultimă şi interesantă observaţie asupra algoritmilor se cuvine a fi menţionată: nu există un algoritm universal pentru elaborarea oricărui algoritm. Pentru o problemă specificată se pot elabora mai mulţi algoritmi de rezolvare, în general interesându-ne cea mai eficientă dintre acestea. Din acest motiv elaborarea unui algoritm nu este întotdeauna o sarcină simplă, mai ales în cadrul problemelor de mare complexitate. Totuşi, similar cu rezolvarea matematică a problemelor, elaborarea de algoritmi este o calitate intelectuală care se poate dobândi şi perfecţiona prin exerciţiu.

2.2 EXPRIMAREA ALGORITMILOR PRIN LIMBAJUL NATURAL

În matematică numeroase probleme sunt rezolvate cu ajutorul algoritmilor. Exemple: adunarea, scăderea, înmulţirea, împărţirea numerelor, calculul rădăcinii pătrate dintr-un număr, algoritmul de rezolvare a ecuaţiilor de grad I, II, III, IV, rezolvarea sistemelor de ecuaţii, etc.

În cele ce urmează vom prezenta două exemple de algoritmi, exprimaţi în limbaj natural, adică într-un limbaj descriptibil, în care fiecare pas al algoritmului este exprimat prin cuvinte preluate din limbajul comun.

E 2.1 Algoritmul de calcul a modulului unui număr real

Fie un număr real "X" arbitrar. Modulul (valoarea absolută) notat prin $|X|$ se defineşte prin:

$$|X| = \begin{Bmatrix} X & daca & X > 0 \\ 0 & daca & X = 0 \\ -X & daca & X < 0 \end{Bmatrix}$$

Considerând X ca dată de intrare, iar Y ca dată de ieşire, algoritmul de calcul este prezentat în fig. 2.2:

1. Început algoritm
2. Citeşte valoarea lui X.
3. Compară pe X cu 0. Dacă X este zero sau pozitiv,
   3.1 atunci
      3.1.1 Y primeşte valoarea X
      3.1.2 execută pasul 4 → 4
   3.2 în caz contrar
      3.2.1 Y primeşte valoarea –X
      3.2.2 execută pasul 4 → 4
4. Afişează valoarea lui Y
5. Sfârşit algoritm.

Fig. 2.2

E 2.2 Algoritmul de rezolvare a ecuației de grad I

Forma generală a ecuației de gradul I, este:

$$a X + b = 0,\ a \neq 0,\ (a, b \in R) \text{ cu soluția } X = -b / a$$

Există următoarele cazuri posibile:

1. Dacă $a \neq 0$ => soluția $X = -b / a$
2. $a = 0$ şi $b \neq 0$ => "*imposibil*"
3. $a = 0$ şi $b = 0$ => "*identitate*", ecuația fiind valabilă pentru orice X.

Considerând a, b ca date de intrare, iar X dată de ieşire, algoritmul de calcul este prezentat în fig. 2.3:

1. Început algoritm
2. Se citesc datele de intrare a, b.
3. Dacă $a \neq 0$
   3.1 atunci
      3.1.1 calculează X = -b / a
      3.1.2 afişează X
      3.1.3 execută pasul 4 → 4
   3.2 în caz contrar, dacă b = 0
      3.2.1 atunci
         3.2.1.1 afişează '*identitate*'
         3.2.1.2 execută pasul 4 → 4
      3.2.2. în caz contrar
         3.2.2.1 afişează '*imposibil*' .
         3.2.2.2 execută pasul 4 → 4
4. Sfârşit algoritm.

Fig. 2.3

În cadrul acestor exemple algoritmi prin "*citirea datelor de intrare*" vom înțelege o modalitate de atribuire de valori numerice pentru acestea, iar prin "*afişare*" vom înțelege modalitatea de obținere a rezultatelor sub formă numerică sau a unor mesaje de informare, fără a ne interesa în acest moment suportul informatic al acestor acțiuni.

2.3 MODURI DE REPREZENTARE A ALGORITMILOR

Limbajul natural nu permite o descriere riguroasă a algoritmilor, iar complexitatea descrierii creşte în mod substanțial odată cu complexitatea algoritmului, ceea ce se constituie ca un handicap major pentru înțelegerea acestuia. Pentru standardizarea modului de reprezentare a algoritmilor, s-au definit mai multe moduri de reprezentare a acestora. In cele ce urmează vor fi abordate două dintre cele mai utilizate moduri de reprezentare: **scheme logice** şi **limbaj algoritmic** sau **pseudocod**. Principala calitate a acestora este de a evidenția cu claritate succesiunile posibile ale acțiunilor algoritmului.

În continuare vor fi definite noțiuni utilizate în paragrafele imediat următoare:

- **variabile** – reprezintă date ale căror valori se pot modifica în cursul execuției algoritmului, care au un conținut numeric sau textual şi sunt identificate prin nume simbolice; într-o variabilă, poate fi memorată o singură valoare; reintroducerea unei noi valori într-o variabilă provoacă pierderea valorii sale anterioare şi înlocuirea cu noua valoare;
- **constante** – reprezintă date ale căror valori nu se modifică în cursul execuției algoritmului, care au un conținut numeric sau textual şi sunt identificate prin nume simbolice;
- **mesaje** – o succesiune de caractere cuprinse între ghilimele;
- **expresii** – o succesiune de variabile, constante şi operatori matematici. Expresiile se scriu linearizat (adică pe o singură linie). Principalii operatorii sunt: + adunare , - scădere , * înmulțire respectiv / împărțire, SQRT - radical.

2.3.1 Scheme logice

Schemele logice realizează o reprezentare grafică a algoritmului şi sunt formate din:

- **instrucțiuni** - care descriu grafic acțiunile algoritmului;
- **săgeți** – care indică sensul transmiterii informației.

Acest mod de reprezentare a algoritmilor mai poartă şi numele de **scheme bloc** sau **organigramă**. În continuare vor fi descrise instrucțiunile principale utilizate în schemele logice, reprezentarea grafică a acestora fiind specificată în fig. 2.4.

a) Instrucțiunea **START** , fig. 2.4.a – marchează locul începerii algoritmului.

b) Instrucțiunea **STOP**, fig. 2.4.b – marchează sfârşitul algoritmului.

c) Instrucțiunea **CITEŞTE**, fig. 2.4.c – determină citirea datelor de intrare. După cuvântul cheie **CITEŞTE** se indică o lista de variabile separate prin virgulă. Vom înțelege prin "*citire*" preluarea de valori de la dispozitivul logic de intrare şi alocarea acestor valori variabilelor din listă.

d) Instrucţiunea **SCRIE**, fig. 2.4.d – determină scrierea datelor de ieşire, care pot fi:

- *variabile* – scrierea unei variabile constă în afişarea valorii ei curente şi nu a numelui acesteia;
- *mesaje* – scrierea unui mesaj constă în afişarea succesiunii de caractere dintre ghilimele;
- *expresii* – scrierea unei expresii constă în afişarea valorii expresiei, valoare rezultată în urma efectuării tuturor calculelor expresiei.

După cuvântul cheie **SCRIE** se indică o lista de variabile, mesaje sau expresii separate prin virgulă. Vom înţelege prin "*scriere*" afişarea pe un suport de informaţie a valorii unor variabile sau expresii, respectiv a unor mesaje informative.

e) Instrucţiunea de **ATRIBUIRE** – are forma din fig. 2.4.e, unde V este o variabilă, iar E este o expresie. Execuţia ei constă în evaluarea (calculul) expresiei E (din dreapta semnului "=") şi atribuirea rezultatului variabilei V (din stânga semnului "="); în urma atribuirii valoarea anterioară a variabilei V (dacă a existat) se va pierde.

- Evaluarea expresiei E se face astfel: se înlocuiesc variabilele din expresie cu valorile lor curente şi se efectuează calculul numeric.
- O instrucţiune de atribuire trebuie să specifice întotdeauna o variabilă care primeşte o nouă valoare, rezultată dintr-un calcul, precum şi o formulă de obţinere a acestui rezultat.
- Este sarcina programatorului ca, în momentul evaluării expresiei, toate variabilele care intervin în expresie să aibe valori atribuite (prin citire sau prin instrucţiuni anterioare de atribuire).
- Semnul "=" din instrucţiunea de atribuire nu trebuie confundat ca semnificaţie cu acelaşi semn din domeniul matematicii. Astfel, semnificaţia informatică a acestui semn este de **atribuire** (în sensul alocării unei valori unei variabile), spre deosebire de semnificaţia matematică a acestuia, care este de **identitate** (în sensul egalităţii valorice a membrului stâng şi drept a unei formule). În contextul precizat anterior, o expresie de genul: **a = a + 1**, este incorectă şi falsă din punct de vedere matematic, oricare ar fi valoarea variabilei **a**, pe când, din punct de vedere informatic, este corectă şi trebuie interpretată astfel: la valoarea curentă a variabilei **a** se adaugă 1, iar rezultatul obţinut se atribuie, prin înlocuire, aceleiaşi variabile **a**; altfel spus valoarea variabilei **a** creşte cu o unitate.

f) Instrucţiunea de **DECIZIE** – are forma din fig. 2.4.f se utilizează pentru selecţia unei singure alternative din două posibile şi execuţia acţiunii variantei selectate funcţie de realizarea condiţiei "C". Execuţia acestei instrucţiuni constă în :

- evaluarea condiţiei "C" pentru valorile curente ale variabilelor care intervin în condiţie;
- dacă condiţia este adevărată, algoritmul se continuă pe ramura notată cu DA;
- dacă condiţia este falsă, algoritmul se continuă pe ramura notată cu NU;

După cum se observă, funcţie de valoarea condiţiei, numai una dintre cele două ramuri ale algoritmului se execută, nu amândouă. Condiţia care intervine în cadrul instrucţiunii este o expresie logică, al cărei rezultat poate lua una din valorile logice **True** (adevărat)

sau **False** (neadevărat). O expresie logică se formează din variabile şi constante legate între ele prin operatori logici **NOT** (NU logic), **AND** (ŞI logic), **OR** (SAU logic) şi / sau operatori relaţionali: = , <> , < , > , >= , >= , cu semnificaţiile lor consacrate din logica matematică.

g) Instrucţiunea de **PROCEDURĂ**, fig. 2.4.g – indică transferul execuţiei algoritmului către o procedură de calcul (secvenţă de instrucţiuni a cărei execuţie se poate repeta de mai multe ori în cadrul programului, eventual pentru alte date de intrare). Procedura se mai numeşte şi **subprogram** sau **subrutină** şi poate fi de două feluri:

- **funcţii** – subprograme care returnează o singură valoare
- **proceduri** – care returnează una, mai multe valori sau nici una.

În interiorul simbolului grafic se specifică numele procedurii, datele de intrare şi ieşire spre respectiv din procedură.

h) **Conector de pagină**, fig. 2.4.h – leagă puncte ale schemei logice aflate pe pagini diferite; de obicei se marchează cu litere greceşti. Conectorii apar în pereche pe pagini diferite şi realizează racordarea, prin acelaşi identificator, a conexiunii ramurilor.

i) **Săgeata**, fig. 2.4.i – leagă instrucţiunile între ele şi indică sensul transmiterii informaţiei.

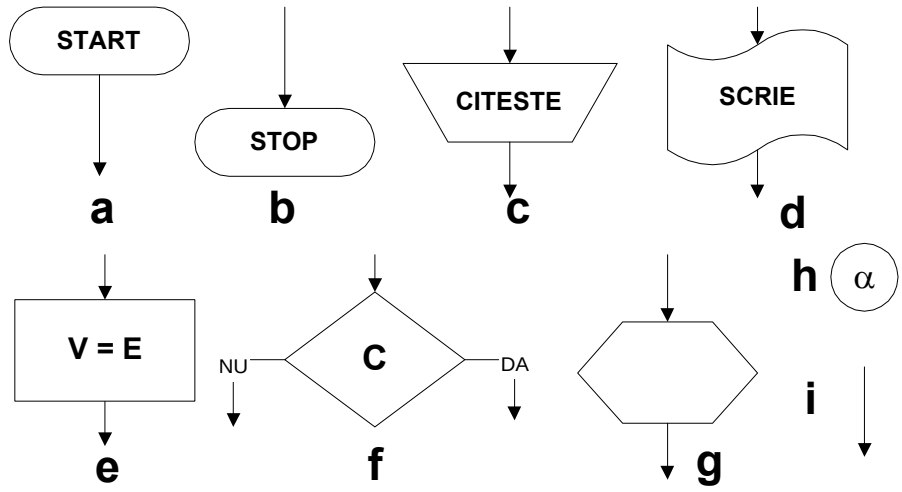


Fig. 2.4

O schemă logică este formată prin înlănţuirea instrucţiunilor, conform următoarelor reguli:

- orice extremitate *iniţială* a unei instrucţiuni trebuie să fie extremitate finală a cel puţin unei instrucţiuni;
- orice extremitate *finală* a unei instrucţiuni trebuie să fie extremitate iniţială a unei singure instrucţiuni;
- trebuie să existe o *unică* instrucţiune START respectiv STOP;

unde noţiunile de extremitate iniţială / finală sunt determinate de sensul de parcurgere stabilit de săgeţi.

În continuare, vor fi prezentate schema logică pentru algoritmul de calcul a modulului unui număr real, fig. 2.5 şi schema logică pentru algoritmul de rezolvarea a ecuaţiei de grad I, fig. 2.6, pentru comparaţie cu aceeaşi algoritmi exprimaţi prin limbaj natural, în & 2.2.

E 2.3 Schema logică pentru algoritmul de calcul a modulului unui număr real

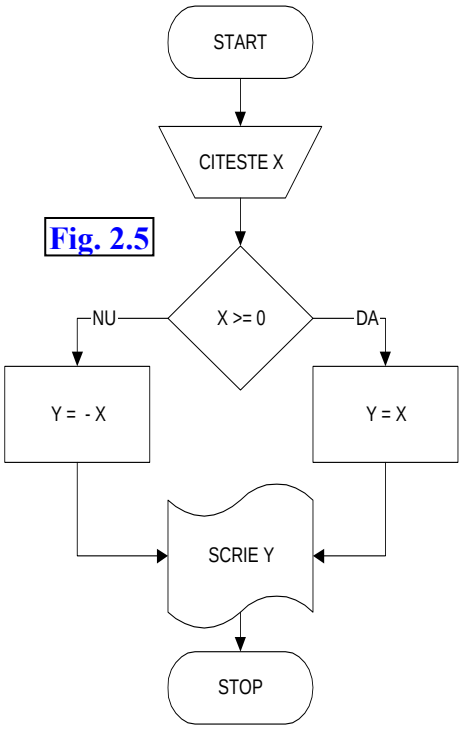


Fig. 2.5

Algoritmul de calcul a modulului unui număr real, exprimat în limbaj natural, este prezentat în exemplul E 2.1, iar fig. 2.5 prezintă schema logică corespunzătoare.

Instrucțiunile **START** şi **STOP** marchează începutul respectiv sfârşitul algoritmului. Instrucțiunile **CITEŞTE X** respectiv **SCRIE X** au rolul de citire a datei de intrare "X", respectiv afişare a datei de ieşire "Y".

Instrucțiunea de decizie este construită pe baza condiției **X >= 0**, funcție de care se atribuie valoarea modulului, respectiv: **X**, dacă condiția este îndeplinită (este adevărată) sau **–X**, dacă condiția nu este îndeplinită (este falsă).

Se observă deci că, numai una din cele două variante posibile se execută, însă oricare din cele două variante se parcurge, algoritmul continuă cu afişarea datei de ieşire, adică a valorii lui Y.

E 2.4 Schema logică pentru algoritmul de rezolvare a ecuației de grad I

Algoritmul de calcul pentru rezolvarea ecuației de gradul I, exprimat în limbaj natural, este prezentat în exemplul E 2.2, iar fig. 2.6 prezintă schema logică corespunzătoare.

Instrucțiunile **START** şi **STOP** marchează începutul respectiv sfârşitul algoritmului. Instrucțiunea **CITEŞTE X** are rolul de citire a datelor de intrare "a" şi "b".

Datele de ieşire variază funcție de varianta de parcurgere a schemei logice, variantă care depinde de valorile efective ale datelor de intrare "a" şi "b".

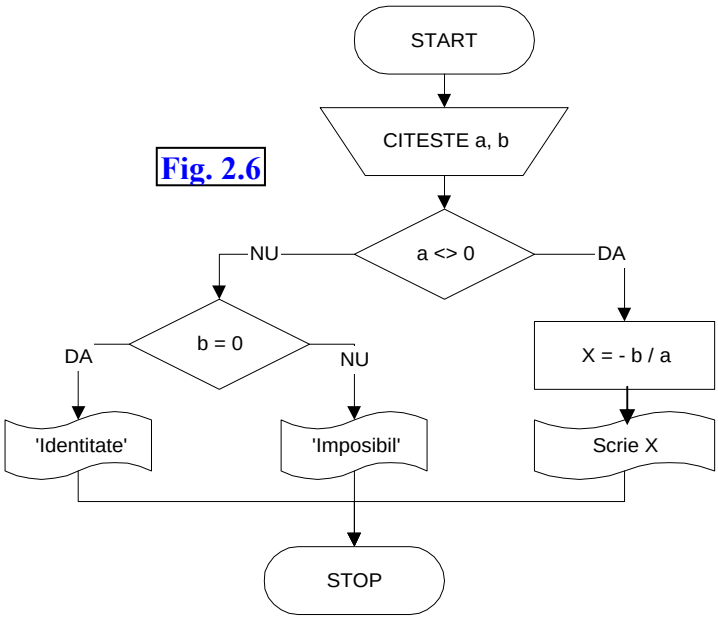


Fig. 2.6

După cum rezultă din schema logică, fig. 2.6, algoritmul se finalizează prin afişarea valorii variabilei "X", calculată anterior prin instrucţiunea de atribuire **X=-b/a**, pentru ramura adevărată a condiţiei **a <> 0**, respectiv sub forma unui mesaj de informare, pentru ramura falsă a aceleiaşi condiţii. Rolul acestui mesaj este de informare asupra modului de finalizare a algoritmului, respectiv:

- pentru valoarea nulă a variabilei "b" (ramura adevărată a condiţiei **b = 0**) - mesajul "**Identitate**", cu semnificaţia obţinerii, din punct de vedere numeric, a unei identităţi matematice, adică a unei egalităţi valabile oricare ar fi valoarea lui "X", pentru datele de intrare "a" şi "b" nule;
- pentru valoarea nenulă a variabilei "b" (ramura falsă a condiţiei **b = 0**) - mesajul "**Imposibil**", cu semnificaţia obţinerii, din punct de vedere numeric, a unei imposibilităţi matematice.

În schema logică din fig. 2.6 se observă linearizarea atribuirii X=-b/a, adică scrierea expresiei pe o singură linie şi nu în forma matematică a acesteia.

În algoritmi se folosesc variabile identificate prin nume, dar logica algoritmului lucrează cu echivalentul lor valoric. De exemplu, pentru fig. 2.6, în comparaţia **a <> 0**, nu se testează dacă litera "a" este diferită de 0, ci dacă valoarea variabilei "a" este diferită de 0, variabila "a" considerând ca primeşte o valoare numerică prin instrucţiunea de citire, indiferent care ar fi aceasta valoare. Procedeul este similar celui din matematică, unde, de exemplu, un sistem de 2 ecuaţii cu două necunoscute, se rezolvă literar, pentru a obţine formulele de rezolvare a sistemului, general valabile oricare ar fi valorile numerice ale coeficienţilor, dar în momentul rezolvării efective a sistemului, variabilele literare sunt înlocuite cu valori numerice concrete. Prin aceasta se asigura caracterul de generalitate al algoritmului (& 2.1).

2.3.2 Limbaj pseudocod

Reprezentarea unui algoritm prin schemă logică are avantajul simplităţii modului de exprimare, precum şi a claritătii vizuale a algoritmului, dezavantajul constituind în faptul că reprezentarea unui algoritm complex poate fi dificilă datorită întinderii şi ramificaţiilor sale. De aceea s-a căutat un limbaj mai apropiat de cel uman, care însă să nu impună cunoaşterea unui limbaj de programare. Descrierea algoritmului într-un asemenea limbaj este asemănătoare cu descrierea sa într-un limbaj de programare, fără a impune însă toate rigorile acestuia.

Limbajele pseudocod au avantajul că folosesc o sintaxă mult mai liberă decât ale limbajelor de programare, ultimele fiind însă singurele recunoscute de calculator. De asemenea un program scris în pseudocod poate fi recunoscut de toţi programatorii, în timp ce unul scris în limbaj de programare poate fi înţeles doar de cunoscătorii limbajului.

Vom înţelege prin program scris în limbaj pseudocod o succesiune de instrucţiuni ale limbajului. Limbajele de tip pseudocod sunt limbaje care folosesc **cuvinte cheie**, preluate intr-un limbaj natural, cu semnificaţii strict definite, care formează **lexicul** (vocabularul

limbajului). Regulile de folosire a cuvintelor cheie pentru formarea instrucțiunilor, împreună cu alte cuvinte / simboluri determină **sintaxa** limbajului.

Convenții ale limbajului pseudocod:

1. Algoritmul începe cu cuvântul cheie **PROGRAM** urmat de numele său, formând **antetul** programului. Numele programului nu este supus vreunei restricții sintactice.

2. Se admit comentarii încadrate între acolade {}; comentariile nu reprezintă acțiuni, ele fiind utile pentru mai buna înțelegere a programului şi pot fi plasate oriunde în cadrul programului.

3. Programul se încadrează între cuvintele cheie **INCEPUT** şi **SFÂRŞIT**, care însă nu include şi antetul acestuia. Aceste instrucțiuni corespund ca logică instrucțiunilor **START** respectiv **STOP** din schemele logice.

4. Cuvintele cheie vor fi scrise cu litere mari, iar instrucțiunile vor fi scrise / executate secvențial.

5. Mai multe instrucțiuni încadrate de cuvintele cheie **INCEPUT** şi **SFÂRŞIT** formează o **instrucțiune compusă**. Instrucțiunea compusă se plasează in interiorul altor instrucțiuni subiect, acolo unde instrucțiunea subiect impune, din punct de vedere al sintaxei, o singură instrucțiune, dar algoritmul necesită execuția mai multor instrucțiuni subordonate aceleiaşi instrucțiuni subiect. În interiorul instrucțiunii compuse se admit oricare alte instrucțiuni valide. Formularea sintactică a instrucțiunii compuse este prezentată în fig. 2.7, iar un posibil şi frecvent loc de aplicare al acesteia este instrucțiunea de decizie (pct. "d" din prezentul paragraf). Exemple concrete de aplicare sunt: algoritmul de rezolvare a ecuației de gradul I, exemplul E 2.6, respectiv a ecuației de gradul II, exemplul E 2.8. Instrucțiunea compusă reprezintă deci o modalitate de grupare a mai multor instrucțiuni într-o singură instrucțiune sintactică, grupare impusă de sintaxa instrucțiunii subiect.

Instrucțiune compusă
ÎNCEPUT
Instrucțiune 1
Instrucțiune 2
……
SFÂRŞIT

Fig. 2.7

6. Expresiile aritmetice se scriu sub formă lineară, pe un singur rând, folosind paranteze rotunde pentru modificarea priorității operațiilor, folosind operatorii: + adunare, - scădere, * înmulțire, / împărțire, SQRT radical.

7. Pentru asigurarea clarității programelor se recomandă alinierea instrucțiunilor pe nivele echivalente (*indentare*).

Instrucțiunile limbajului pseudocod sunt similare ca şi execuție şi logică cu cele ale schemelor logice (& 2.3.1), diferența fiind generată de modalitatea concretă de reprezentare a instrucțiunilor. Ca atare, toate considerațiile prezentate în & 2.3.1 rămân valabile şi pentru instrucțiunile corespondente ale limbajului pseudocod.

a) Instrucțiunea **CITEŞTE (*lista de variabile*)** – determină citirea datelor de intrare; în *lista de variabile*, acestea sunt enumerate una după alta, separatorul dintre ele fiind ",". De exemplu, instrucțiunea **CITEŞTE** (a,b,c) se referă la citirea variabilelor a, b, c.

b) Instrucțiunea **SCRIE (*lista de variabile*)** – determină scrierea datelor de ieşire; în *lista de variabile*, acestea sunt enumerate una după alta, separatorul dintre ele fiind ",". De exemplu, instrucțiunea **SCRIE** (a,b,c) se referă la scrierea variabilelor a, b, c.

c) Instrucțiunea de atribuire are forma **V := expresie**, cu aceleaşi semnificații şi considerente ca şi la instrucțiunea corespondentă de la schemele logice. Se remarcă operatorul "**:=**", prin care se accentuează sensul de atribuire şi nu de identitate matematică, explicațiile aferente diferenței de sens fiind prezentate la pct. "e" din & 2.3.1.

d) Instrucțiunea de decizie se exprimă prin sintaxa :

DACĂ C **ATUNCI** I1 **ALTFEL** I2,

unde vom înțelege prin "C" – condiție, iar prin "I1" respectiv "I2" câte o unică instrucțiune. Instrucțiunea de decizie poate fi scrisă linear sau pe mai multe linii. Logica şi execuția instrucțiunii constă în testarea condiției "C" şi execuția instrucțiunii "I1", dacă condiția este adevărată respectiv execuția instrucțiunii "I2", dacă condiția este falsă; după execuția unei singure instrucțiuni dintre cele două ("I1" sau "I2") execuția programului continuă cu instrucțiunea imediat următoare instrucțiunii de decizie; ramura **ALTFEL** "I2" poate lipsi, în situația în care algoritmul nu impune nimic din punct de vedere al execuției pentru ramura neadevărată a condiției "C".

Se observă faptul că sintaxa instrucțiunii de decizie, impune execuția, unei singure instrucțiuni pe oricare din cele două ramuri. Dacă însă algoritmul necesită execuția mai multor instrucțiuni asociate aceleiaşi ramuri, atunci acestea se încadrează între cuvintele cheie **INCEPUT** şi **SFÂRŞIT**, formând astfel o instrucțiune compusă, care este considerată, din punct de vedere al sintaxei instrucțiunii de decizie, ca şi cum ar fi o singură instrucțiune (vezi pct. 5 din prezentul paragraf).

Instrucțiunea de decizie admite ca instrucțiune executabilă, pe oricare din cele două ramuri, inclusiv o altă instrucțiune de decizie, care este contabilizată din punct de vedere al numărului de instrucțiuni ca şi o unică instrucțiune; aceasta, deoarece, instrucțiunea de decizie reprezintă sintactic o singură instrucțiune, chiar dacă ea include în formularea sintactică, pe fiecare din cele două ramuri, instrucțiuni executabile.

e) Funcție de necesități impuse de algoritm se poate folosi instrucțiunea de salt necondiționat

TRECI LA PASUL A,

unde "A" este o etichetă ce marchează o altă instrucțiune din program. Instrucțiunea provoacă transferul necondiționat al execuției programului la instrucțiunea marcată cu eticheta "A" şi se poate folosi în algoritmi care impun revenirea la instrucțiuni precedente ale programului pentru reexecutarea lor. Eticheta poate fi de tip numeric sau identificabilă prin nume (vezi exemplul 2.19).

În continuare, vor fi prezentate limbajul pseudocod pentru algoritmul de calcul a modulului unui număr real şi pentru algoritmul de rezolvarea a ecuației de grad I, pentru

comparație cu aceeaşi algoritmi exprimați prin limbaj natural, în & 2.2., respectiv în scheme logice în fig. 2.5 şi fig. 2.6.

E 2.5 Limbaj pseudocod ptr. algoritmul de calcul a modulului unui număr real

```
PROGRAM Calcul modul
{Calculul modulului unui număr real }
ÎNCEPUT
        CITEŞTE (X)
        DACĂ X >= 0 ATUNCI
                Y: = X
        ALTFEL
                Y: = -X
        SCRIE (Y)
SFÂRŞIT
```

Fig. 2.8

Din fig. 2.8 se observă antetul programului, format din cuvântul cheie **PROGRAM** şi numele acestuia.

Comentariul încadrat între acolade are efect explicativ, fiind util celui care citeşte algoritmul, dar fără a avea vreun efect în decursul execuției acestuia.

Instrucțiunea de decizie, formulată pe baza condiției **X >= 0**, provoacă atribuirea valorii "X" variabilei "Y", pe ramura adevărată a condiției, respectiv a valorii "–X" pe ramura falsă.

După afişarea valorii variabilei "Y" algoritmul se încheie, prin **SFÂRŞIT**.

E 2.6 Limbaj pseudocod pentru algoritmul de rezolvare a ecuației de grad I

```
PROGRAM Ec grad 1
{ Rezolvarea ecuației de grad 1 }
ÎNCEPUT
        CITEŞTE (a, b)
        DACĂ a <> 0 ATUNCI
                ÎNCEPUT
                        X: = -b/a
                        SCRIE (X)
                SFÂRŞIT
        ALTFEL
                DACĂ b = 0 ATUNCI
                        SCRIE ('Identitate' )
                ALTFEL
                        SCRIE ('Imposibil')
SFÂRŞIT
```

Instrucțiune compusă

Fig. 2.9

După citirea datelor de intrare, respectiv variabilele "a", "b", instrucțiunea de decizie, formulată pe baza condiției :
a <> 0, provoacă, pe ramura adevărată a condiției, execuția a două instrucțiuni, respectiv: calcul rădăcinii "X" prin instrucțiunea de atribuire şi afişarea rădăcinii. Deoarece sintaxa instrucțiunii de decizie impune pe oricare din ramuri execuția unei singure instrucțiuni, dar algoritmul impune 2 instrucțiuni de executat pe această ramură, acestea trebuie încadrate între cuvintele cheie **INCEPUT** şi **SFÂRŞIT**, formând astfel instrucțiunea compusă, considerată sintactic ca o unică instrucțiune. Această instrucțiune este evidențiată grafic explicativ în programul alăturat, cu observația că marcatorii grafici nu fac parte din limbajul pseudocod.

Ramura falsă a condiției **a <> 0**, provoacă, execuția unei alte instrucțiuni de decizie, cu condiția **b = 0**, care include instrucțiuni de afișare de mesaje informative, dar care în ansamblul ei reprezintă o unică instrucțiune (de decizie) şi deci nu trebuie să formeze o instrucțiune compusă.

2.4 ELEMENTE ALE PROGRAMĂRII STRUCTURATE

Pentru cazul algoritmilor de mare complexitate, libera utilizare a instrucțiunilor, poate genera mari probleme în înțelegerea acestora. Problema devine cu atât mai complicată cu cât algoritmul face apel la mai multe reveniri respectiv ramificații. Programarea structurată constituie în răspuns la aceste complicații şi se constituie ca o metodă prin care generează o standardizare a programării, care însă este independentă de limbajul de programare.

Programarea structurată are meritul de uşura citirea şi clarifica algoritmilor, comparativ cu reprezentarea lor în formă clasică şi care depinde foarte mult de imaginația programatorului, în lipsa unei metodologii unitare de elaborare. Scopul programării structurate este elaborarea unor programe uşor de scris, de depanat şi de modificat.

Programarea structurată reprezintă o manieră de concepere a programelor conform unor reguli bine stabilite, utilizând un set redus de **tipuri de structuri de control**. **O structură de control** este caracterizată de o singură intrare şi ieşire şi reprezintă o combinație de operații folosită la scrierea algoritmilor.

Principiul programării structurate, formulat de Bohm şi Jacopini şi cunoscut ca "*teorema de structură*" precizează că orice algoritm având o singură intrare şi ieşire poate fi reprezentat printr-o combinație a 3 structuri de control:

1. **Secvența** - succesiune de două sau mai multe operații;
2. **Decizia** – alegerea unei variante din două alternative posibile;
3. **Ciclul cu test inițial** – repetarea unei operații cât timp o condiție este îndeplinită.

Programarea structurată admite şi utilizarea altor structuri de control, cum ar fi:

a) Ciclul cu test final*;*
b) Ciclul cu contor;

Fiecare din aceste structuri conțin instrucțiuni elementare organizate pe o logică unitară funcțională. În continuare sunt descrise structurile de control utilizate în programarea structurată, utilizând în paralel reprezentarea prin scheme logice şi limbaj pseudocod.

2.4.1 Secvența

Reprezintă o succesiune de instrucțiuni simple fig. 2.4. Din punct de vedere al execuției, logica acestei structuri constă în executarea secvențială a instrucțiunilor (una după alta), în ordinea scrierii lor. Fig. 2.10 prezintă succesiunea instrucțiunilor secvenței (care coincide şi cu ordinea de execuție) exprimată prin limbaj pseudocod ("a") respectiv schemă logică ("b"). Secvența va fi exemplificată prin algoritmul de calcul a ariei unui triunghi prin formula lui Heron.

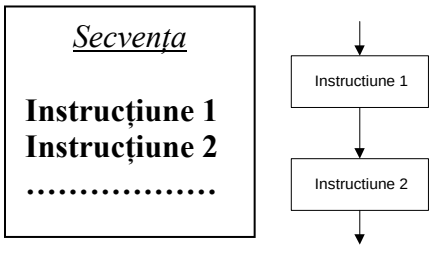


Fig. 2.10

E 2.7 Schemă logică şi program în limbaj pseudocod pentru algoritmul de calcul a ariei unui triunghi prin formula lui Heron

Aria unui triunghi S (data de ieşire) se calculează prin relația lui Heron : $S = \sqrt{p(p-a)(p-b)(p-c)}$ unde a, b, c sunt laturile triunghiului (date de intrare) iar $p = \frac{(a+b+c)}{2}$ este semiperimetrul triunghiului.

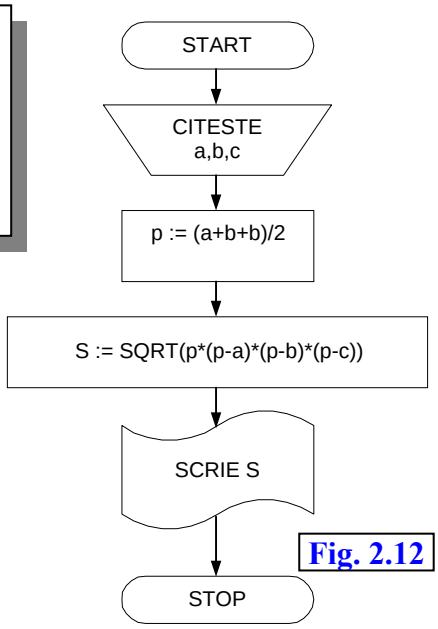


Fig. 2.12

```
PROGRAM Calcul arie triunghi
{Calculul ariei unui triunghi prin formula lui Heron}
ÎNCEPUT
        CITEŞTE (a,b,c)
        P := (a+b+c) / 2
        S := SQRT (p * (p-a) * (p-b) * (p-c))
        SCRIE (S)
SFÂRŞIT
```

Fig. 2.11

Secvența este constituită din 4 instrucțiuni: o instrucțiune de citire a variabilelor "a,b,c", două instrucțiuni de atribuire pentru calculul semiperimetrului P respectiv a ariei S şi o instrucțiune de afişare a ariei calculate anterior, care sunt executate în această ordine.

Din exemplu se observă obligativitatea plasării operatorului de înmulțire dintre factorii operației; "*subânțelegerea matematică*" a acestui operator nu se admite.

Echivalentul "*informatic*" al funcției radical este **SQRT**, iar expresia din care se extrage radicalul trebuie inclusă între paranteze rotunde.

2.4.2 Structura de decizie

Structura de decizie permite alegerea unei acțiuni din două alternative posibile, exprimată prin fig. 2.13 în reprezentarea prin limbaj pseudocod, respectiv prin fig. 2.14 pentru reprezentarea schemei logice. Execuția decurge astfel: testarea **condiției** şi execuția instrucțiunii **I1**, dacă condiția este adevărată respectiv execuția instrucțiunii **I2**, dacă condiția este falsă.

```
DACĂ condiție ATUNCI
        Instrucțiune I1
ALTFEL
        Instrucțiune I2
```

Fig. 2.13

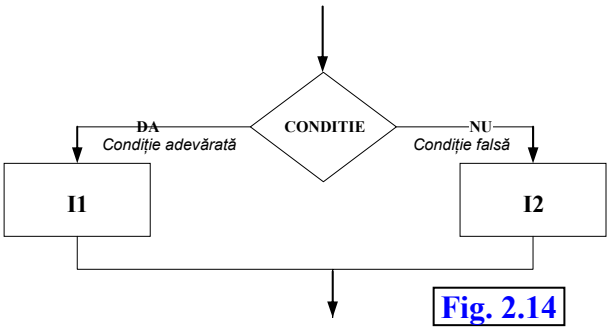


Fig. 2.14

După execuția unei singure instrucțiuni din cele două disponibile, algoritmul va continua cu prima instrucțiune imediat următoare acestei structuri.

Dacă pe oricare din cele două ramuri, algoritmul impune execuția mai multor instrucțiuni, *pentru reprezentarea în limbaj pseudocod*, acestea trebuie grupate într-o instrucțiune compusă, pentru a respecta sintaxa structurii (vezi pct. 5, & 2.3.2).

Pentru această structură este posibilă şi **decizia cu varianta unei căi nule**, exprimată prin fig. 2.15 în reprezentarea prin limbaj pseudocod, respectiv prin fig. 2.16 pentru reprezentarea schemei logice.

Execuția decurge astfel: testarea **condiției** şi execuția instrucțiunii **I1**, dacă condiția este adevărată; dacă condiția este falsă se continuă execuția programului.

DACĂ condiție **ATUNCI**
Instrucțiune I1

Fig. 2.15

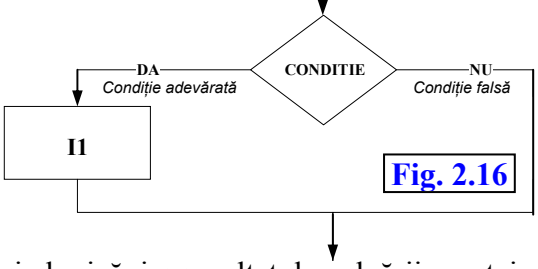


Fig. 2.16

În această structură **condiție** este o expresie logică, iar rezultatul evaluării acesteia este o valoare logică, concretizată prin una din valorile **adevărat** sau **fals**.

Expresia logică poate include mai multe condiții, care pot fi legate prin operatorii logici **AND** (ŞI logic) sau **OR** (SAU logic). Combinarea a două condiții prin operatorul **AND** generează valoarea logică **adevărat** numai dacă ambele condiții sunt adevărate. Combinarea a două condiții prin operatorul **OR** generează valoarea logică **adevărat** dacă cel puțin una din condiții este adevărată.

Structura de decizie va fi exemplificată prin algoritmul de rezolvare a ecuației de gradul 2.

E 2.8 Schemă logică şi program în limbaj pseudocod pentru algoritmul de rezolvare a ecuației de gradul 2

Forma generală a ecuației de gradul 2 este: $aX^2 + bX + C = 0$. Datele de intrare sunt constituite din coeficienții ecuației "a,b,c", iar datele de ieşire pot fi:

- Soluțiile ecuației, sub forma a 2 rădăcini X1, X2 (pentru gradul 2 al ecuației) respectiv rădăcina X (pentru gradul 1 al ecuației, generată pentru valoarea 0 a coeficientului "a");

- Unul din mesajele de informare "*Identitate*" / "*Imposibil*" corespunzătoare ecuației degenerate de gradul I sau "*Ecuația nu are rădăcini reale*" dacă determinantul ecuației este negativ, ceea ce ar corespunde rădăcinilor complexe.

Algoritmul de rezolvare a ecuației de gradul 2 este exprimat prin fig. 2.17 în reprezentarea prin limbaj pseudocod, respectiv prin fig. 2.18 pentru reprezentarea schemei logice. Logica algoritmului este următoarea:

- Dacă coeficientul "a" are o valoare diferită de 0 atunci se poate calcula valoarea determinantului **delta**, iar funcție de valoarea acestuia se pot calcula și afișa două rădăcini reale X1 și X2 sau se va afișa numai un mesaj de informare privind varianta rădăcinilor complexe, fără însă a le calcula;
- Dacă coeficientul "a" are o valoare egală 0, atunci ecuația degenerează într-o ecuație de grad I, al cărei algoritm de rezolvare corespunde celui detaliat în reprezentarea prin schemă logică în exemplul E 2.4, respectiv în exemplul E 2.6 pentru reprezentarea prin limbaj pseudocod.

Fig. 2.17

```
PROGRAM Rezolvare ec. grad 2
{ Rezolvarea ecuației de grad 2 }
ÎNCEPUT
    CITEŞTE (a, b,c)
    DACA a <> 0
        ÎNCEPUT
            Delta := b*b-4*a*c
            DACA delta < 0
                SCRIE ('Ecuatia nu are radacini reale')
            ALTFEL
                ÎNCEPUT
                    X1: = (-b-SQRT(delta))/2/a
                    X2: =(-b+SQRT(delta))/2/a
                    SCRIE (X1, X2)
                SFÂRŞIT
        SFÂRŞIT
    ALTFEL
        DACĂ b <> 0 ATUNCI
            ÎNCEPUT
                X: = -c/b
                SCRIE (X)
            SFÂRŞIT
        ALTFEL
            DACĂ c = 0 ATUNCI
                SCRIE ('Identitate')
            ALTFEL
                SCRIE ('Imposibil')
SFÂRŞIT
```

In fig. 2.17 sunt marcate grafic explicativ instrucțiunile compuse (**IC**), precum și ramificațiile structurilor de decizie (cu linie îngroșată), cu observația că marcatorii grafici au un rol informativ și nu fac parte din programul exprimat în limbaj pseudocod.

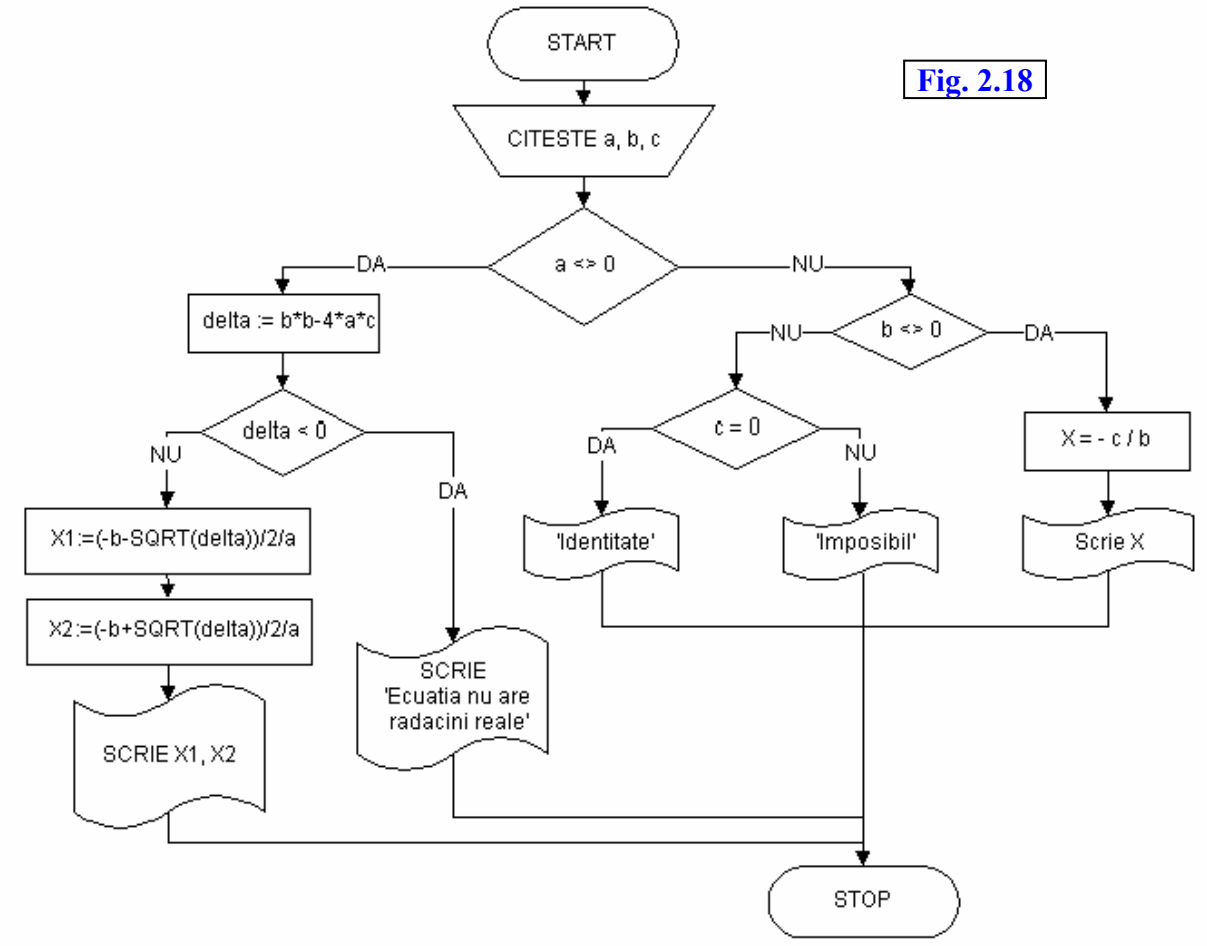


Fig. 2.18

2.4.3 Repetiția

Una din structurile cele mai des utilizate în cadrul algoritmilor este structura de repetiție, prin care se pot executa în mod repetat una sau mai multe instrucțiuni. Această structură este cunoscută şi sub numele de **ciclu**, **buclă** sau **iterație**.

Sunt posibile trei forme ale repetiției:

- **Ciclul cu test inițial** - unde numărul de repetiții este determinat pe baza unei condiții inițiale;
- **Ciclul cu test final** - unde numărul de repetiții este determinat pe baza unei condiții finale;
- **Ciclul cu contor** - unde numărul de repetiții este cunoscut apriori, evidența acestora fiind gestionată de o variabilă denumită **contor**.

Deşi prima formă este suficientă pentru a exprima structurat orice algoritm, în programarea structurată se admit şi celelalte forme din considerente legate de calculator.

Un alt considerent care trebuie menționat este cel referitor la forma programului care include o astfel de structură: astfel, principial, este posibilă şi varianta ca execuția repetitivă să rezulte din rescrierea instrucțiunilor repetitive de câte ori este impus prin algoritm, dar o asemenea posibilitate nu este şi practică. Este mult mai simplu scrierea o singură dată a

instrucțiunilor repetitive şi execuția repetată a lor. Este evident că în acest al doilea caz forma programului se reduce simțitor, cu toate efectele benefice rezultate din această reducere: consum redus de memorie, reducerea posibilității de generare de erori, depanarea mai simplă a programului, etc.

2.4.3.1 Ciclul cu test inițial

Această structură asigură execuția repetitivă a uneia sau mai multor instrucțiuni, numărul total de repetiții nefiind cunoscut la momentul elaborării algoritmului, ci rezultând – în timpul execuției algoritmului – pe baza unei condiții plasate înaintea instrucțiunilor repetitive, motiv pentru care această structură de repetiție se numeşte **ciclu cu test inițial**.

Reprezentarea structurii ciclului cu test inițial este exprimată prin fig. 2.19 în reprezentarea prin limbaj pseudocod, respectiv prin fig. 2.20 pentru reprezentarea schemei logice.

CÂT TIMP condiție **EXECUTĂ**
Instrucțiune I1

Fig. 2.19

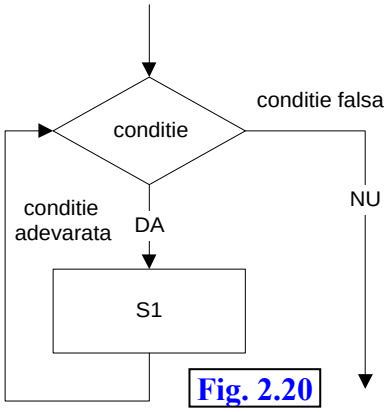


Fig. 2.20

Se observă faptul că în *reprezentarea prin limbaj pseudocod*, fig. 2.19, sintaxa instrucțiunii impune o singură instrucțiune **I1** pentru execuție repetată. Dacă însă algoritmul necesită mai multe instrucțiuni de executat repetitiv, atunci acestea trebuie grupate într-o instrucțiune compusă, pentru a respecta sintaxa structurii (vezi pct. 5, & 2.3.2).

Pentru reprezentarea în schemă logică, fig. 2.20, prin **S1** se poate înțelege o singură instrucțiune sau o secvență de instrucțiuni.

Execuția structurii decurge astfel: testarea **condiției** şi execuția instrucțiunii **I1**/secvenței **S1**, dacă **condiția** este adevărată; în caz contrar se încheie execuția ciclului.

Se observă caracterul repetitiv al execuției instrucțiunii **I1**/secvenței **S1**, atâta timp cât **condiția** este adevărată, numărul de repetiții este deci determinat de comutarea **condiției** din starea adevărat în starea fals. De aici rezultă următoarea concluzia logică, datorată şi poziționării anterioare a **condiției** în raport cu instrucțiunea **I1**/secvența **S1**: dacă **condiție** este de la bun început falsă, atunci instrucțiunea **I1**/secvența **S1** nu se execută nici măcar odată, numărul de repetiții fiind în acest caz 0.

Natura structurii impune necesitatea existenței uneia sau mai multor instrucțiuni, prin care, în timpul repetiției, să se intervină asupra **condiției**, pentru a comuta valoarea adevărată a acesteia într-o valoare falsă, în caz contrar repetiția se va executa la infinit din punct de vedere logic.

Structura de repetiție de tip ciclu cu test inițial va fi exemplificată prin algoritmul de calcul a radicalului unui număr printr-o relație de recurență.

E 2.9 Schemă logică şi limbaj pseudocod pentru algoritmul de calcul a radicalului unui număr prin relaţie de recurenţă – varianta ciclu cu test iniţial

Radicalul unui număr X se poate calcula prin următoarea relaţie de recurenţă:

$$Xcrt = 0.5 \cdot (Xant + X / Xant) \qquad [2.1]\ ,$$

unde vom înţelege prin: *Xcrt* – valoarea curentă a radicalului, calculată pe baza valorii sale anterioare *Xant*, prima valoare fiind impusă *Xant* = 1.

Algoritmul este exemplificat numeric în tabelul 2.1, pentru valoarea X=13. Se observă aplicarea repetitivă a relaţiei de recurenţă şi că la fiecare repetiţie (exceptând prima) valoarea curentă a variabilei *Xant* preia valoarea anterior calculată a variabilei *Xcrt*. Problema care se pune este care este criteriul de oprire al recurenţei. Vom admite că execuţia repetitivă va avea loc atâta timp cât diferenţa absolută între două valori calculate *Xcrt* - *Xant* este mai mare sau cel mult egală cu o precizie ε impusă. Altfel spus repetiţia ia sfârşit în momentul în care această diferenţă devine mai mică decât impus ε, iar ultima valoare *Xcrt* se admite ca fiind valoarea căutată a radicalului. Pentru exemplul numeric analizat valoarea aproximată a radicalului este 3.605551, pentru o precizie de calcul ε =0.001.

Tabel 2.1

Nr. Crt.	*Xant*	*Xcrt*
1	1.000000	7.000000
2	7.000000	4.428571
3	4.428571	3.682028
4	3.682028	3.606345
5	3.606345	3.605551

Algoritmul este exprimat prin fig. 2.21 în reprezentarea prin limbaj pseudocod, respectiv prin fig. 2.22 pentru reprezentarea schemei logice.

Fig. 2.21

```
PROGRAM Calcul radical
{ Calcul radical prin relaţie de recurenţă }
ÎNCEPUT
    CITEŞTE (X, EPS)
    Xant := 1
    Xcrt := 0.5 * (Xant + X / Xant)
    CÂT TIMP ABS(Xcrt - Xant) >= EPS EXECUTĂ
        ÎNCEPUT
IC          Xant := Xcrt
            Xcrt := 0.5 * (Xant + X/Xant)
        SFÂRŞIT
    SCRIE (Xcrt)
SFÂRŞIT
```

In figură este marcată grafic explicativ instrucţiunea compusă (**IC**), formată din 2 instrucţiuni de atribuire, care se execută repetitiv, cu observaţia că marcatorii grafici au un rol informativ şi nu fac parte din programul exprimat în limbaj pseudocod. Prima instrucţiune de atribuire **Xant:=Xcrt** programează preluarea valorii variabilei **Xcrt** de către variabila **Xant**, iar a doua instrucţiune aplică relaţia de recurenţă. Ambele instrucţiuni (fiind grupate prin instrucţiunea compusă) se execută repetitiv cât timp diferenţa **Xcrt-Xant** este mai mare decât valoarea EPS impusă.

Echivalentul "*informatic*" pentru valoare absolută (modul) este funcţia **ABS**, iar expresia asupra căreia se aplică modulul trebuie inclusă între paranteze rotunde.

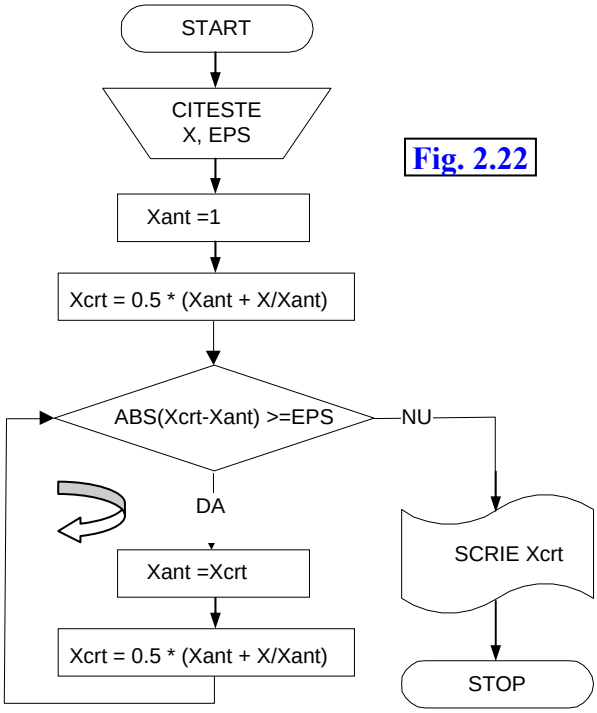


Fig. 2.22

Repetiția este evidențiată prin simbolul grafic ↻, care are un rol informativ şi care nu face parte din schema logică.

Se observă plasarea inițială a condiției, anterior instrucțiunilor repetitive **Xant=Xcrt** respectiv **Xcrt=0.5*(Xant+X/Xant)**.

Testarea condiției şi execuția instrucțiunilor repetitive are loc atâta timp cât condiția este adevărată. Când condiția devine falsă se iese din repetiție.

La fiecare repetiție se recalculează o nouă valoare pentru **Xcrt**, în sensul apropierii acesteia de valoarea corectă a radicalului numărului **X**, astfel încât la fiecare repetiție diferența dintre valoarea curentă **Xcrt** şi cea anterioară **Xant** scade.

La obținerea unei valori absolute a acestei diferența sub valoarea EPS impusă, calculul repetitiv se încheie. Deci, prin intermediul valorilor **Xcrt** şi respectiv **Xant** se intervine asupra condiției, creând astfel premizele comutării valorii adevărate a acesteia în valoarea falsă şi determinând deci finalizarea repetiției.

2.4.3.2 Ciclul cu test final

Această structură asigură execuția repetitivă a uneia sau mai multor instrucțiuni, numărul total de repetiții nefiind cunoscut la momentul elaborării algoritmului, ci rezultând – în timpul execuției algoritmului – pe baza unei condiții plasate ulterior instrucțiunilor repetitive, motiv pentru care această structură de repetiție se numeşte **ciclu cu test final**.

Reprezentarea structurii ciclului cu test final este exprimată prin fig. 2.23 în reprezentarea prin limbaj pseudocod, respectiv prin fig. 2.24 pentru reprezentarea schemei logice, unde prin **S1** se înțelege o secvență formată din una sau mai multe instrucțiuni.

Pentru *reprezentarea prin limbaj pseudocod*, fig. 2.23, deoarece grupul de instrucțiuni repetabile este marcat şi încadrat de cuvintele cheie **REPETĂ** respectiv **PÂNĂ CÂND**, obligativitatea formării unei instrucțiuni compuse a acestor instrucțiuni repetabile (prin încadrare între cuvintele

REPETĂ
Secvență S1
PÂNĂ CÂND condiție

Fig. 2.23

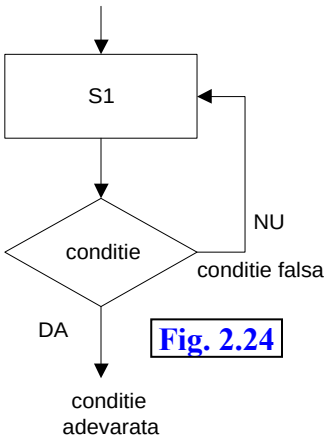


Fig. 2.24

cheie **ÎNCEPUT** respectiv **SFÂRŞIT**, vezi pct. 5, & 2.3.2) nu mai este impusă.

Execuția structurii decurge astfel: execuția secvenței de instrucțiuni **S1** urmat de testarea **condiției** şi reexecutarea secvenței **S1**, dacă **condiția** este falsă; în caz contrar (condiție adevărată) se încheie execuția ciclului. Se observă caracterul repetitiv al execuției secvenței **S1**, până când **condiția** devine adevărată, numărul de repetiții este deci determinat de comutarea **condiției** din starea fals în starea adevărat. Datorată poziționării ulterioare a **condiției** în raport cu secvența **S1**, secvența se execută cel puțin odată, aceasta fiind un motiv de a utiliza acest tip de repetiție în locul repetiției cu test inițial, & 2.4.3.1.

Natura structurii impune necesitatea existenței uneia sau mai multor instrucțiuni, prin care, în timpul repetiției, să se intervină asupra **condiției**, pentru a comuta valoarea adevărată a acesteia într-o valoare falsă, în caz contrar repetiția se va executa la infinit din punct de vedere logic.

Structura de repetiție de tip ciclu cu test final va fi exemplificată prin algoritmul de calcul a radicalului unui număr printr-o relație de recurență, pentru a se evidenția diferențele în raport cu varianta programării prin utilizarea structurii ciclu cu test inițial, & 2.4.3.1.

E 2.10 Schemă logică şi limbaj pseudocod pentru algoritmul de calcul a radicalului unui număr prin relație de recurență – varianta ciclu cu test final

Toate considerentele prezentate în exemplul E 2.9 rămân valabile, iar algoritmul este exprimat prin fig. 2.25 în reprezentarea prin limbaj pseudocod, respectiv prin fig. 2.26 pentru reprezentarea schemei logice. În continuare vor fi accentuate numai deosebirile de programare dintre cele două variante (exemplul E 2.9 şi exemplul E 2.10).

```
PROGRAM Calcul radical
{ Calcul radical prin relație de recurență }
ÎNCEPUT
    CITEŞTE (X, EPS)
    Xcrt := 1
    REPETĂ
        Xant := Xcrt
        Xcrt := 0.5 * (Xant + X/Xant)
    PÂNĂ CÂND ABS(Xcrt - Xant) < EPS
    SCRIE (Xcrt)
SFÂRŞIT
```

Fig. 2.25

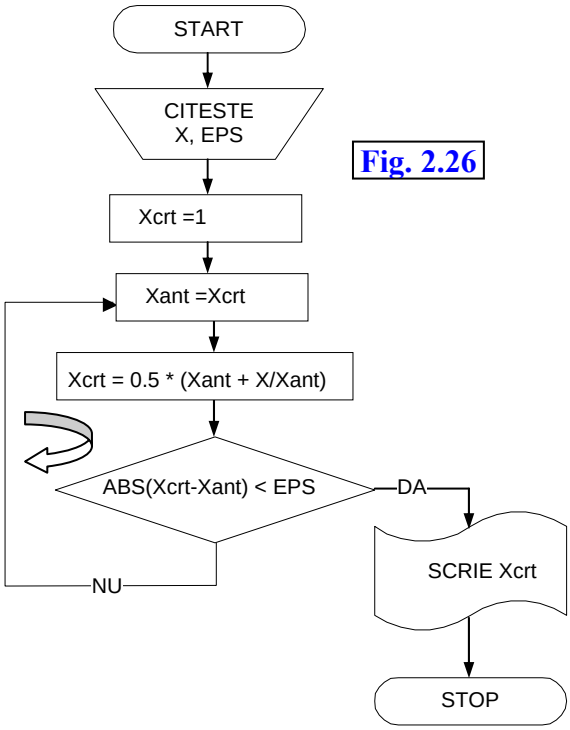


Fig. 2.26

În fig. 2.26 repetiția este evidențiată prin simbolul grafic ↻, care are un rol informativ şi care nu face parte din schema logică.

În exemplul 2.9, în care se utilizează ciclul cu test inițial, se impune calcularea (anterior repetiției) a primelor valori ale variabilelor **Xant** respectiv **Xcrt**, deoarece aceste variabile intervin în condiția repetiției **CÂT TIMP**.

În prezentul exemplu, anterior repetiției trebuie precizată numai valoarea de start a procesului, respectiv valoarea inițială egală cu 1 a variabilei **Xcrt**. În interiorul repetiției, se execută două instrucțiuni de atribuire: transferul către variabila **Xant** a valorii variabilei **Xcrt**, respectiv calcularea noii valori **Xcrt** funcție de valoarea anterioară **Xant**, conform relației de recurență. Se poate observa eliminarea necesității de încadrare a acestora într-o instrucțiune compusă, ele fiind încadrate între cuvintele cheie **REPETĂ** respectiv **PÂNĂ CÂND** specifice ciclului cu test final. Se remarcă de asemenea plasarea condiției ulterior secvenței repetitive formată din cele 2 instrucțiuni de atribuire, precum şi modificarea condiției, în sensul testării valorii diferenței mai mici decât a valorii preciziei impuse, adică exact condiția contrară celei din exemplul 2.9.

2.4.3.3 **Ciclul cu contor**

Această structură asigură execuția repetitivă a uneia sau mai multor instrucțiuni, numărul total de repetiții fiind cunoscut la momentul elaborării algoritmului, evidența repetițiilor fiind gestionată de o variabilă denumită **contor**, motiv pentru care această structură de repetiție se numeşte **ciclu cu contor**.

Variabila **CONTOR** are un rol bine precizat în această structură: contorizează numărul de execuții ale ciclului, iar valoarea variază în timpul execuției ciclului de la o valoarea inițială **VI**, la o valoarea finală **VF**, aceste valori fiind cunoscute înaintea începerii ciclului. Variația contorului se produce prin modificarea valorii acesteia cu o cantitate constantă, denumită **PAS,** după relația **CONTOR = CONTOR + PAS**. La depăşirea de către **CONTOR** a valorii finale **VF**, ciclul ia sfârşit. Deci finalizarea execuțiilor repetitive se produce ca o consecință a modificării valorilor acestei variabile.

Această structură este disponibilă în două forme:

- Forma **ascendentă** – în care **VI < VF**, valoarea variabilei **PAS** este pozitivă, contorul evoluează în sensul creşterii valorice, iar condiția de ieşire din ciclul este **CONTOR > VF**;
- Forma **descendentă** – în care **VI > VF**, valoarea variabilei **PAS** este negativă, contorul evoluează în sensul scăderii valorice, iar condiția de ieşire din ciclul este **CONTOR < VF**.

Numărul de repetiții **Nr** rezultă din relația: **Nr** = $|$ **VF – VI** $|$ + 1. Din această relație rezultă că ciclul se execută o singură dată dacă **VF = VI**. Dacă **VI > VF** pentru forma ascendentă, sau, pentru forma descendentă, dacă **VI < VF**, ciclul nu se execută niciodată.

Dacă valoarea variabilei **PAS** nu este specificată, valoarea sa este considerată a fi 1.

Reprezentarea structurii ciclului cu contor este exprimată prin fig. 2.27 în reprezentarea prin limbaj pseudocod, respectiv prin fig. 2.28 (forma ascendentă) pentru reprezentarea schemei logice, unde prin **S1** se înțelege o secvență formată din una sau mai multe instrucțiuni.

La fiecare parcurgere a ciclului se execută următoarele :

- se testează condiția de ieşire din ciclu; dacă condiția este adevărată se iese din ciclu;
- în caz contrar ciclul se continuă prin executarea instrucțiunii **I1**/secvenței **S1**;
- se modifică valoarea contorului cu **PAS**-ul ciclului;
- se reia ciclul prin testarea condiției de ieşire din ciclu.

PENTRU CONTOR := Vi, VF, PAS **EXECUTĂ**
Instrucțiune I1

Fig. 2.27

Pentru reprezentarea în limbajul pseudocod a acestei structuri se observă că sintaxa impune execuția repetitivă a unei unice instrucțiuni **I1**. Dacă însă algoritmul impune execuția, în același ciclu, a mai multor instrucțiuni, atunci acestea trebuie încadrate între cuvintele cheie **ÎNCEPUT** respectiv **SFÂRŞIT** formând astfel o instrucțiune compusă, pentru a respecta sintaxa structurii de ciclare (vezi pct. 5, & 2.3.2).

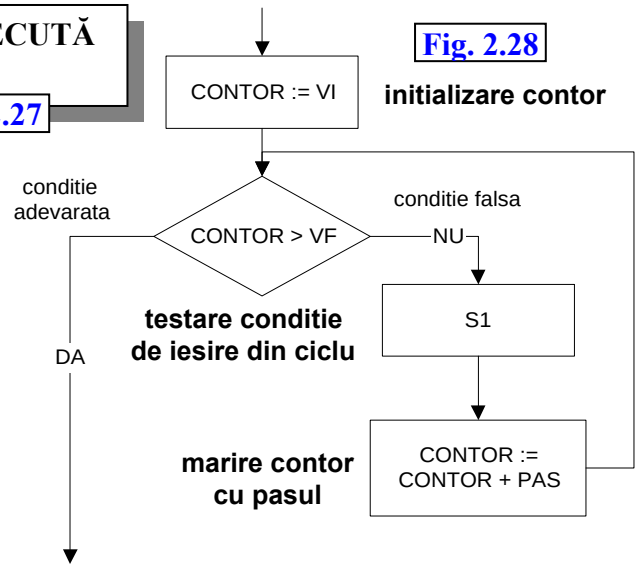


Fig. 2.28

Concluzionând, putem defini etapele unei structuri de ciclare astfel:

a) inițializare contorului (cu valoarea inițială);
b) testarea condiției de ieşire din ciclu (care variază funcție de forma ascendentă sau descendentă a ciclului); dacă condiția de ieşire se îndeplineşte se finalizează ciclul, în caz contrar se ciclul continuă;
c) execuția instrucțiunii sau secvenței de instrucțiuni repetabile;
d) modificarea valorii contorului cu pasul;
e) revenire la punctul **b)**.

După cum rezultă din fig. 2.27, pentru reprezentarea în limbajul pseudocod a acestei structuri, inițializarea contorului, testarea condiției de ieşire din ciclu şi modificarea valorii contorului cu pasul nu se scriu în mod explicit prin sintaxa instrucțiunii, dar se consideră că se execută automat în același mod ca şi la reprezentarea prin schemă logică. Altfel spus, logica instrucțiunii este identică, indiferent de modul concret de reprezentare a acesteia.

Pentru forma descendentă, reprezentarea prin schemă logică este similară, singura deosebire fiind condiția de ieşire din ciclu sub forma: **CONTOR < VI**.

În interiorul ciclului, pot fi executate repetitiv instrucțiuni valide, inclusiv alt ciclu. În această situație avem de-a face cu cicluri **îmbricate** sau **suprapuse**. Regula de suprapunere care trebuie respectată este ca domeniul ciclului interior să fie strict inclus în domeniul ciclului exterior. Regulă de funcționare a ciclurilor suprapuse este următoarea: ciclurile interioare se execută mai repede decât ciclurile exterioare, în sensul că, pentru fiecare repetiție a ciclului exterior, ciclul interior se parcurge complet şi trebuie finalizat înainte de a se produce o nouă repetiție a ciclului exterior.

Ciclul cu contor va fi exemplificat prin algoritmul de determinare a maximului dintre N numere citite succesiv, exprimat prin fig. 2.29 în reprezentarea prin limbaj pseudocod, respectiv prin fig. 2.30 pentru reprezentarea schemei logice.

E 2.11 Schemă logică şi limbaj pseudocod pentru algoritmul de calcul a maximului a N numere

Se consideră N numere reale, citite succesiv. Se cere determinarea maximului acestor numere. Numerele se vor citi succesiv în aceeaşi variabilă X, iar valoarea maximului se va regăsi în variabila MAXIM. Se va citi iniţial valoarea variabilei N şi primul dintre cele N numere, a cărei valoare se va atribui variabilei MAXIM. În continuare, de la al doilea la ultimul număr, *se vor executa repetitiv* următoarele două acţiuni: citirea unui nou număr şi compararea cu valoarea curentă a variabilei MAXIM; ca efect al comparării, dacă valoarea citită a numărului este mai mare decât valoarea curentă a variabilei MAXIM, atunci valoarea numărului se va atribui variabilei MAXIM. În final se va afişa valoarea variabilei MAXIM.

```
PROGRAM maximN
{ Maxim a N numere citite succesiv }
ÎNCEPUT
    CITEŞTE (N, X)
    MAXIM := X
    PENTRU I := 2,N EXECUTĂ
      ÎNCEPUT
        CITEŞTE X
        DACA X>MAXIM ATUNCI MAXIM := X
      SFÂRŞIT
    SCRIE (MAXIM)
SFÂRŞIT
```

IC

Fig. 2.29

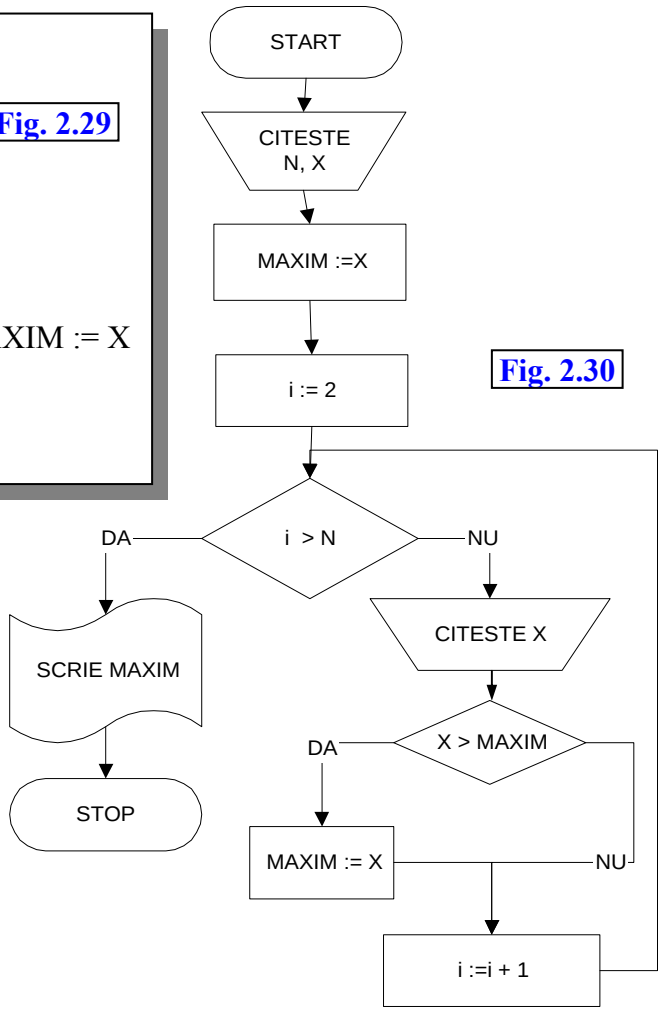


Fig. 2.30

Exemplificare numerică execuţie algoritm

N = 5 ; Numerele sunt X= 3,6,2,9,4

Iniţial X=3 MAXIM=3

Tabel 2.2

i	i > N	X	X >MAXIM	MAXIM
2	NU	6	DA	6
3	NU	2	NU	
4	NU	9	DA	9
5	NU	4	NU	
6	DA		Ieşire din ciclu	

MAXIM=9

Contorul ciclului este variabila " **i** ", care parcurge succesiv valorile cuprinse între valoarea iniţială 2 şi valoarea finală N, inclusiv. Instrucţiunile repetabile: citirea lui X şi compararea cu MAXIM se grupează prin instrucţiunea compusă **IC** (pct. 5, & 2.3.2).

2.5 EXEMPLE DE REPREZENTARE ALGORITMI PRIN SCHEME LOGICE ŞI LIMBAJ PSEUDOCOD

În acest paragraf vom exemplifica algoritmi, exprimaţi prin intermediul schemelor logice şi a limbajului pseudocod. Dacă pentru rezolvarea unui algoritm există mai multe posibilităţi de rezolvare, este recomandabil să se aleagă soluţia cea mai eficientă.

Calculatorul reprezintă numai un instrument de lucru, superior omului ca viteză de calcul, care însă nu are posibilitatea să poată lua decizii corespunzătoare în situaţii neprevăzute. Astfel, calculatorul rămâne subordonat programatorului. Din acest motiv, algoritmul de rezolvare trebuie să răspundă la toate variantele posibile, astfel încât programul elaborat pe baza acestuia să se afle în orice moment într-o stare bine determinată. De exemplu, la rezolvarea ecuaţiei de gradul I (exemplul E 2.2) soluţia poate fi obţinută numai dacă coeficientul "a" este diferit de 0, în caz contrar programatorul trebuie să prevadă o posibilitate în algoritm de avertizare a utilizatorului asupra cauzei lipsei soluţiei. În lipsa acestei prevederi, programul se poate finaliza cu o eroare sau cu rezultate imprevizibile.

De asemenea rezolvarea algoritmului trebuie să corespundă cazului general, chiar dacă pentru analiza şi elaborarea acestuia, se folosesc date particularizate. De exemplu, la rezolvarea ecuaţiei de gradul II (exemplul E 2.8), algoritmul va utiliza coeficienţi definiţi prin variabile şi nu prin valori numerice, generând astfel un program valabil oricare ar fi valorile acestora, urmând ca la execuţia programului să fie precizate valorile concrete.

2.5.1 Media aritmetică a trei numere

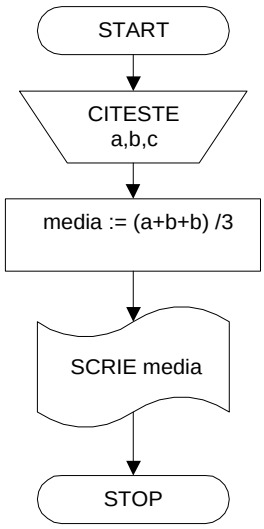
E 2.12 Schemă logică şi limbaj pseudocod pentru algoritmul de calcul a mediei aritmetice a trei numere

Se consideră 3 numere a, b, c (date de intrare). Se cere calculul mediei aritmetice a acestora M (dată de ieşire), definită matematic prin relaţia: $M = \frac{a+b+c}{3}$. Algoritmul este construit pe baza secvenţei (& 2.4.1) a 3 instrucţiuni: citirea datelor de intrare, instrucţiunea de atribuire pentru calculul mediei M şi afişarea acesteia.

Fig. 2.31

```
PROGRAM medie3
{Calculul mediei a trei numere}
ÎNCEPUT
        CITEŞTE (a,b,c)
        media := (a+b+c) / 2
        SCRIE (media)
SFÂRSIT
```

Fig. 2.32



2.5.2 Inversarea valorii a două variabile

E 2.13 Schemă logică şi limbaj pseudocod pentru algoritmul de inversare a valorilor a două variabile

Schimbarea valorii a două variabile a, b (date de intrare) între ele se face prin utilizarea unei variabile intermediare c. Datele de ieşire sunt aceleaşi variabile a, b, dar cu valori inversate. Pentru acest algoritm se poate face o analogie cu procedeul de inversare a conținutului a două vase, prin utilizarea unui vas suplimentar. Pentru înțelegerea corectă a acestui algoritm, revenim asupra instrucțiunii de atribuire (& 2.3.1, & 2.3.2), prin următoarea precizare: prin atribuire, valoarea calculată a expresiei sau a variabilei din partea dreaptă a semnului egal este transferată variabilei din stânga semnului egal. Rezultă următoarele concluzii:

- Acest transfer are deci un singur sens (de la dreapta la stânga);
- În stânga semnului egal se va afla o variabilă, iar în dreapta se poate afla o variabilă sau o expresie.

```
PROGRAM invers
{Inversarea valorii a două variabile între ele}
ÎNCEPUT
        CITEŞTE (a, b)
        c := a
        a := b
        b := c
        SCRIE (a, b)
SFÂRŞIT
```

Fig. 2.33

Deoarece într-o variabilă poate fi memorată o singură valoare, reintroducerea unei noi valori într-o variabilă provoacă pierderea valorii sale anterioare şi înlocuirea cu noua valoare (& 2.3).

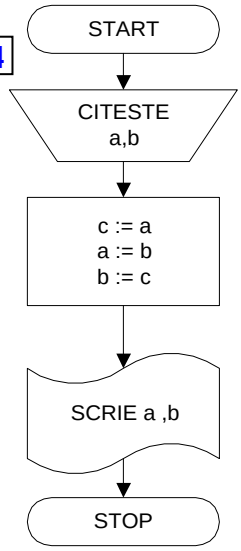


Fig. 2.34

Tabel 2.2

Exemplificare numerică execuție algoritm				**Analogie algoritm cu inversarea conținutului a două vase**			
Instrucțiune	**a**	**b**	**c**	**Vas a**	**Vas b**	**Vas c**	**Acțiune**
Valori inițiale	4	9		Apă	Ulei		Conținut inițial
c := a			4			Apă	Conținut vas “a” este transferat în vas “c”
a := b	9			Ulei			Conținut vas “b” este transferat în vas “a”, rămas liber
b := c		4			Apă		Conținut vas “c” este transferat în vas “b”, rămas liber
Valori finale: **a=9 ; b=4**				Conținut final: Vas “a” – Ulei ; Vas “b” – Apă			

2.5.3 Minim/maxim între două numere

E 2.14 Schemă logică şi limbaj pseudocod pentru algoritmul de determinare a minimului / maximului dintre două numere

Datele de intrare sunt două numere a, b. Pentru determinarea minimului (maximului) se compară valorile celor două numere, iar valoarea corespunzătoare se va depune in variabila MIN respectiv MAX, care reprezintă datele de ieşire. Variabilele MIN respectiv MAX vor dobândi deci cea mai mică respectiv mare valoare dintre "a" şi "b". Comparaţia valorilor se va efectua prin structura de decizie (& 2.4.2). Diferenţa dintre cei doi algoritmi este dată de condiţia specificată în structura de decizie.

```
PROGRAM Minim
{Minimul a doua numere}
ÎNCEPUT
        CITEŞTE (a, b)
        DACA a < b ATUNCI
                MIN := a
        ALTFEL
                MIN := b
        SCRIE (MIN)
SFÂRŞIT
```

Fig. 2.35

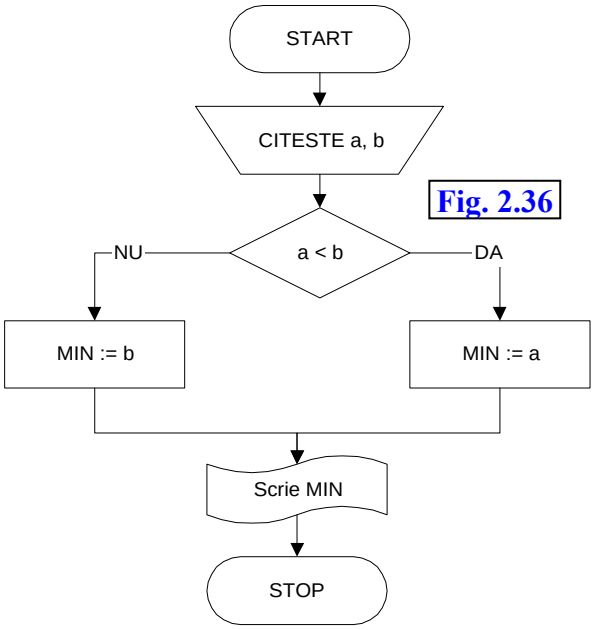


Fig. 2.36

```
PROGRAM Maxim
{Maximul a doua numere}
ÎNCEPUT
        CITEŞTE (a, b)
        DACA a > b ATUNCI
                MAX := a
        ALTFEL
                MAX := b
        SCRIE (MAX)
SFÂRŞIT
```

Fig. 2.37

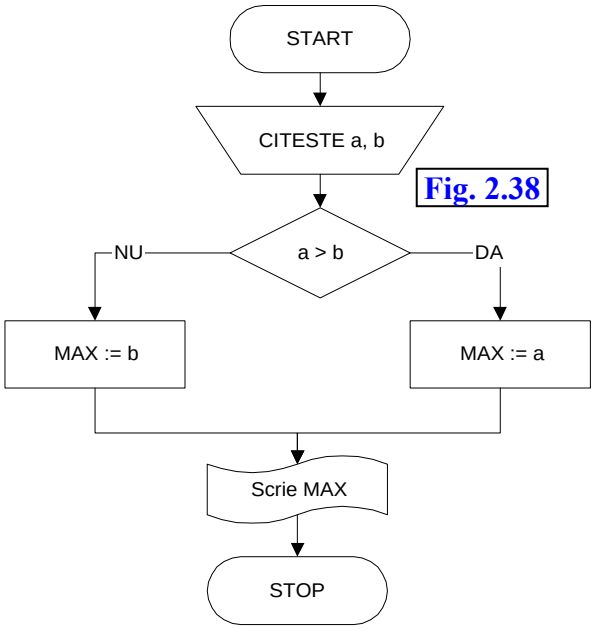


Fig. 2.38

2.5.4 Intersecția a două intervale

E 2.15 Schemă logică şi limbaj pseudocod pentru algoritmul de determinare a intersecției a două intervale

Se cere intersecția a 2 intervale [a, b] şi [c, d], date prin valorile reale a, b, c, d ∈ R, iar a<b şi c<d. Fie e - maximul limitei stânga: e = max(a, c) şi f – minimul limitei dreapta f = min(b, d). Vom avea două situații posibile (fig. 2.39):

- Dacă e > f => [a,b] ∩ [c,d] = Φ
- Daca e <= f => [a,b] ∩ [c,d] = [e,f].

Algoritmul face apel la algoritmul de determinare a maximului / minimului dintre 2 numere (& 2.5.3). Datele de intrare sunt limitele intervalelor (a, b, c, d), iar datele de ieşire pot fi limitele intersecției (e, f) sau un mesaj de informare asupra inexistenței intersecției.

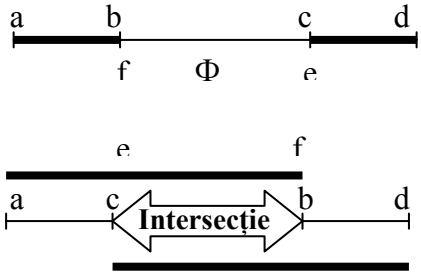


Fig. 2.39

```
PROGRAM Intersectia                          Fig. 2.40
{Intersecția a două intervale}
ÎNCEPUT
   CITEŞTE (a, b, c, d)
   DACA a > c ATUNCI
         e := a
   ALTFEL
         e := c
   DACA b < d ATUNCI
         f := d
   ALTFEL
         f := b
   DACA e < f ATUNCI
         SCRIE (e,f)
   ALTFEL
         SCRIE ('Mulțime vidă')
SFÂRŞIT
```

In fig. 2.40 sunt marcate grafic explicativ structurile de decizie (& 2.4.2), cu observația că marcatorii grafici au un rol informativ şi nu fac parte din programul exprimat în limbaj pseudocod.

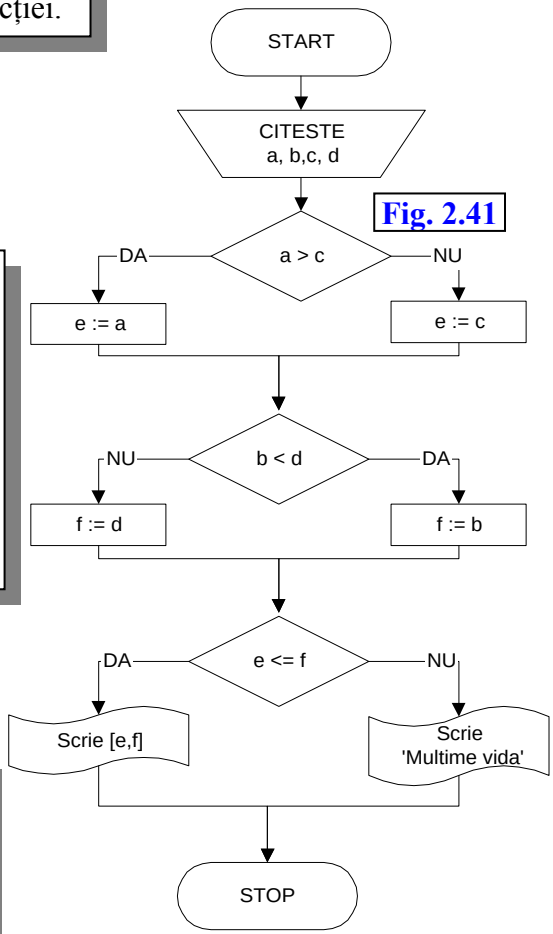


Fig. 2.41

Tabel cu valori de verificare algoritm

a	b	c	d	e	f	Intersecția
3	5	7	9	7	5	Φ
3	8	5	10	5	8	[5,8]
5	9	1	7	5	7	[5,7]
5	8	1	3	5	3	Φ

2.5.5 Minim între patru numere

E 2.16 Schemă logică şi limbaj pseudocod pentru algoritmul de calcul a minimului dintre patru numere

Varianta 1 (fig. 2.42 şi 2.43)**:** Se cere determinarea minimului dintre 4 numere a, b, c, d (date de intrare). Rezultatul va fi depus în variabila de ieşire MIN (data de ieşire).

Logica algoritmului este următoarea: se declară primul număr ca fiind minim, prin atribuirea valorii acestuia variabilei MIN, apoi se compară succesiv valorile următoarelor trei numere cu variabila MIN, iar dacă valoarea comparată este mai mică decât a variabilei MIN, valoarea numărului este preluată în această variabilă.

Fig. 2.42

```
PROGRAM Minim4
{Minim a 4 numere-varianta 1}
ÎNCEPUT
   CITEŞTE (a,b,c,d)
   MIN := a
   DACA b < MIN ATUNCI
         MIN := b
   DACA c < MIN ATUNCI
         MIN := c
   DACA d < MIN ATUNCI
         MIN := d
   SCRIE (MIN)
SFÂRŞIT
```

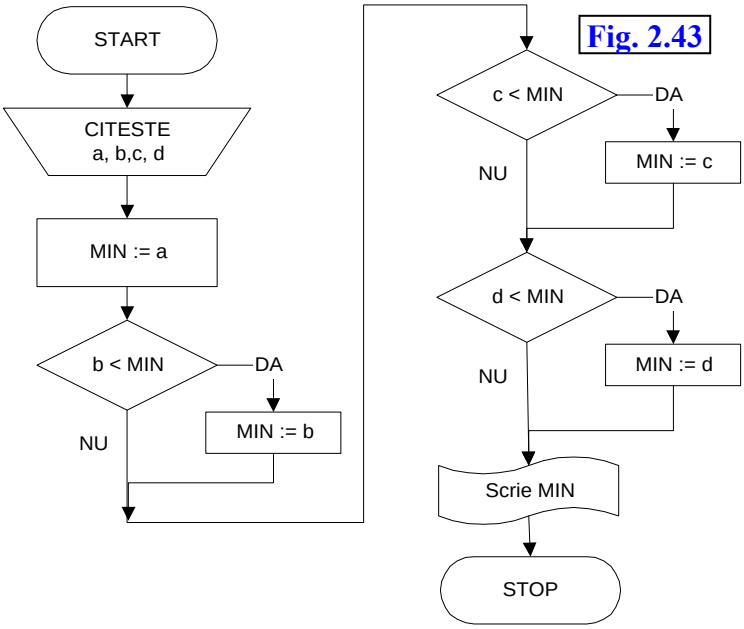


Fig. 2.43

Varianta 2 (fig. 2.44)**:** Logica algoritmului este următoarea: se determina minimul dintre "a" si "b" în variabila MIN1, iar minimul dintre "c" si "d" în variabila MIN2. Minimul absolut (MIN) se determină ca fiind minimul între MIN1 şi MIN2.

Fig. 2.44

```
PROGRAM Minim4
{Minim a 4 numere-varianta 2}
ÎNCEPUT
   CITEŞTE (a,b,c,d)
   DACA a < b ATUNCI
         MIN1 := a
   ALTFEL
         MIN1 := b
   DACA c < d ATUNCI
         MIN2 := c
   ALTFEL
         MIN2 := d
   DACA MIN1<MIN2 ATUNCI
         MIN := MIN1
   ALTFEL
         MIN := MIN2
SCRIE (MIN)
SFÂRŞIT
```

In fig. 2.42 şi 2.44 sunt marcate grafic explicativ structurile de decizie (& 2.4.2), cu observaţia că marcatorii grafici au un rol informativ şi nu fac parte din programul exprimat în limbaj pseudocod.

2.5.6 Suma / produs a două numere

E 2.17 Schemă logică şi limbaj pseudocod pentru algoritmul de calcul a sumei / produsului a două numere

Se dau trei numere reale X, Y. Dacă N <> 0 atunci să se calculeze suma numerelor X,Y, iar în caz contrar produsul lor. Datele de intrare pentru acest algoritm sunt deci numerele X, Y respectiv N, iar data de ieşire este o valoare numerică, ce reprezintă suma respectiv produsul numerelor X, Z.

```
PROGRAM suma produs
{Suma sau produsul lui X,Y }
ÎNCEPUT
        CITEŞTE (N, X, Y)
        DACA N <> 0 ATUNCI
            INCEPUT
   IC           S: = X+Y
                SCRIE (S)
            SFARSIT
        ALTFEL
            INCEPUT
   IC           P: = X*Y
                SCRIE (P)
            SFARSIT
SFÂRŞIT
```

Fig. 2.45

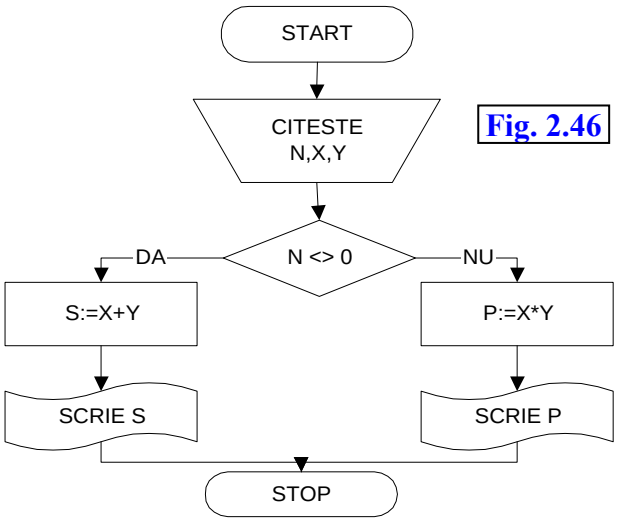


Fig. 2.46

In fig. 2.45 sunt marcate grafic explicativ instrucţiunile compuse IC (& 2.3.2) şi structura de decizie **DACA – ALTFEL** (& 2.4.2), cu observaţia că marcatorii grafici au un rol informativ şi nu fac parte din programul exprimat în limbaj pseudocod.

Algoritmul este didactic şi îşi propune să exemplifice modul de lucru cu instrucţiunea compusă IC (& 2.3.2), pentru formularea în limbaj pseudocod a unui algoritm. Pe ambele ramuri ale structurii de decizie, algoritmul impune execuţia a câte două instrucţiuni, respectiv calcularea, prin instrucţiunea de atribuire, a sumei sau produsului şi afişarea valorii calculate. Dar structura de decizie (& 2.4.2) impune, din punct de vedere sintactic, pentru fiecare din cele două ramuri, execuţia câte unei singure instrucţiuni. Această aparentă contradicţie, se rezolvă prin gruparea instrucţiunilor corespunzătoare fiecărei ramuri, formând astfel o instrucţiune compusă, care se consideră din punct de vedere sintactic ca şi cum ar fi o singură instrucţiune.

2.5.7 Rezolvare sistem de 2 ecuații cu 2 necunoscute

E 2.18 Schemă logică şi limbaj pseudocod pentru algoritmul de rezolvarea a unui sistem, cu 2 ecuații cu 2 necunoscute

Considerând un sistem de două ecuații cu 2 necunoscute, soluțiile x, y acestuia se obțin prin următoarele relații:

$$\begin{cases} ax+by=c \\ dx+ey=f \end{cases} \Rightarrow y=\frac{af-cd}{ae-bd} \quad x=\frac{ce-bf}{ae-bd} \qquad [2.2]$$

Datele de intrare sunt coeficienții a, b, d, e şi termenii liberi c, f, iar datele de ieşire sunt necunoscutele x, y. Soluția se poate calcula numai dacă numitorul N, exprimat prin relația : **N** = **a** x **e-b** x **d**, este diferit de 0, motiv pentru care această alternativă este conținută în algoritm. Pentru cazul contrar, rezultatul algoritmului se constituie într-un mesaj de informare.

```
PROGRAM sistem2
{Rezolvare sistem 2 ecuatii
cu 2 necunoscute}
ÎNCEPUT
        CITEŞTE (a,b,c,d,e,f)
        N := a*e-b*d
        DACA N <> 0 ATUNCI
            INCEPUT
                X:=(c*e-b*f)/N
                Y:=(a*f-c*d)/N
                SCRIE (X,Y)
            SFÂRŞIT
        ALTFEL
            SCRIE ('Sistemul nu are solutie')
SFÂRŞIT
```

IC

In fig. 2.47 este marcată grafic explicativ instrucțiunea compusă IC (& 2.3.2) şi structura de decizie **DACA – ALTFEL** (& 2.4.2), cu observația că marcatorii grafici au un rol informativ şi nu fac parte din programul exprimat în limbaj pseudocod.

Fig. 2.47

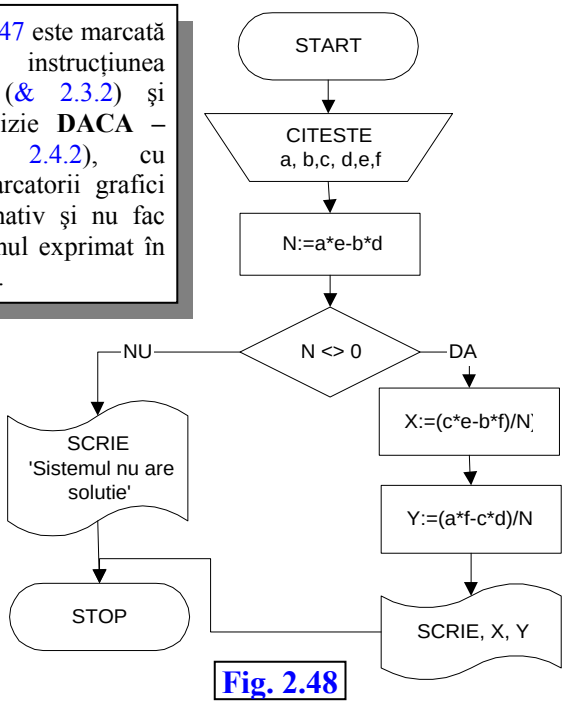


Fig. 2.48

Pe ramura **DACA** a structurii de decizie, algoritmul impune execuția a trei instrucțiuni, respectiv calcularea, prin instrucțiunea de atribuire, a necunoscutelor x, y şi afişarea valorilor calculate. Dar structura de decizie (& 2.4.2) impune sintactic, pentru fiecare din cele două ramuri, execuția câte unei singure instrucțiuni. Problema se rezolvă prin gruparea celor 3 instrucțiuni, formând astfel o instrucțiune compusă (& 2.3.2), care se consideră din punct de vedere sintactic ca şi cum ar fi o singură instrucțiune.

2.5.8 Afişare numere impare până la un număr N impus

E 2.19 Schemă logică şi limbaj pseudocod pentru algoritmul de afişare a numerelor impare până la un număr N impus

Să se scrie toate numerele impare până la un număr N impus. Numerele impare nu reprezintă date de intrare (deci nu vor fi introduse ca valori), ci vor fi generate pe baza următoarea relaţii: valoarea curenta a numărului impar rezultă din valoarea anterioară la care se adaugă 2, prima valoare din acest şir fiind 1. Singura dată de intrare pentru acest algoritm este numărul N până la care se doreşte afişarea numerelor impare. Datele de ieşire din algoritm sunt concretizate prin afişarea şirului de valori impare.

```
PROGRAM impare
{Afişare numere impare mai mici decât N}
ÎNCEPUT
        CITEŞTE (N)
        i := 1
A       DACA  i <= N ATUNCI
                ÎNCEPUT
                        SCRIE (i)
        IC              i: = i+2
                        TRECI LA PASUL A
                SFÂRŞIT
SFÂRŞIT
```

Fig. 2.49

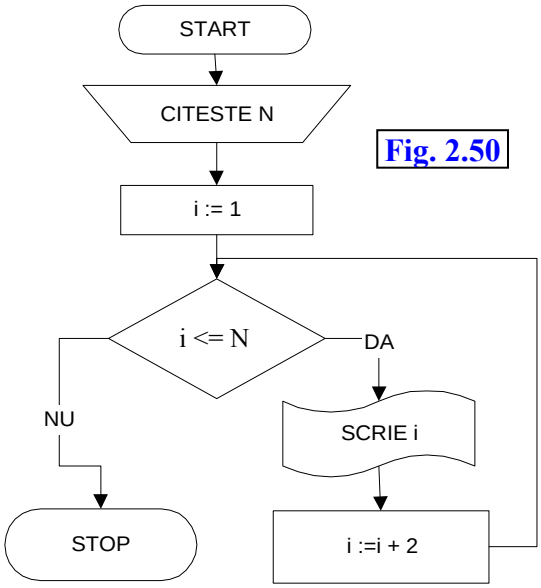


Fig. 2.50

In fig. 2.49 este marcată grafic explicativ instrucţiunea compusă IC (& 2.3.2) şi revenirea la structura de decizie **DACA – ATUNCI** (& 2.4.2), cu observaţia că marcatorii grafici au un rol informativ şi nu fac parte din programul exprimat în limbaj pseudocod.

Logica algoritmului este următoarea: după citirea valorii N, în variabila "i" se generează prima valoarea impară 1. Prin instrucţiunea de decizie, marcată prin eticheta "A" se verifică dacă valoarea curentă a variabilei "i" este inferioară sau cel mult egală cu valoarea N impusă, situaţie în care se execută 3 instrucţiuni grupate într-o instrucţiune compusă IC: afişarea valorii curenta a variabilei "i", generarea unei noi valori impare în variabila "i" prin suplimentarea cu 2 a valorii sale şi revenirea la instrucţiunea de decizie, creând astfel o repetiţie, a cărei execuţie încetează la o valoare a variabilei "i" care depăşeşte valoarea lui N. Se observă faptul că, în limbajul pseudocod, referirea la instrucţiunea de revenire se face prin intermediul etichetei "A", care precede această instrucţiune.

Varianta de execuție a algoritmului din fig. 2.49 nu respectă principiile de programare structurată, deoarece repetiția nu se execută printr-o structură admisă prin principiile programării structurate (& 2.4, & 2.4.3), ci printr-o instrucțiune de transfer necondiționat al execuției algoritmului spre altă instrucțiune (punctul e, & 2.3.2). În consecință, algoritmul va fi reformulat în două variante, pentru a respecta principiile programării structurate astfel ca repetiția să fie generată prin una din structurile admise: ciclul cu test inițial (& 2.4.3.1) în fig. 2.51, respectiv ciclul cu test final (& 2.4.3.2) în fig. 2.52.

PROGRAM impare1
{Afişare numere impare mai mici decât N}
ÎNCEPUT **Fig. 2.51**
    CITEŞTE (N)
    i := 1
    CÂT TIMP i <= N **EXECUTĂ**
        ÎNCEPUT
        IC **SCRIE** (i)
        i: = i+2
        SFÂRŞIT
SFÂRŞIT

PROGRAM impare2
{Afişare numere impare mai mici decât N}
ÎNCEPUT **Fig. 2.52**
    CITEŞTE (N)
    i := 1
    REPETĂ
        SCRIE (i)
        i: = i+2
    PÂNĂ CÂND i > N
SFÂRŞIT

În fig. 2.49, 2.51, 2.52 repetiția este evidențiată prin simbolul grafic, care are un rol informativ şi care nu face parte din reprezentarea algoritmului.

2.5.9 Suma a N numere. Algoritmul de sumare.

E 2.20 Schemă logică şi limbaj pseudocod pentru algoritmul de calcul a sumei a N numere

Se consideră N valori, ce se introduc succesiv. Se cere suma acestora. Datele de intrare pentru acest algoritm sunt:

- Numărul de valori N;
- Şirul celor N numere care vor fi însumate; toate cele N valori vor fi introduse în aceeaşi variabilă de memorie X, dar nu simultan ci succesiv.

Singura dată de ieşire din acest algoritm este suma valorilor, depozitată în variabila SUMA, apelată sub numele de variabilă de sumare.

Deoarece acest algoritm este foarte des folosit în interiorul altor algoritmi, în baza acestei aplicații, vom generaliza principiile algoritmului de sumare.

Modul clasic de sumare a mai multor valori numerice, se poate exprima matematic prin relația:

$$Suma = \sum_{i=1}^{N} X_i = X_1 + X_2 + X_3 + ... + X_{N-1} + X_N \qquad [2.3]$$

Datorită particularității modului de memorare a valorilor numerice (succesiv şi în aceeaşi variabilă) acest mod de calcul al sumei nu posibil de aplicat, valorile nefiind cunoscute simultan, ci pe rând (prin citire / calcul în aceeaşi variabilă) ceea ce provoacă pierderea valorii curente prin înlocuire cu noua valoare; această restricție impune deci un calcul succesiv al sumei, concomitent cu preluarea valorilor numerice în aceeaşi variabilă rezervată.

Vom exemplifica numeric operația de sumare, pentru a evidenția mai sugestiv particularitățile acesteia într-o formulare care să permită eficientizarea algoritmului şi generalizarea acestuia. Considerând N=5 valori numerice, concretizate prin următorul şir: 3, 6, 9, 2, 4 şi o valoare inițială nulă a variabilei de sumare, respectiv SUMA = 0, suma finală se poate calcula conform exemplificării din tabelul **2.3**, bazată pe proprietatea de asociativitate a adunării [rel. 2.4]:

$$Suma = (((((0 + X_1) + X_2) + X_3) + ... + X_{N-1}) + X_N) \qquad [2.4]$$

Tabel **2.3**

X	**SUMA**	**Observații**
-	0	Inițializarea sumei
3	**SUMA** = **SUMA** + **X** = 0 + 3 = 3	La valoarea curentă a variabilei **SUMA** se adaugă valoarea curentă a variabilei **X**
6	**SUMA** = **SUMA** + **X** = 3 + 6 = 9	
9	**SUMA** = **SUMA** + **X** = 9 + 9 = 18	
2	**SUMA** = **SUMA** + **X** = 18 + 2 = 20	
4	**SUMA** = **SUMA** + **X** = 20 + 4 = 24	

Se observă că procedeul aplicat în tabelul 2.3 respectă restricția existenței unei singure variabile **X**, în care sunt preluate *pe rând* valorile care trebuie însumate.

Din tabelul 2.3 rezultă următoarele concluzii specifice algoritmului de sumare:

- valoarea curentă a sumei se obține din valoarea sa anterioară la care se adaugă cantitatea de însumat **X**, conform relației generale de sumare:

  SUMA = SUMA + X [2.5]

  unde, "*contradicția matematică*" a relației [2.5] este rezolvată "*din punct de vedere algoritmic*" prin intermediul instrucțiunii de atribuire (punctul e, & 2.3.1);
- eficientizarea algoritmului impune utilizarea structurii de repetiție (& 2.4.3.3), aplicată următoarelor două instrucțiuni: generarea valorii numărului în variabila rezervată **X** (prin citire sau calcul) şi însumarea valorii citite la variabila de sumare **SUMA**, conform relației [2.5];
- aplicarea repetitivă a relației [2.5] impune precizarea prealabilă a primei valori a variabilei de sumare **SUMA**, ceea ce constituie inițializarea variabilei de sumare, efectuată anterior structurii de repetiție. În general, variabila de sumare se inițializează cu valoarea 0, dar, diverşi algoritmi pot impune şi alte valori.

În relația generală de sumare [2.5] locul variabilei simple **X** poate fi ocupat de o expresie, cu semnificația însumării valorii calculate a acesteia.

```
PROGRAM suma
{Suma a N numere de la tastatura}
ÎNCEPUT
        CITEŞTE (N)
        SUMA := 0
        PENTRU i := 1,N EXECUTĂ
            INCEPUT
IC          CITEŞTE (X)
            SUMA := SUMA + X
            SFÂRŞIT
        SCRIE (SUMA)
SFÂRŞIT
```

Fig. 2.53

Exemplificare numerică execuție algoritm

N = 5 Inițial SUMA=0

Numerele X sunt: 3,6,9,2,4

i	i > N	X	SUMA
1	NU	3	3
2	NU	6	9
3	NU	9	18
4	NU	2	20
5	NU	4	24
6	DA	Ieşire din ciclu	

SUMA=24

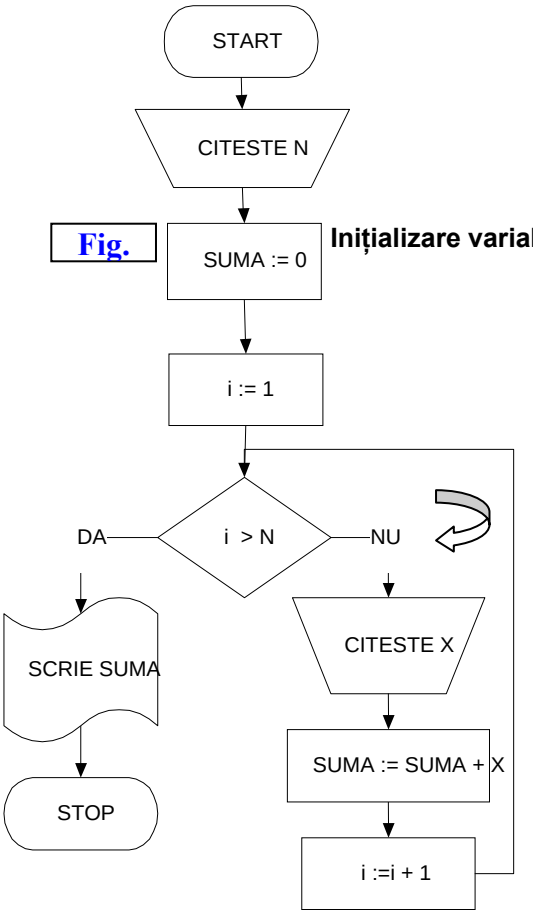


Fig. **Inițializare variabila de s**

In fig. 2.53 este marcată grafic explicativ instrucțiunea compusă IC (& 2.3.2), cu observația că marcatorii grafici au un rol informativ şi nu fac parte din programul exprimat în limbaj pseudocod.

În fig. 2.53, 2.54 repetiția este evidențiată prin simbolul grafic , care are un rol informativ şi nu face parte din reprezentarea algoritmului.

Cele două instrucțiuni: citirea numărului în variabila **X** şi aplicarea relației de sumare [2.5], se aplică repetitiv, de **N** ori, motiv pentru care acestea se grupează într-o instrucțiune compusă **IC** (& 2.3.2), care constituie instrucțiunea executivă a ciclului cu contor "i" (& 2.4.3.3).

2.5.10 Suma numerelor pozitive şi negative dintr-un şir de N numere

Se consideră un şir de N numere, ce se introduc succesiv. Se cere suma numerelor pozitive şi negative. Datele de intrare:

- Numărul de valori din şir N;
- Şirul celor N numere care vor fi însumate; toate cele N valori vor fi introduse în aceeaşi variabilă de memorie X, dar nu simultan ci succesiv.

Datele de ieşire sunt: suma valorilor pozitive SPOZ respectiv negative SNEG.

E 2.21 Schemă logică şi limbaj pseudocod pentru algoritmul de calcul a sumei numerelor pozitive şi negative dintr-un şir de N numere

```
PROGRAM sumapozneg
{Suma numerelor pozitive si negative
din N numere}
ÎNCEPUT
   CITEŞTE (N)
   SPOZ:=0
   SNEG:=0
   PENTRU i := 1,N EXECUTĂ
      INCEPUT
         CITEŞTE (X)
         DACA X>=0 ATUNCI
            SPOZ:=SPOZ+X
         ALTFEL
            SNEG:=SNEG+X
      SFÂRŞIT
   SCRIE (SPOZ)
   SCRIE (SNEG)
SFÂRŞIT
```

IC

Fig. 2.55

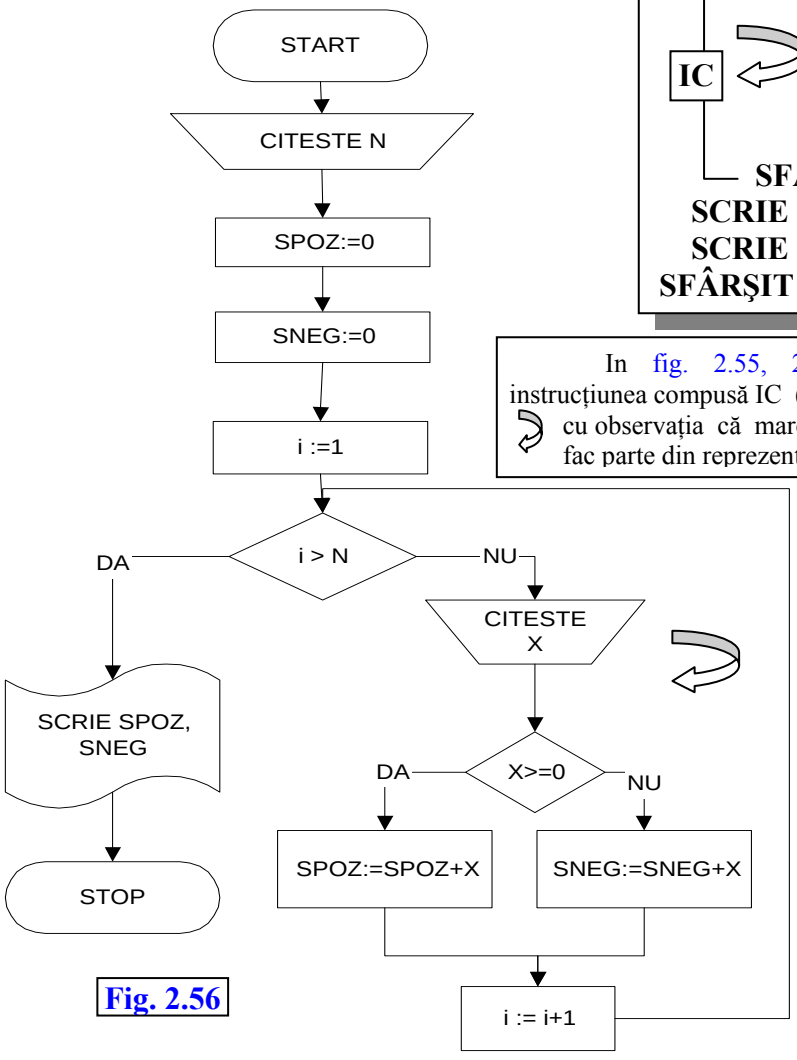


Fig. 2.56

In fig. 2.55, 2.56 este marcată grafic explicativ instrucţiunea compusă IC (& 2.3.2), iar repetiţia cu simbolul grafic cu observaţia că marcatorii grafici au un rol informativ şi nu fac parte din reprezentarea algoritmului.

Cele două instrucţiuni: citirea numărului în variabila **X** şi structura de decizie & 2.4.2 [2.5], se aplică repetitiv, de **N** ori, motiv pentru care acestea se grupează într-o instrucţiune compusă **IC** (& 2.3.2), care constituie instrucţiunea executivă a ciclului cu contor "i" (& 2.4.3.3).

Instrucţiunile executabile ale structurii de decizie aplică principiile algoritmului de sumare, & 2.5.9.

2.5.11 Produsul a N numere. Algoritmul produsului

E 2.22 Schemă logică şi limbaj pseudocod pentru algoritmul de calcul a produsului a N numere

Se consideră N valori, ce se introduc succesiv. Se cere produsul acestora. Datele de intrare pentru acest algoritm sunt:

- Numărul de valori din şir N;
- Şirul celor N numere care vor fi înmulţite; toate cele N valori vor fi introduse în aceeaşi variabilă de memorie X, dar nu simultan ci succesiv.

Singura dată de ieşire din acest algoritm este produsul valorilor, depozitată în variabila **P**, apelată sub numele de variabilă produs.

Deoarece acest algoritm este foarte des folosit în interiorul altor algoritmi, în baza acestei aplicaţii, vom generaliza principiile algoritmului produsului.

Modul clasic de calcul a produsului mai multor valori numerice, se poate exprima matematic prin relaţia:

$$P = \prod_{i=1}^{N} X_i = X_1 \cdot X_2 \cdot X_3 \cdot \ldots\ldots \cdot X_{N-1} \cdot X_N \qquad [2.6]$$

Datorită particularităţii modului de memorare a valorilor numerice (succesiv şi în aceeaşi variabilă) acest mod de calcul al produsului nu posibil de aplicat, valorile nefiind cunoscute simultan, ci pe rând (prin citire / calcul în aceeaşi variabilă) ceea ce provoacă pierderea valorii curente prin înlocuire cu noua valoare; această restricţie impune deci un calcul succesiv al produsului, concomitent cu preluarea valorilor numerice în aceeaşi variabilă rezervată.

Vom exemplifica numeric operaţia de calcul a produsului, pentru a evidenţia mai sugestiv particularităţile acesteia într-o formulare care să permită eficientizarea algoritmului şi generalizarea acestuia. Considerând N=5 valori numerice, concretizate prin următorul şir: 3,6,1,2,2 şi o valoare iniţială a variabilei produs **P = 1**, produsul final se poate calcula conform exemplificării din tabelul **2.4**:

Tabel **2.4**

X	Produs **P**	**Observaţii**
-	1	Iniţializare variabilă produs
3	**P** = **P** x **X** = 1 x 3 = 3	Valoarea curentă a variabilei produs **P** se înmulţeşte cu valoarea curentă a variabilei **X**
6	**P** = **P** x **X** = 3 x 6 = 18	
1	**P** = **P** x **X** = 18 x 1 = 18	
2	**P** = **P** x **X** = 18 x 2 = 36	
2	**P** = **P** x **X** = 36 x 2 = 72	

Se observă că procedeul aplicat în tabelul 2.3 respectă restricţia existenţei unei singure variabile **X**, în care sunt preluate *pe rând* valorile care trebuie înmulţite.

Din tabelul 2.4 rezultă următoarele concluzii specifice algoritmului produsului:

- valoarea curentă a produsului se obţine din valoarea sa anterioară înmulţită cu factorul de înmulţit **X**, conform relaţiei generale de calcul a produsului:

$$\mathbf{P = P \times X} \qquad [2.7]$$

  unde, "*contradicţia matematică*" a relaţiei [2.7] este rezolvată "*din punct de vedere algoritmic*" prin intermediul instrucţiunii de atribuire (punctul e, & 2.3.1);
- eficientizarea algoritmului impune utilizarea structurii de repetiţie (& 2.4.3.3), aplicată următoarelor două instrucţiuni: generarea valorii numărului în variabila rezervată **X** (prin citire sau calcul) şi înmulţirea valorii cu variabila produs **P**, conform relaţiei [2.7];
- aplicarea repetitivă a relaţiei [2.7] impune precizarea prealabilă a primei valori a variabilei produs **P**, ceea ce constituie iniţializarea variabilei produs, efectuată anterior structurii de repetiţie. În general, variabila produs se iniţializează cu valoarea 1, dar, diverşi algoritmi pot impune şi alte valori. Oricum, valoarea iniţială a variabilei produs nu poate fi nulă, deoarece, în acest caz, valoarea finală a produsului va fi nulă.

În relaţia generală de calcul a produsului [2.7] locul variabilei simple **X** poate fi ocupat de o expresie, cu semnificaţia înmulţirii valorii calculate a acesteia.

```
PROGRAM produs
{Produsul a N numere }
ÎNCEPUT
    CITEŞTE (N)
    P := 1
    PENTRU I := 1,N EXECUTĂ
        INCEPUT
IC      CITEŞTE X
        P:= P * X
        SFÂRŞIT
    SCRIE (P)
SFÂRŞIT
```

Fig. 2.57

In fig. 2.57, 2.58 este marcată grafic explicativ instrucţiunea compusă IC (& 2.3.2), iar repetiţia cu simbolul grafic cu observaţia că marcatorii grafici au un rol informativ şi nu fac parte din reprezentarea algoritmului.

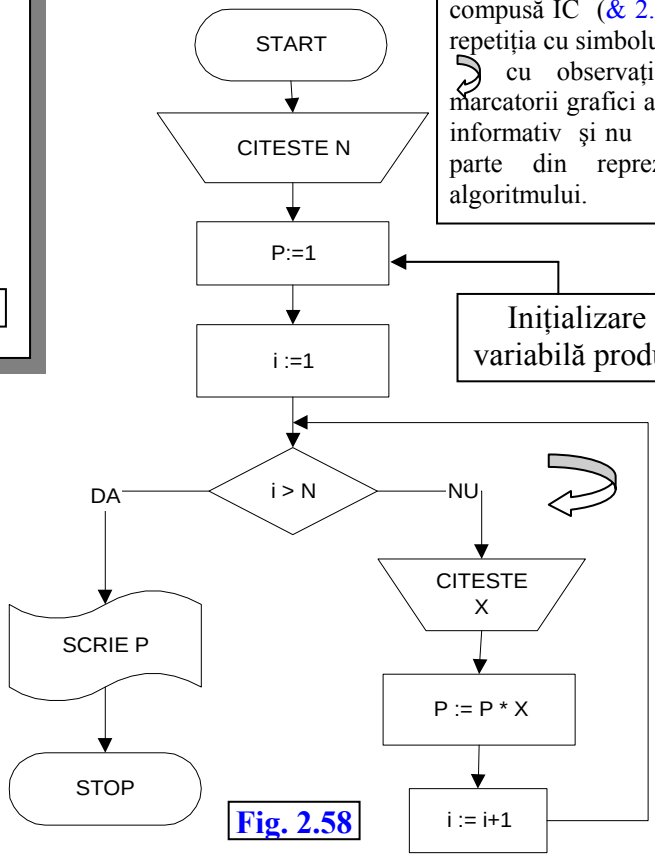


Fig. 2.58

Exemplificare numerică execuţie algoritm

N = 5 Iniţial P=1

Numerele sunt X= 3,6,1,2,2

i	i > N	X	P
1	NU	3	3
2	NU	6	18
3	NU	1	18
4	NU	2	36
5	NU	2	72
6	DA	Ieşire din ciclu	

P=72

Cele două instrucțiuni: citirea numărului în variabila **X** şi aplicarea relației de calcul a produsului [2.7], se aplică repetitiv, de **N** ori, motiv pentru care acestea se grupează într-o instrucțiune compusă **IC** (& 2.3.2)**,** care constituie instrucțiunea executivă a ciclului cu contor "i" (& 2.4.3.3).

2.5.12 Contorizare numere pozitive, negative şi nule dintr-un şir de N numere. Algoritmul contorizării.

E 2.23 Schemă logică şi limbaj pseudocod pentru algoritmul de contorizare a numerelor pozitive, negative şi nule dintr-un şir de N numere

Se consideră N valori, ce se introduc succesiv. Se cere să se determine câte dintre acestea sunt pozitive, câte sunt negative şi câte sunt nule. Datele de intrare pentru acest algoritm sunt:

- Numărul de valori din şir N;
- Şirul celor N numere care vor analizate; toate cele N valori vor fi introduse în aceeaşi variabilă de memorie X, dar nu simultan ci succesiv.

Datele de ieşire din acest algoritm sunt depozitate în trei variabile **POZ**, **NULE** şi **NEG**.

În cele ce urmează vom analiza un exemplu sugestiv, care simulează, din punct de vedere practic, problema teoretică propusă. Vom considera cazul extragerii a succesive a N bile dintr-o urnă, fiecare bilă având înscris un număr, a cărei valoare poate fi pozitivă, nulă sau negativă. Înaintea extragerilor vom rezerva trei căsuțe, corespunzătoare celor trei categorii de numere, căsuțe inițial goale, al căror scop este de a înmagazina bilele a căror semn / valoare corespunde categoriei. În continuare, procesul decurge repetitiv (pentru fiecare dintre cele N bile), prin efectuarea următoarelor operații:

- ✓ extragerea câte unei bile din urnă şi citirea valorii / semnului acesteia;
- ✓ testarea condiției pozitivității valorii bilei (valoare > 0):
  - ❑ dacă condiția este îndeplinită, bila se depozitează în căsuța de "*bile pozitive*";
  - ❑ în caz contrar, se testează condiția nulității valorii bilei (valoare = 0):
    - ➢ dacă condiția este îndeplinită, bila se depozitează în căsuța de "*bile nule*";
    - ➢ în caz contrar, bila se depozitează în căsuța de "*bile negative*".

La sfârşitul extragerilor, fiecare dintre cele trei căsuțe va avea înmagazinat un număr de bile corespunzător categoriei rezervate. Să accentuăm faptul că în acest proces nu suntem interesați de valoarea numerică asociată bilelor, ci de numărul de bile care verifică condițiile impuse. Privind problema din acest punct de vedere, vom evidenția următoarele:

a) depunerea unei bile într-o căsuță se realizează numai la îndeplinirea condiției asociate căsuței;

b) în concordanță cu scopul declarat al problemei, depunerea unei bile într-o căsuță provoacă creşterea cu o unitate a numărului bilelor din căsuță.

Prin analogie cu exemplul analizat, vom sintetiza principiile algoritmului contorizării:

- Scopul operației de contorizare este înregistrarea frecvenței de îndeplinire a unei condiții.
- Vom înțelege prin **contor** variabila **C** care înregistrează frecvența de îndeplinire a unei condiții, prin mărire cu o cantitate specificată **X,** conform relației de contorizare:

$$\mathbf{C = C + X} \qquad [2.8]$$

  În general, cantitatea specificată **X** are valoarea 1, fără a exclude însă posibilitatea ca, pentru diverşi algoritmi, aceasta să fie totuşi diferită de 1. Se observă similitudinea acestei relații cu relația generală de sumare [2.5], contorizarea putând fi privită ca un caz particular al algoritmului de sumare, în care cantitatea de însumat este unitară. "*Contradicția matematică*" a relației [2.8] este rezolvată "*din punct de vedere algoritmic*" prin intermediul instrucțiunii de atribuire (punctul e, & 2.3.1);
- eficientizarea algoritmului impune utilizarea structurii de repetiție (& 2.4.3.3), în care să fie inclusă testarea condiției impuse şi creşterea contorului la îndeplinirea condiției, conform relației [2.8];
- aplicarea repetitivă a relației [2.8] impune precizarea prealabilă a valorii inițiale a contorului **C**, ceea ce constituie inițializarea contorului, efectuată anterior structurii de repetiție. În general, variabila contor se inițializează cu valoarea 0 (similar variabilei de sumare - & 2.5.9), dar, diverşi algoritmi, pot impune şi alte valori.

```
PROGRAM contorizare
{Contorizare numere pozitive, negative şi nule
dintr-un şir de N numere}
ÎNCEPUT
    CITEŞTE (N)
    POZ := 0
    NULE := 0
    NEG := 0
    PENTRU I := 1,N EXECUTĂ
      INCEPUT
          CITEŞTE (X)
          DACĂ X > 0 ATUNCI
              POZ := POZ+1
          ALTFEL
              DACĂ X = 0 ATUNCI
                  NULE := NULE +1
              ALTFEL
                  NEG := NEG+1
      SFÂRŞIT
    SCRIE (POZ, NULE, NEG)
SFÂRŞIT
```

IC

In fig. 2.59, 2.60 este marcată grafic explicativ instrucțiunea compusă IC (& 2.3.2), iar repetiția cu simbolul grafic ↻ cu observația că marcatorii grafici au un rol informativ şi nu fac parte din reprezentarea algoritmului.

Fig. 2.59

Contorizarea celor N numere pozitive, nule şi negative se va face prin intermediul a trei variabile de tip contor: POZ, NULE, NEG, care se inițializează cu valoarea 0. Se citesc pe rând numerele şi se măreşte cu 1 unul din contorii POZ, NUL, NEG, funcție de valoarea pozitivă, nulă sau negativă a numărului introdus.

Exemplificare numerică execuție algoritm

N = 5 Numerele sunt X= 3,-6,0,2,4

Inițial: POZ=0, NULE=0, NEG=0

i	i > N	X	POZ	NULE	NEG
1	NU	3	1		
2	NU	-6			1
3	NU	0		1	
4	NU	2	2		
5	NU	4	3		

POZ=3, NULE=1, NEG=1

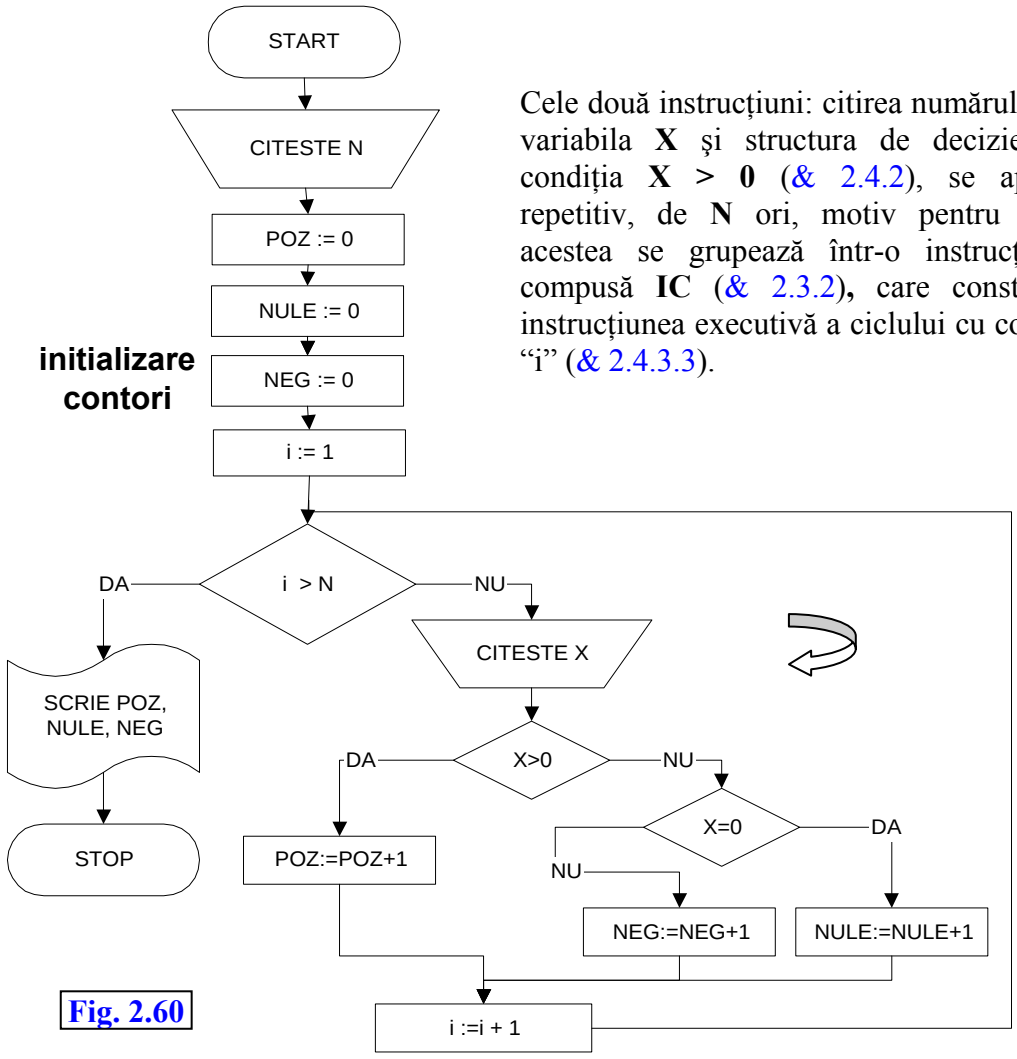


Fig. 2.60

Cele două instrucţiuni: citirea numărului în variabila **X** şi structura de decizie cu condiţia **X > 0** (& 2.4.2), se aplică repetitiv, de **N** ori, motiv pentru care acestea se grupează într-o instrucţiune compusă **IC** (& 2.3.2)**,** care constituie instrucţiunea executivă a ciclului cu contor "i" (& 2.4.3.3).

Structura de decizie cu condiţia **X > 0** include ca instrucţiuni executive:

- pe ramura **DA**, aplicarea relaţiei de contorizare [2.8] pentru contorul **POZ**;
- pe ramura **NU**, o altă structură de decizie, cu condiţia **X = 0**, care, la rândul ei, include ca instrucţiuni executive:
  - pe ramura **DA**, aplicarea relaţiei de contorizare [2.8] pentru contorul **NULE**;
  - pe ramura **NU**, aplicarea relaţiei de contorizare [2.8] pentru contorul **NEG**.

Se observă că algoritmul lucrează cu o singură variabilă **X**, în care sunt preluate *pe rând*, prin citire, valorile care trebuie analizate, ceea ce, pentru exemplul practic descris anterior, corespunde cu extragerea succesivă şi nu simultană a bilelor. Datorită particularităţii modului de memorare a valorilor numerice (succesiv şi în aceeaşi variabilă) se produce efectul pierderii valorii curente prin înlocuire cu noua valoare, restricţie care impune deci efectuarea contorizării concomitent cu preluarea valorilor numerice în aceeaşi variabilă rezervată.

2.5.13 Sumare serie

E 2.24 Schemă logică şi limbaj pseudocod pentru algoritmul de contorizare a numerelor pozitive, negative şi nule dintr-un şir de N numere

Se cere să se calculeze suma seriei:

$$S = 1 + \sum_{i=1}^{N} \frac{i}{\prod_{j=1}^{i+1} j} = 1 + \underbrace{\frac{1}{1\cdot 2} + \frac{2}{1\cdot 2\cdot 3} + \ldots + \frac{i}{1\cdot 2\cdot 3\cdot\ldots\cdot(i+1)} + \ldots + \frac{N}{1\cdot 2\cdot 3\cdot\ldots\cdot(N+1)}}_{\text{i = 1…N}} \qquad [2.9]$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{j =1…(i+1)}}$$

Singura dată de intrare pentru acest algoritm este numărul de termeni din serie N, până la care se impune însumarea, iar data de ieşire este suma calculată a seriei. Valorile numerice care se doresc a se însuma se vor genera folosind proprietăţile structurii repetiţiei cu contor (& 2.4.3.3).

Din analiza relaţiei [2.9] se observă următoarele:

- numărătorul termenilor seriei, exceptând primul, parcurg valorile cuprinse între 1 şi N, valori care pot fi generate printr-o structură de repetiţie, de contor "i";
- numitorii termenilor seriei, exceptând primul, reprezintă produsul primelor i+1 numere naturale, numere care pot fi generate printr-o structură de repetiţie, de contor "j", simultan cu calculul produsului, conform algoritmului produsului (& 2.5.11), prin aplicarea relaţiei [2.7];
- suma seriei se obţine prin însumarea termenilor fracţionali, conform algoritmului de sumare (& 2.5.9), prin aplicarea relaţiei [2.5], unde cantitatea de însumat reprezintă raportul între valorile generate prin contorului "i" şi produsul calculat în ciclul de contor "j".

Algoritmul analizat îmbină deci algoritmul de sumare şi algoritmul produsului, incluzând calculul acestora în structuri de repetiţie (& 2.4.3.3). Reprezentarea în limbaj pseudocod a algoritmului se prezintă în fig. 2.61, iar reprezentarea prin schemă logică în fig. 2.62, în baza cărora se pot formula următoarele observaţii:

- iniţializarea variabilei de sumare **SUMA** se face, în acest caz particular, cu valoarea 1, pentru a include în suma finală şi primul termen al seriei (de valoare 1), care nu este generat conform logicii matematice a termenilor fracţionali;
- produsul se calculează în ciclul de contor "j", care conţine o singură instrucţiune executivă şi anume aplicarea relaţiei [2.7]; deoarece produsul se recalculează pentru fiecare termen al seriei, se impune reiniţializarea variabilei produs **PROD** cu valoarea 1;
- cele trei instrucţiuni: iniţializarea variabilei produs, execuţia ciclului cu contor "j" şi aplicarea relaţiei de însumare [2.5], se aplică repetitiv, de **N** ori, motiv pentru care acestea se grupează într-o instrucţiune compusă **IC** (& 2.3.2)**,** care constituie instrucţiunea executivă a ciclului cu contor "i" (& 2.4.3.3).
- ciclul de contor "j" este interior ciclului exterior de contor "i" şi deci, pentru fiecare repetiţie a ciclului exterior, ciclul interior se parcurge complet şi trebuie finalizat înainte de a se produce o nouă repetiţie a ciclului exterior.

```
PROGRAM sumaserie
{Suma unui sir }
ÎNCEPUT
        CITEŞTE (N)
        SUMA :=1
        PENTRU  i := 1,N EXECUTA
            INCEPUT
IC              PROD := 1
                PENTRU j := 1,i+1 EXECUTA PROD := PROD*j
                SUMA := SUMA+i / PROD
            SFÂRŞIT
        SCRIE (SUMA)
SFÂRŞIT
```

Fig. 2.61

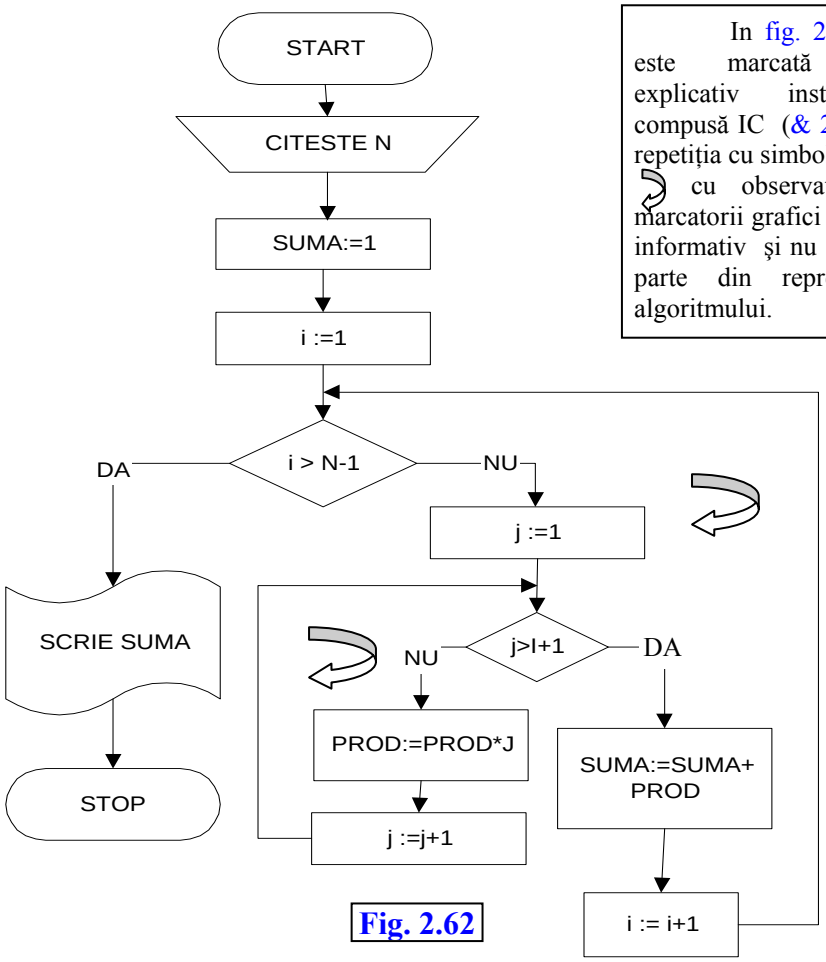


Fig. 2.62

In fig. 2.61, 2.62 este marcată grafic explicativ instrucțiunea compusă IC (& 2.3.2), iar repetiția cu simbolul grafic cu observația că marcatorii grafici au un rol informativ şi nu fac parte din reprezentarea algoritmului.

3. MEDIUL DE PROGRAMARE TurboPascal

3.1 GENERALITĂŢI

Limbajul Pascal a fost elaborat de Niklaus WIRTH, profesor la Universitatea tehnică din Zurich – Elveţia (1970). Deşi limbajul a fost proiectat iniţial pentru scopuri didactice, acest limbaj s-a transformat într-un puternic instrument pentru rezolvarea problemelor inginereşti şi ştiinţifice.

Popularitatea limbajului se datorează firmei americane Borland, care a elaborat un compilator performant pentru calculatoare personale, inclus într-un mediu de programare avansat, care mai conţine un editor profesional de texte şi un asamblor integrat. Elementele limbajului au fost îmbogăţite substanţial, iar ceea ce a rezultat a fost denumit **TurboPascal**. Caracteristica **Turbo** se referă la rapiditatea compilatorului.

Caracteristicile limbajului sunt:

- este puternic tipizat, oferind o gamă variată de tipuri de date predefinite şi posibilitatea definirii tipurilor utilizator;
- este modularizat;
- oferă un număr mare de proceduri şi funcţii predefinite;
- posibilitatea separării unui program complex în module independente, care pot fi programate şi compilate separat.

Intrarea în mediul de programare TurboPascal se realizează prin lansarea în execuţie a fişierului **turbo.exe** din directorul de instalare al limbajului. Dacă se lucrează sub sistemul de operare Windows, intrarea în mediul de programare TurboPascal se realizează prin dublu click stânga pe icoana versiunii corespunzătoare programului (fig. 3.1).

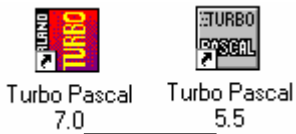


Fig. 3.1

3.2 INTERFAŢA MEDIULUI DE PROGRAMARE TurboPascal

Ne vom referi în continuare la versiunea TurboPascal 7.0, dar multe dintre elemente sunt valabile şi pentru versiuni anterioare. După lansarea în execuţie a mediul de programare TurboPascal se generează interfaţa prezentată în fig. 3.2, care este divizată în 3 zone:

- **linia meniului (linia superioară)** – meniul TurboPascal permite un dialog interactiv cu utilizatorul, în sensul execuţiei imediate a comenzilor asociate opţiunilor de meniu selectate sau a activării unor ferestre de dialog, prin care utilizatorul poate specifica opţiuni suplimentare corespunzătoare unor acţiuni specializate.
- **zona de lucru** – partea din mijloc a ecranului este destinată afişării simultane a mai multor ferestre de editare sau ale sistemului: ferestre de dialog, de afişare a rezultatelor, de urmărire a valorilor, etc. Zona de lucru poate fi ocupată concomitent de mai multe ferestre.
- **linia de stare (linia inferioară)** – afişează informaţii referitoare la :
  - descriere pe scurt a funcţiei meniului sau comenzii selectate;
  - afişează destinaţie taste funcţionale;
  - prezentare succintă a destinaţiei câmpurilor de dialog.

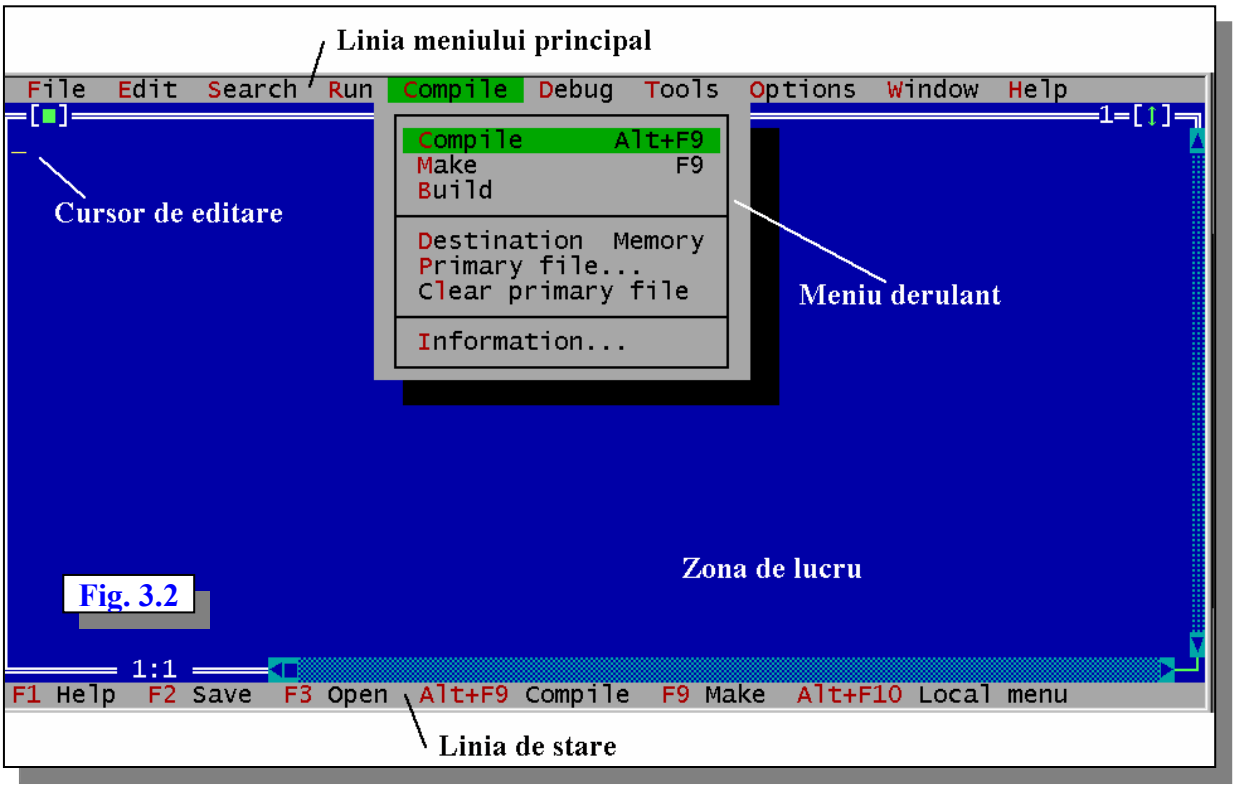


Fig. 3.2

3.3 STRUCTURA FERESTRELOR

Prin **fereastră** se înțelege o zonă dreptunghiulară încadrată a ecranului, ce poate fi deplasată şi redimensionată.

La un moment dat pot fi deschise mai multe ferestre simultan, dar numai una este activă şi anume cea în care se lucrează în mod curent. Comenzile se aplică doar ferestrei active. Fereastra activă are chenarul dublu, iar cea inactivă are chenarul simplu. Activarea unei ferestre se poate face prin parcurgere succesiv/circulară prin tasta **F6** sau prin click stânga mouse în fereastra dorită, dacă aceasta este vizibilă.

Forma generala a unei ferestre este dată în fig. 3.3, iar elementele acesteia sunt următoarele:

- **Simbolul de închidere al ferestrei** – sub forma unui pătrățel încadrat între paranteze drepte, care permite închiderea ferestrei prin click cu butonul stâng mouse. Aceeaşi acțiune se obține şi prin combinația de taste **Alt + F3** sau prin opțiunea **Close** din meniul **Window**.
- **Numele ferestrei** – reprezintă numele fişierului sub care este salvat programul sau indică funcția ferestrei. Poziționarea cursorului mouse pe linia superioară a ferestrei şi menținerea apăsată a butonului stâng mouse concomitent cu mişcarea acestuia provoacă deplasarea ferestrei şi repoziționarea ei la eliberarea butonului mouse. Aceeaşi acțiune se lansează şi prin combinația de taste **Ctrl + F5** sau selecția opțiunii **Size/Move** din meniul **Window**, urmând ca repoziționarea ferestrei să se execute prin tastele direcționale, iar fixarea finală a poziției prin apăsarea tastei **Enter**. În timpul repoziționării (indiferent dacă operația se execută cu

mouse-ul sau cu tastatura) chenarul ferestrei este marcat prin linie simplă, iar, la fixarea finală a poziției, chenarul redevine dublu.

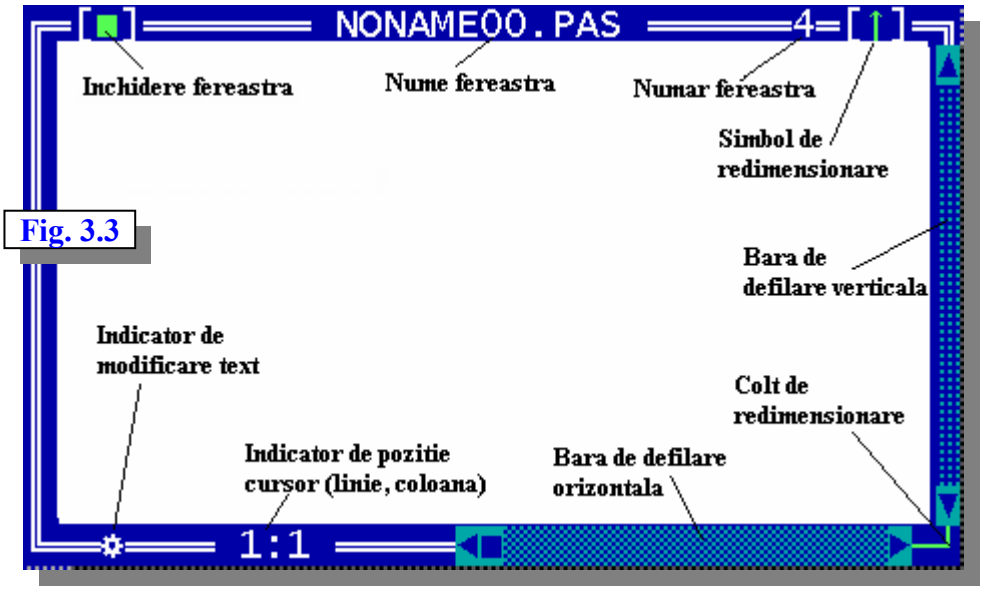


Fig. 3.3

- **Numărul ferestrei** – doar primele 9 ferestre vor avea un număr ataşat, generat automat la deschiderea acesteia; activarea unei ferestre se poate realiza prin combinația de taste **Alt + număr**, unde cifrele numerice se vor lua din zona superioară a tastaturii şi nu din zona numerică dreapta. De asemenea, activarea unei ferestre se poate realiza prin selecția din lista de ferestre generată prin combinația de taste **Alt + 0** sau opțiunea de meniu **List** din meniul **Window** sau prin parcurgere succesiv/circulară crescătoare prin tasta **F6** respectiv descrescătoare prin **Shift + F6** sau opțiunile corespondente **Next** respectiv **Previous** din meniul **Window**.
- **Simbolul de redimensionare** – care poate avea forma unei săgeți simple **[↑]** sau duble **[↕]**, încadrate între paranteze drepte; în primul caz fereastra poate fi maximizată (dispusă la dimensiunile maximale), prin click stânga mouse pe simbolul de redimensionare; în al doilea caz, fereastră este restaurată la dimensiunile de dinaintea maximizării. Aceleaşi efecte se obțin şi prin dublu click stânga mouse pe linia superioară a ferestrei sau prin tasta **F5** respectiv opțiunea **Zoom** din meniul **Window**.
- **Barele de defilare orizontală şi verticală** – permit defilarea orizontală respectiv verticală a conținutului ferestrei, atunci când informația alocată ferestrei depăşeşte dimensiunile acesteia. Defilarea se face prin poziționarea cursorului stâng mouse pe direcția de defilare dorită şi acționarea concomitent cu menținerea apăsată a acestuia, ceea ce va provoca defilarea textului conform direcției săgeții.
- **Colțul de redimensionare** – permite redimensionarea dinamică a ferestrei; operația se realizează prin poziționarea cursorului stâng mouse pe acest colț şi menținerea apăsată a butonului mouse concomitent cu mişcarea acestuia, ceea ce va provoaca redimensionarea ferestrei şi fixarea finală a dimensiunilor la eliberarea butonului mouse.
- **Indicatorul poziției cursorului** – indică în formatul **linie : coloana** poziția curentă a cursorului de editare, fig. 3.2.

- **Indicatorul de modificare al textului** – se generează automat sub forma caracterului ☸ dacă în fişierul curent s-au efectuat modificări, în caz contrar acesta nefiind vizibil.

Dacă sunt deschise simultan mai multe ferestre, acestea pot fi dispuse sub formă de "**cărămidă**" prin selecţia opţiunii de meniu **Tile** din bara de meniu **Window** sau în "**cascadă**" (suprapuse) prin selecţia opţiunii de meniu **Cascade** din bara de meniu **Window.**

Închiderea tuturor ferestrelor deschise se poate realiza prin selecţia opţiunii de meniu **Close All** din bara de meniu **Window.**

3.4 FERESTRE DE DIALOG

Ferestrele de dialog se folosesc la interacţiunea utilizatorului cu mediul de programare TurboPascal, fig. 3.4 şi conţin următoarele tipuri de controale:

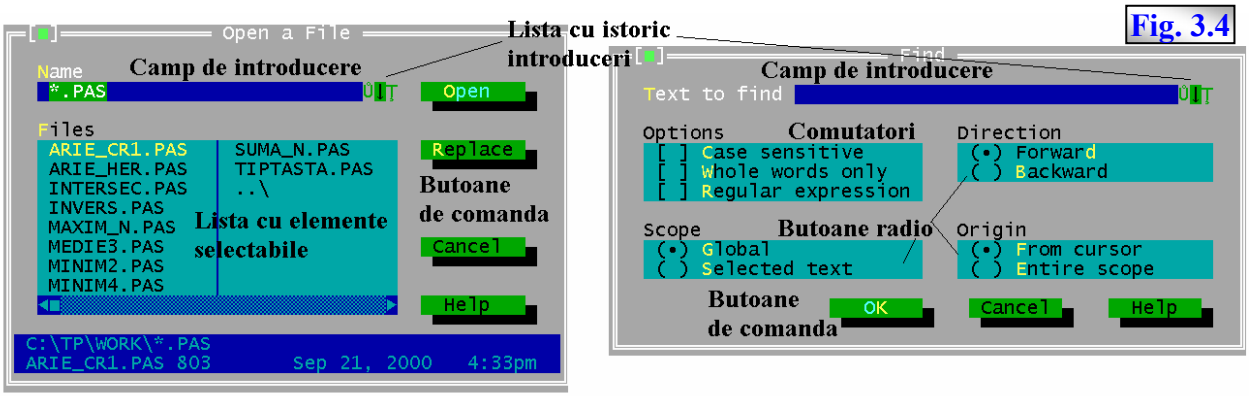


Fig. 3.4

- **câmp de introducere date** – zona rezervată pentru introducere date: nume fişier, text, etc.;
- **lista cu istoric introduceri [↓]** – afişează valorile introduse anterior în câmpul asociat, în vederea reselecţiei acestora;
- **lista cu elemente selectabile** - permite selecţia unei opţiuni din mai multe posibilităţi predefinite: fişiere, directoare, unităţi de disc;
- **butoane radio** – permite selecţia unei singure opţiuni din mai multe posibile grupate unitar; fiecare opţiune este precedată de o pereche de paranteze rotunde (); prezenţa unui "•" în interiorul parantezei specifică activarea opţiunii (•), iar absenţa acestuia specifică neactivarea opţiunii ( );
- **comutatoare** – permite activarea sau nu a unei opţiuni; fiecare opţiune este precedată de o pereche de paranteze drepte []; prezenţa unui "X" în interiorul parantezei specifică activarea opţiunii [X], iar absenţa acestuia specifică neactivarea opţiunii [ ];
- **butoane de comandă** – servesc la declanşarea unei acţiuni; numele butonului de comandă se află încadrat într-un chenar şi descrie succint acţiunea asociată butonului. Butonul **OK** declanşează execuţia comenzii şi închiderea ferestrei de dialog; butonul **Cancel** anulează execuţia comenzii şi provoacă închiderea ferestrei de dialog, efecte care se obţin şi prin apăsarea tastei **ESC**; butonul **Help** declanşează apariţia unui ecran ce conţine informaţii în limba engleză despre comanda în curs de execuţie, acţiune care se poate obţine şi prin intermediul tastei **F1**. Activarea unui buton se face prin click stânga mouse pe buton sau, prin tasta **Enter**, după poziţionarea cursorului pe acesta.

Parcurgerea controalelor unei ferestre de dialog se face circular prin tasta **TAB** sau **SHIFT+TAB**.

3.5 TASTE DE APEL

Acestea sunt taste foarte des utilizate şi permit realizarea rapidă a acţiunilor, dublând funcţiile meniului. Cele mai importante / frecvent utilizate taste sau combinaţii de taste sunt:

- **F1** – activare help contextual;
- **F2** – salvare fişier din fereastra curentă;
- **F3** – activează fereastra de dialog pentru deschiderea unui fişier;
- **F6** – parcurgere succesiv/circulară a mai multor ferestre deschise;
- **F9** – (Make) compilare cu legare de unit-uri;
- **F10** – activare meniu principal;
- **Alt + F9** – compilare program;
- **Ctrl + F9** – rulare program;
- **Alt + F5** – afişare fereastră cu rezultatele programului;
- **Alt + X** – ieşire din mediul TurboPascal;
- **ESC** – ieşire din meniu sau abandonarea comenzii în curs de execuţie;
- **Ctrl + C -** întreruperea execuţiei unui program.

Pentru execuţia unei combinaţii de taste de genul **Alt + F5** se procedează astfel: se apasă şi se menţine apăsată prima tastă din combinaţie (**Alt**, **Ctrl** sau **Shift**) în timp ce se apasă a doua tastă (exemplu **F5**).

3.6 EDITORUL DE TEXTE

Editorul de texte este încorporat în mediul TurboPascal şi se foloseşte la introducerea programele în forma sursă a lor. Editorul de texte este de tip multifişier, adică pot fi editate simultan mai multe fişiere (în mai multe ferestre de editare), cu dimensiuni ce pot depăşi 1 Mb. Numărul maxim de linii este 2^{15}-1, adică 32767. Lungimea maximă a unei linii program este 127, iar a unui fişier document ASCII este de 1023. La depăşirea lungimii maxime apare mesaj de eroare "**Error. Line too long, truncated**".

Se defineşte **cursorul de editare** ca fiind caracterul ce marchează poziţia curentă în cadrul fişierului şi este evidenţiat prin afişare alternativ clipitoare. Forma cursorului depinde de modul editare activ:

- formă de **liniuţă de subliniere orizontală** (fig. 3.2) - atunci când este activ modul **inserare**; acest mod de lucru corespunde posibilităţii de intercalare, la stânga poziţiei cursorului, a unui nou caracter între cele existente; de exemplu pentru cuvântul scris iniţial în forma "**barça**" şi poziţia cursorului liniuţă pe litera "**c"**, tastarea literei "**a**" va produce intercalarea literei "**a**", în faţa poziţiei cursorului, rezultând astfel cuvântul "**baraca**";
- formă de **dreptunghi** - atunci când este activ modul **suprascriere**; acest mod de lucru corespunde posibilităţii de scriere de caractere, la poziţia cursorului, prin înlocuirea celor existente; de exemplu, pentru cuvântul scris iniţial în forma "**banca"** şi poziţia cursorului dreptunghiular pe litera "**n"**, tastarea literei "**r**" va produce înlocuirea literei "**n"** cu "**r**", generând cuvântul "**barca"**.

Comutarea modului de editare între variantele **inserare** respectiv **suprascriere** se face prin intermediul tastei **Ins** sau a combinației de taste **Ctrl + V**.

Tabelul 3.1 prezintă principalele acțiuni de editare şi combinațiile de taste asociate:

Tabel 3.1

Acțiunea	**Combinația de taste / tasta**	**Domeniu de aplicare**
Un caracter la stânga	←	Comenzi pentru deplasarea cursorului
Un caracter la dreapta	→	
Un rând în sus	↑	
Un rând în jos	↓	
Deplasare rapidă la început de linie	Home	
Deplasare rapidă la sfârşit de linie	End	
Deplasare cu o pagină în sus	PgUp	
Deplasare cu o pagină în jos	PgDn	
Deplasare rapidă la început de fişier	Ctrl +Q R	
Deplasare rapidă la sfârşit de fişier	Ctrl +Q C	
Tabulare	TAB	
Trecere la linie nouă	Enter	
Inserare linie nouă	Ctrl +N	Comenzi de inserare - ştergere
Ştergere linie	Ctrl +Y	
Ştergere caracter marcat de cursor	Del	
Ştergere caracter din stânga cursorului	BackSpace sau ←	
Căutare text	Ctrl +Q F	Comenzi de căutare înlocuire text
Căutare şi înlocuire text	Ctrl +Q A	
Marcare început bloc	Ctrl +K B	Comenzi de lucru cu blocuri
Marcare sfârşit bloc	Ctrl +K K	
Selecție bloc de la început spre sfârşit sau invers	Selecție prin taste direcționale cu tasta SHIFT apăsată sau prin deplasare mouse cu butonul stâng apăsat	
Copiere bloc la poziția cursorului	Ctrl +K C	
Mutare bloc la poziția cursorului	Ctrl +K V	
Ştergere bloc	Ctrl +K Y	
Ascundere marcaj bloc	Ctrl +K H	
Copiere bloc în **Clipboard**	Shift + Del	Comenzi de lucru cu **Clipboard**
Mutare bloc în **Clipboard**	Ctrl + Ins	
Readucere bloc din **Clipboard** la poziția cursorului	Ctrl + Del	

Pentru execuția unei combinații de taste de genul **Ctrl+K H** se procedează astfel: se apasă şi se menține apăsată tasta **Ctrl** în timp ce se apasă a doua tastă (exemplu **K**), urmat de apăsarea celei de-a doua taste (exemplu **H**).

Este frecventă acțiunea de copiere sau mutare a unei porțiuni din text (bloc) în altă parte a aceluiaşi fişier, care impune următoarea succesiune de operații:

- selecția blocului, care se poate realiza prin oricare din modalitățile următoare:

➢ poziționare cursor la început bloc şi marcare început prin combinația de taste **CTRL+K B**, urmat de poziționare cursor la sfârşit bloc şi de marcare sfârşit prin combinația de taste **CTRL+K K**;

➢ deplasarea cursorului dinspre început bloc spre sfârşit bloc sau invers, cu tasta **SHIFT** apăsată, simultan cu specificarea direcției de selecție folosind tastele direcționale;

➢ deplasarea cursorului mouse cu butonul stâng apăsat dinspre început bloc spre sfârşit bloc sau invers.

Indiferent de modalitatea de selecție utilizată, blocul va fi evidențiat prin afişare pe fundal diferit de cel al textului neselectat;

- deplasarea cursorului la poziția la care se va copia sau se va muta blocul;
- apăsarea combinației de taste **CTRL+K C** pentru copiere sau **CTRL+K V** pentru mutare blocului; dacă se doreşte ştergerea blocului marcat se apasă combinația de taste **CTRL+K Y**.

Transferul porțiunilor de text între diferite fişiere se poate face prin intermediul unui fişier special, denumit **Clipboard**, gestionat de mediu TurboPascal. Deci nu se poate copia o porțiune de text direct, dintr-o fereastră în alta. Pentru acțiunea de copiere sau mutare a unei porțiuni din text (bloc) dintr-un fişier sursă în alt fişier destinație, succesiunea de operații impuse de această acțiune este următoarea:

- deschiderea celor două fişiere;
- selecția blocului, în fişierul sursă, care se poate realiza prin oricare din modalitățile anterior specificate, ceea ce va provoca evidențierea blocului prin afişare pe fundal diferit de cel al textului neselectat;
- transferul blocului în **Clipboard** pentru copiere prin combinația **Shift + Del** sau pentru mutare prin combinația **Ctrl + Ins**;
- activarea ferestrei fişierului destinație, prin una posibilitățile descrise în & 3.3;
- poziționarea cursorului în fişierul destinație în locul unde se va copia sau se va muta blocul;
- apăsarea combinației de taste **Ctrl + Del** pentru readucere din **Clipboard** a blocului la poziția cursorului.

Această acțiune se poate utiliza şi pentru transferul de blocuri între diverse porțiuni ale aceluiaşi fişier.

Este frecventă greşeala de rupere a unui cuvânt sau linii, care poate rezulta ca şi o consecință a apăsării tastei **Enter**, atunci când cursorul de editare este activ în modul **inserare**. Din acest motiv vom insista asupra acestei greşeli de editare, pe care o vom exemplifica şi grafic, pentru a fi mai uşor de înțeles: să presupunem că cursorul de editare se află poziționat pe litera **G** din cuvântul **PROGRAM** (fig. 3.5a). Apăsarea tastei **Enter** va produce efectul exemplificat în fig. 3.5 b: textul care urmează după litera **G** inclusiv este transferat pe linia următoare (cursorul rămânând poziționat pe litera **G**), deoarece tasta **Enter** produce efectul trecerii la o nouă linie, efect datorat introducerii unui caracter care nu este vizibil pe ecran. Readucerea textului în forma sa inițială se realizează prin ştergerea acestui caracter invizibil, folosind tasta **BackSpace** (marcată pe alte tastaturi prin simbolul ←).

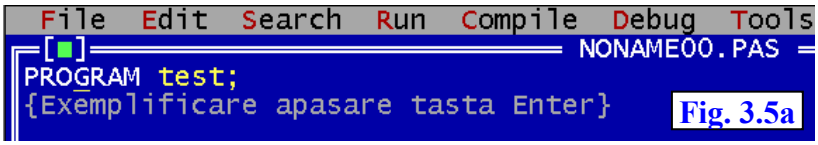


Fig. 3.5a

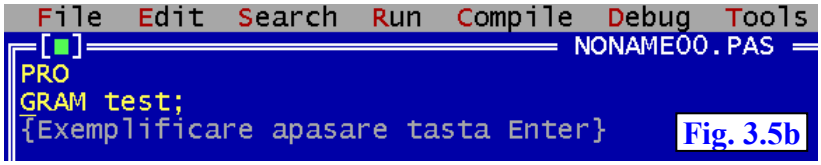


Fig. 3.5b

3.7 MENIUL TurboPascal

Meniul TurboPascal reprezintă un element al interfeței prin care utilizatorul poate comunica interactiv cu mediul TurboPascal. Linia meniului este localizată în partea superioară a ferestrei TurboPascal (fig. 3.2).

Selecția unei opțiuni din meniu poate avea ca şi consecință:

- lansarea unei comenzi;
- activarea unui submeniu (pentru opțiunile marcate cu ►);
- deschiderea unei ferestre de dialog (pentru opțiunile marcate cu …), & 3.4.

Meniul conține 10 opțiuni principale şi se activează prin tasta **F10**; deplasarea prin meniu se face prin tastele direcționale, iar selecția unei opțiuni din meniu se realizează, după poziționarea cursorului pe opțiunea dorită, prin intermediul tastei **Enter** sau prin selecția cu butonul stâng mouse. Activarea unui meniu derulant se poate realiza prin combinația de taste **Alt+literă**, unde prin **literă** se înțelege primul caracter al meniului, care este evidențiat printr-o culoare contrastantă în raport cu culoarea de afişarea a celorlalte caractere din denumirea meniului. Opțiunile inactive sunt marcate prin afişare în gri deschis, ele devenind active numai la îndeplinirea unor condiții impuse de mediul TurboPascal. Ieşirea din meniu se face prin **ESC**, iar din mediul TurboPascal prin combinația de taste **ALT+X** sau prin opțiunea de meniu **Exit** din meniul **File**.

În continuare vor fi prezentate principalele opțiuni ale meniului TurboPascal, pentru versiunea TurboPascal 7.0, elemente ale acestuia fiind însă valabile şi pentru versiuni anterioare.

3.7.1 Meniul File

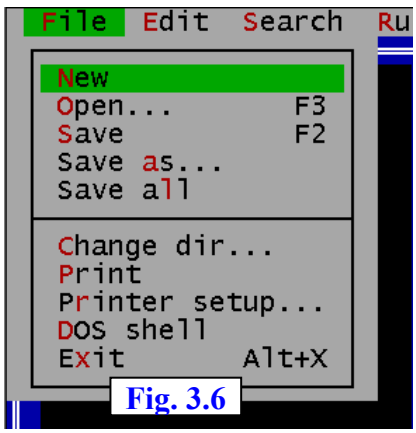


Fig. 3.6

Meniul **File** (fig. 3.6) permite gestionarea fişierelor, a directoarelor şi a ieşirilor din mediul TurboPascal.

Opțiunea **New** permite deschiderea unei noi ferestre de editare. Numele implicit asociat acesteia va fi sub forma **NONAMExx.PAS**, unde **xx** reprezintă un număr de ordine care poate varia de la 00 la 99. Numele final al fişierului poate fi specificat la salvarea acestuia.

Opțiunea **Open** permite deschiderea unui fişier salvat anterior, prin intermediul ferestrei de dialog **Open a File**, fig. 3.7. Acelaşi efect se obține şi prin intermediul tastei **F3**.

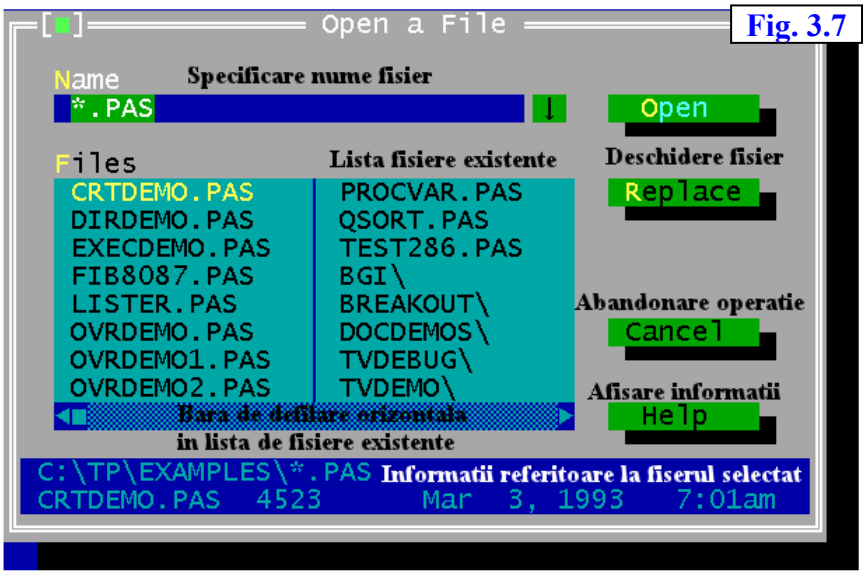


Fig. 3.7

Câmpul **Name** permite introducerea prin tastare a numelui fişierului sau se autocompletează cu numele fişierului selectat din lista **Files**. Iniţial acest câmp conţine o mască de tipul ***.PAS**, prin care se restricţionează afişarea numelor de fişiere din directorul curent numai la cele care au extensia **PAS**, adică a fişierelor sursă de tip PASCAL.

Lista **Files** afişează lista fişierelor din directorul curent. Poziţionarea cursorului pe oricare din nume (prin cursor mouse sau taste direcţionale) provoacă autocompletarea numele fişierului selectat în câmpul **Name**. Defilarea orizontală în această listă se poate realiza prin bara de defilare orizontală. La baza ferestrei se află o zonă de afişare de informaţiireferitoare la fişierul selectat curent în câmpul **Name**: unitate de disc, calea de directoare, nume, extensie, lungimea în octeţi, data şi ora ultimei editări. Butoanele **Open** respectiv **Replace** provoacă deschiderea fişierului selectat într-o fereastră de editare, diferenţa fiind generarea unei noi ferestre (prin butonul **Open**) sau înlocuirea fişierului din fereastra de editare curentă (prin butonul **Replace**). Butonul **Cancel** va produce abandonarea operatiei de deschidere a unui fişier, iar butonul **Help** oferă informaţii referitoare la semnificaţia controalelor din fereastra de dialog **Open a File**.

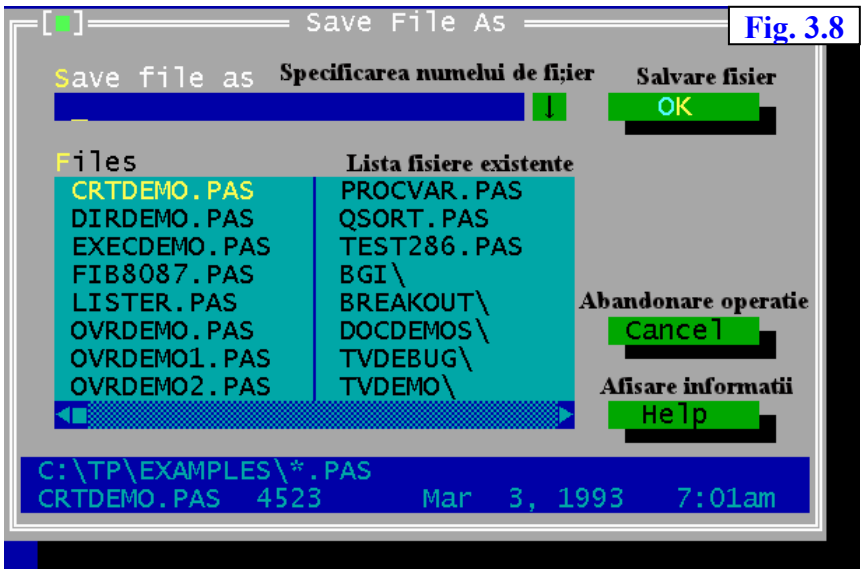


Fig. 3.8

Opţiunea **Save** permite salvarea unui fişier, prin intermediul ferestrei de dialog **Save File As**, fig. 3.8. Acelaşi efect se obţine şi prin intermediul tastei **F2**. Câmpul **Save File As** permite introducerea prin tastare a numelui fişierului sau se autocompletează cu numele fişierului selectat din lista **Files**. Numele fişierului trebuie să fie format din maxim 8 caractere şi să respecte restricţiile sistemului de operare MS-DOS.

Butonul **OK** declanşează operaţia de salvare, iar butonul **Cancel** provoacă abandonarea acesteia.

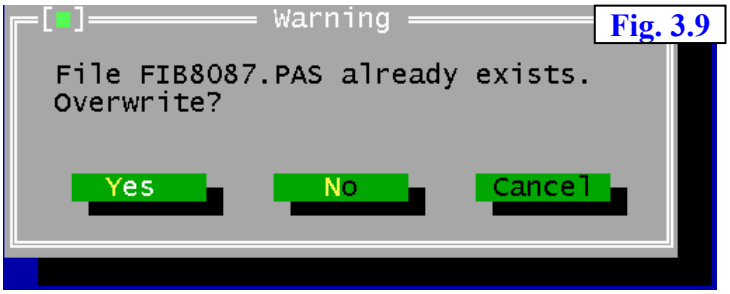


Fig. 3.9

Încercarea de salvare a unui fişier sub un nume al unui fişier existent va provoca apariţia ferestrei din fig. 3.9, prin care se cere confirmarea suprascrierii fişierului, adică înlocuirea versiunii existente cu versiunea nouă, pentru care se cere salvarea sub aceeaşi denumire. Confirmarea suprascrierii se realizează prin butonul **Yes**, infirmarea suprascrierii şi revenirea în fereastra de editare se realizează prin butonul **No**, iar abandonarea operaţiei de salvare se realizează prin butonul **Cancel**.

Opţiunea **Save As** permite salvarea fişierului din fereastra de editare curentă sub o altă denumire, care va provoca apariţia ferestrei de dialog **Save File As**, fig. 3.8.

Opţiunea **Save All** permite salvarea tuturor fişierelor din fereastrele de editare deschise.

Opţiunea **Change Dir** permite specificarea unei unităţi de disk şi/sau director ca unitate/director de lucru curent.

Opţiunea **Print** permite tipărirea conţinutului ferestrei curente de editare.

Opţiunea **Printer Setup** permite instalarea unei imprimante.

Opţiunea **Dos Shell** permite ieşirea temporară din mediul TurboPascal în sistemul de operare, unde pot fi executate comenzi sau lansate în execuţie alte programe. Revenirea în mediul TurboPascal se poate realiza prin comanda **Exit**, introdusă la prompter-ul sistemului de operare.

Opţiunea **Exit** permite ieşirea din mediul de programare TurboPascal şi revenirea în sistemul de operare. Dacă există ferestre de editare cu fişiere nesalvate, atunci, anterior ieşirii, se va cere succesiv confirmarea de salvare a acestora.

3.7.2 Meniul Edit

Meniul **Edit** (fig. 3.10) permite anularea sau refacerea unor operaţii de editare precum şi manipularea operaţiilor cu **Clipboard**-ul. **Clipboard**-ul este un fişier tampon, prin intermediul căruia se poate realiza transferul de porţiuni (blocuri) de texte între diverse fişiere aflate în ferestre de editare diferite sau între diferite porţiuni ale aceluiaşi fişier, conform metodologiei detaliate în & 3.6.

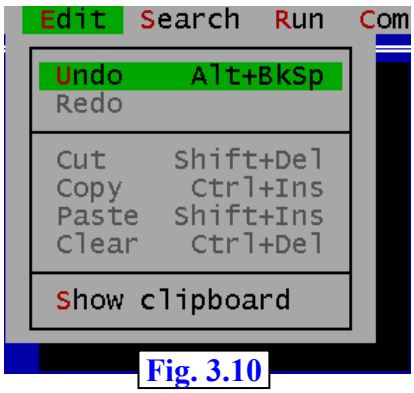


Fig. 3.10

Opţiunea **Undo**, realizabilă şi prin combinaţia de taste **Alt+BkSp**, permite anularea uneia sau mai multor operaţii de editare (ştergeri, inserări sau suprascrieri de texte, deplasări ale cursorului). Dacă este activ comutatorul **Group Undo** din fereastra **Editor Options**, activată din meniul TurboPascal în succesiunea **Options → Environment → Editor Options**, se permite anularea unui grup de acţiuni, în sens invers redactării, în caz contrar se poate anula numai ultima operaţie de editare.

Opţiunea **Redo** permite refacerea acţiunilor anulate prin comanda **Undo**, în forma şi ordinea originală. În cazul absenţei acestora, opţiunea nu este activă.

Opțiunile **Cut** şi **Copy** depune în **Clipboard** un bloc selectat în vederea mutării sau copierii acestuia, modalitățile de selecție fiind detaliate în & 3.6.

Opțiunea **Paste** transferă din **Clipboard** la poziția curentă a cursorului un bloc depus anterior în **Clipboard** printr-una din operațiile **Cut** sau **Copy**.

Opțiunea **Clear** şterge un bloc marcat din fereastra curentă de editare, fără însă a îl depune în **Clipboard**.

Opțiunea **Show Clipboard** permite vizualizarea **Clipboard**-lui într-o fereastră de editare cu numele de **Clipboard**, fără însă a se permite operațiile de salvare sau compilare.

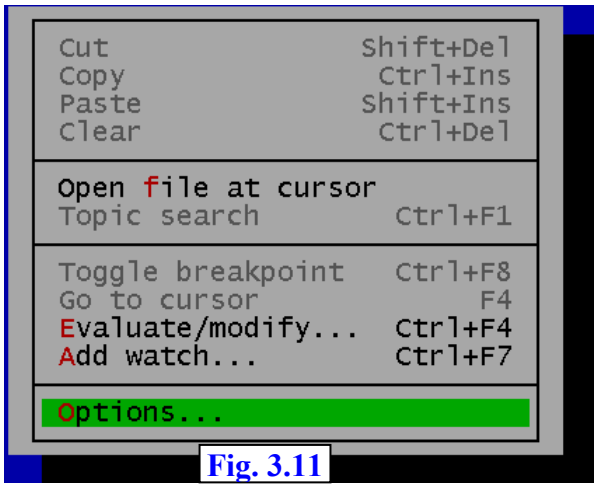


Fig. 3.11

Meniul **Edit** are ataşat un meniu local, apelabil prin combinația de taste **Alt+F10** sau prin buton dreapta mouse, fig. 3.11.

Opțiunile **Cut**, **Copy**, **Paste** sunt identice cu cele descrise anterior.

Opțiunea **Open file at cursor** permite deschiderea rapidă a unui fişier al cărui nume coincide cu cuvântul pe care se află poziționat cursorul; dacă un astfel de fişier nu există sau nu poate fi deschis se afişează mesaj de eroare **File …… not found**, în caz contrar fişierul este deschis într-o nouă fereastră de editare.

Opțiunea **Topic Search este** identică cu opțiunea similară din meniul **Help**.

Opțiunea **Toggle breakpoint, Evaluate/Modify** şi **Add Watch** sunt identice cu opțiunile similare din meniul **Debug**.

Opțiunea **Go to cursor** este identică cu opțiunea similară din meniul **Run**.

Opțiunea **Options** este identică cu opțiunea **Editor Options** din meniul **Options** submeniul **Environment**.

3.7.3 Meniul Search

Meniul **Search** (fig. 3.12) permite efectuarea de căutări şi înlocuiri de texte, gestiune de erori, deplasări ale cursorului.

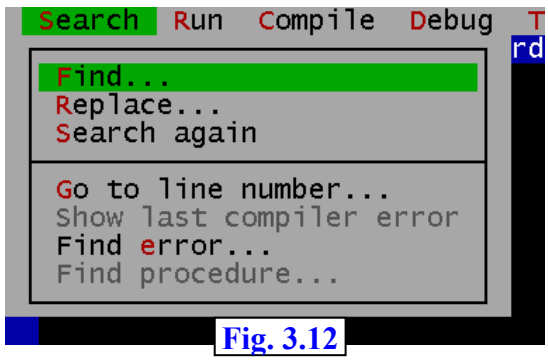


Fig. 3.12

Opțiunea **Find** permite căutarea unui text în interiorul fişierului şi deplasarea cursorului pe acesta, dacă căutarea este finalizată cu succes; în caz contrar se afişează mesajul de atenționare **Search string not found**. Selecția comenzii provoacă apariția ferestrei **Find**, fig. 3.13, prevăzută cu următoarele controale:

- **Text to find** – câmp rezervat pentru introducerea şirului de caractere ce se va căuta; la apariția ferestrei în acest câmp se autocompletează cuvântul pe care se află poziționat cursorul, dar acesta poate fi modificat şi/sau editat; de asemenea câmpul are asociat un control de tip listă cu istoric **[↓]** din care se poate reselecta un şir pentru care s-a efectuat o căutare anterioară;

- zona **Options**- conține 3 comutatoare prin care se impune sau nu efectuarea căutării:
  - ținând cont de diferențe între litere mari și mici (comutatorul **Case sensitive**);
  - căutarea numai a cuvintelor distincte, adică a celor care sunt încadrate de spațiu sau semne de punctuație (comutatorul **Whole words only**); de exemplu activarea acestui comutator pentru șirul **apa** nu va finaliza cu succes căutarea, chiar dacă în text există cuvântul **aparat**, deoarece șirul **apa** este inclus în cuvântul **aparat** și nu există ca și cuvânt distinct și separat, marcat de spații înainte și după; dezactivarea acestui comutator va finaliza cu succes căutarea pentru exemplul analizat, deoarece în această situație se vor căuta și șirurile incluse ca subșiruri în alte cuvinte;
  - utilizarea unor semne speciale în operația de căutare (comutatorul **Regular expresion**).
- controlul **Direction** – este un buton radio prin care se impune căutarea spre sfârșitul sau începutul fișierului (**Forward**) respectiv (**Backward**);
- controlul **Scope** - este un buton radio prin care se impune căutarea în întregul fișier (**Global**) sau numai într-o porțiune selectată a acestuia (**Selected text**);
- controlul **Origin** – este un buton radio prin care se impune căutarea de la poziția curentă a cursorului (**From cursor**) respectiv în întregul fișier sau bloc, neținând cont de poziția curentă a cursorului (**Entire scope**);
- comutatorul **OK** declanșează operația de căutare;
- comutatorul **Cancel** abandonează operația de căutare;
- butonul **Help** oferă informații referitoare la semnificația controalelor din fereastra de dialog **Find**.

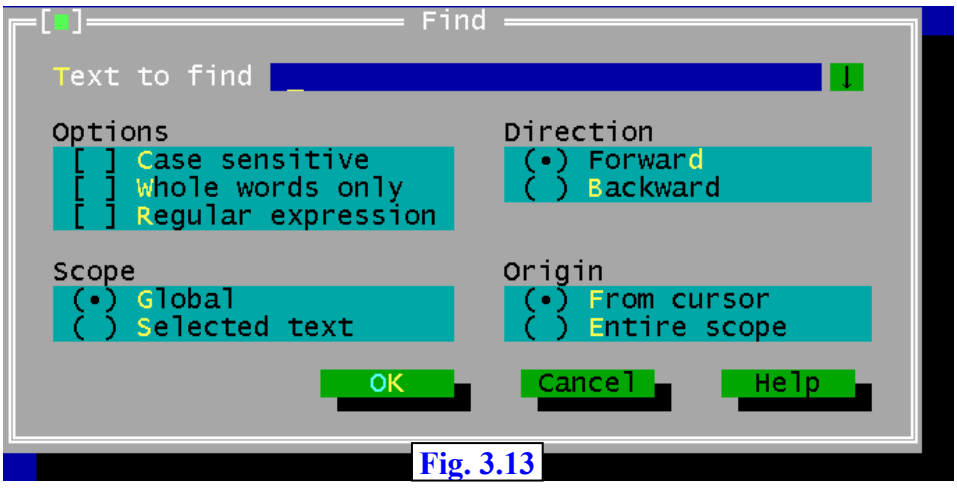


Fig. 3.13

Opțiunea **Replace** din meniul **Search** permite căutarea unui text în interiorul fișierului și înlocuirea acestuia cu un alt text; dacă căutarea nu este finalizată cu succes se afișează mesajul de atenționare **Search string not found**. Selecția comenzii provoacă apariția ferestrei **Replace**, fig. 3.14, prevăzută cu controale similare funcțional cu cele din fereastra **Find**, fig. 3.13, cu următoarele observații suplimentare:

- în câmpul **Text to find** se introduce șirul de caractere căutat pentru înlocuire;

- în câmpul **New text** se introduce şirul de caractere cu care se va înlocui şirul specificat în câmpul **Text to find**;
- comutatorul **Prompt on replace** – cere confirmarea de către utilizator a efectuării înlocuirii (prin butoanele **Yes** sau **No**) sau abandonarea operației prin **Cancel**, confirmare realizată prin intermediul ferestrei **Information**, fig. 3.15;
- butonul **Change all** – va provoca înlocuirea tuturor aparițiilor textului căutat cu textul cel nou.

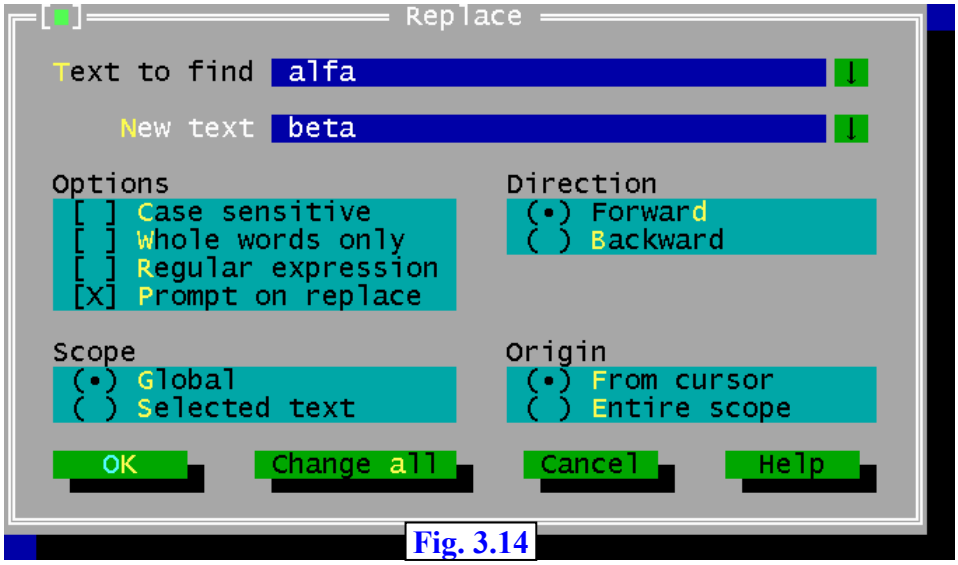


Fig. 3.14

Opțiunea **Search again** din meniul **Search** permite reluarea ultimei operații de căutare **Find** sau înlocuire **Replace**, cu conservarea opțiunilor specificate în fereastra de dialog asociată comenzii.

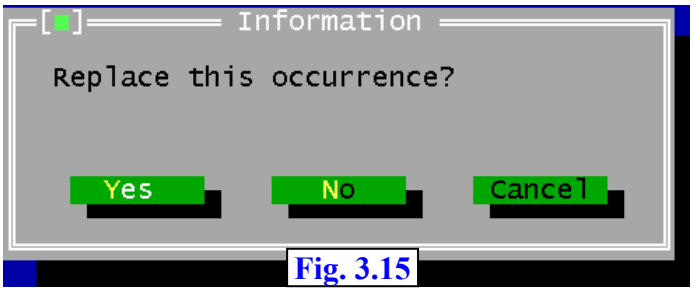


Fig. 3.15

Opțiunea **Go to line number** din meniul **Search** activează fereastra de dialog **Go to line number**, fig. 3.16, care permite deplasarea cursorului la o linie din fişier al cărui număr este specificat în câmpul **Enter new line number** sau este selectat din istoricul **[↓]** asociat acestui câmp.

Încercarea de căutare a unui număr de linie mai mare decât numărul total de linii din fişier este semnalizată cu o fereastră de atenționare cu mesajul **Value not within the valid range,** urmat de specificarea domeniului valid, prin indicarea primului şi a ultimului număr de linie disponibil în fişier.

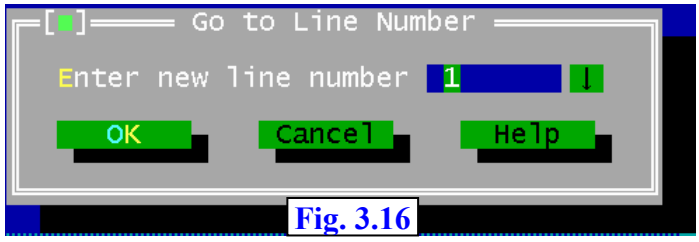


Fig. 3.16

Opțiunea **Show last compiler error** din meniul **Search** afişează din nou codul şi textul mesajului de eroare rezultat în urma ultimei compilări, concomitent cu poziționarea cursorului în dreptul erorii.

Opțiunea **Find error** din meniul **Search** permite localizarea, în codul sursă, a unei instrucțiuni ce cauzează o eroare de execuție, obținută prin lansarea în execuție a fişierului executabil rezultat în urma compilării. Pentru identificarea liniei sursă, se introduce în fereastra de dialog **Find Error** parametrii erorii, respectiv adresa segmentului şi a deplasamentului instrucțiunii, parametrii generați la oprirea cu eroare a execuției fişierului executabil.

Opțiunea **Find procedure** din meniul **Search** permite localizarea, în codul sursă, a unei proceduri, prin specificarea numelui său în fereastra de dialog **Find Procedure**.

3.7.4 Meniul Run

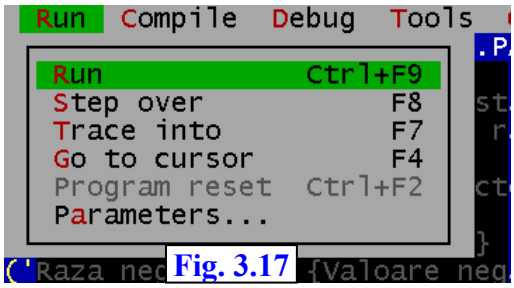


Fig. 3.17

Meniul **Run** (fig. 3.17) permite execuția unui program, precum şi opțiuni specifice acestei acțiuni.

Opțiunea **Run** declanşează lansarea în execuție a unui program, utilizând parametrii de comandă specificați în opțiunea **Parameters** din acelaşi meniu. Dacă programul sursă a fost modificat de la ultima sa rulare sau dacă nu a fost compilat, anterior lansării în execuție se produce operația de compilare a programului. Programul este executat până la sfârşitul său sau până la primul punct de întrerupere (dacă există definit), după care controlul revine înapoi mediului TurboPascal.

Opțiunea **Step over** sau tasta **F8** declanşează lansarea în execuție a programului, dar execuția va fi efectuată prin parcurgere instrucțiune cu instrucțiune şi evidențierea în paralel a instrucțiunii aflate în curs de execuție, prin afişarea acesteia într-o culoare contrastantă comparativ cu restul instrucțiunilor (*bară de execuție*). Trecerea la următoarea instrucțiune se face prin apăsarea tastei **F8**. Dacă instrucțiunea curentă reprezintă un apel de procedură, aceasta este parcursă normal şi nu prin evidențiere linie cu linie. În timpul execuției pas cu pas, programul afişează alternativ fereastra cu rezultate, aşteaptă introducerea de date, deci se comportă ca şi la execuția normală, dar prin execuție pas cu pas. Această modalitate de lucru este deosebit de utilă la depanarea programelor şi este disponibilă numai dacă este activat **comutatorul Debug Information** din fereastra **Compiler Options** activată din meniul TurboPascal în succesiunea **Options → Compiler**. Comanda lansează o sesiune de depanare a programului.

Opțiunea **Trace into** sau tasta **F7** are acelaşi efect ca şi opțiunea **Step over**, cu diferențele că sunt parcurse pas cu pas şi evidențiate prin bara de execuție inclusiv instrucțiunile procedurilor, iar trecerea la pasul următor se realizează prin tasta **F8**. Comanda lansează o sesiune de depanare a programului.

Opțiunea **Go to cursor** sau tasta **F4** are efectul transferului controlului execuției programului prin execuția rapidă a porțiunii de cod cuprinse între poziția curentă aflată în curs de execuție (sau prima linie executabilă a programului, în cazul în care nu s-a început depanarea acestuia) şi linia la care se află poziționat cursorul în fereastra de editare. Această

modalitate de lucru este deosebit de utilă la depanarea programelor pentru a scurtcircuita porțiuni din cod pentru care s-a efectuat verificarea sau la care nu este necesară verificarea prin execuție linie cu linie. Comanda este disponibilă numai dacă este activat **comutatorul Debug Information** din fereastra **Compiler Options** activată din meniul TurboPascal în succesiunea **Options → Compiler** şi lansează o sesiune de depanare a programului.

Opțiunea **Program reset** sau combinația de taste **Ctrl+F2** reinițializează sesiunea de depanare curentă prin:

- ştergerea barei de execuție
- eliberarea memoriei ocupate de variabilele programului
- închiderea fişierelor deschise
- conservarea valorii variabilelor şi a punctelor de întrerupere
- reluarea următoarei execuții a programului de la început.

Opțiunea **Parameters** activează fereastra de dialog **Program Parameters,** fig. 3.18, care permite introducerea în câmpul **Parameter** a mai multor parametrii asociați programului, despărțiți de câte un caracter blanc. În interiorul programului numărul parametrilor se poate determina prin funcția **ParamCount**, iar valoarea parametrului "**n**" cu funcția **ParamStr(n)**.

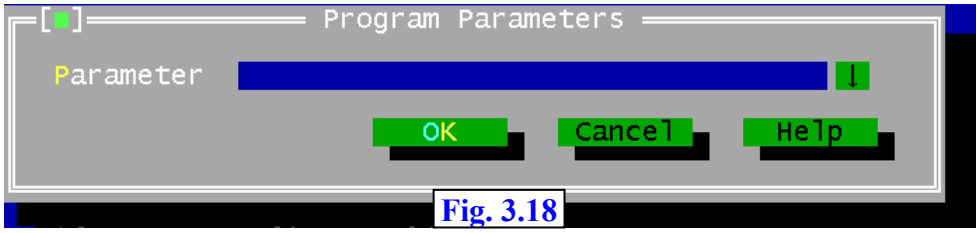


Fig. 3.18

3.7.5 Meniul Compile

Meniul **Compile** (fig. 3.19) permite compilarea programelor şi a **unit**-urilor, precum şi opțiuni specifice acestei acțiuni. **Unit**-ul reprezintă o colecție de constante, declarații de tip şi de variabile, proceduri şi funcții, care poate fi compilată separat şi poate fi utilizată de un program principal sau alt **unit**.

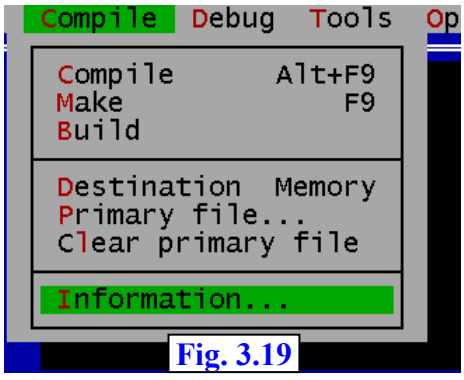


Fig. 3.19

Opțiunea **Compile** din meniul **Compile** permite compilarea fişierului din fereastra de editare curentă, fişier care poate fi un program principal sau un **unit**. Textul compilat poate conține referiri şi la fişiere externe de includere (directive de tip **$I nume**). Compilarea poate fi declanşată şi prin combinația de taste **Alt+F9**.

Destinația compilării poate fi fixată atât în memorie, cât şi pe disk, în această ultimă situație rezultatul final al compilării va fi memorat într-un fişier cu acelaşi nume şi cu extensia "**exe**", iar al unui unit într-un fişier cu extensia "**TPU**" (Turbo Pascal Unit). În timpul compilării şi la sfârşit se afişează informații referitoare la procesul compilării, prin fereastra **Compiling**, fig. 3.20. Afişarea fereastrei poate fi anulată prin apăsarea unei taste arbitrare.

Fig. 3.20

În situația în care compilatorul descoperă o eroare, se trece automat în fereastra de editare, se poziționează cursorul în zona erorii şi se afişează o bandă de eroare, care conține codul şi natura erorii, fig. 3.21. Anexa **2** conține lista celor mai frecvente erori de compilare.

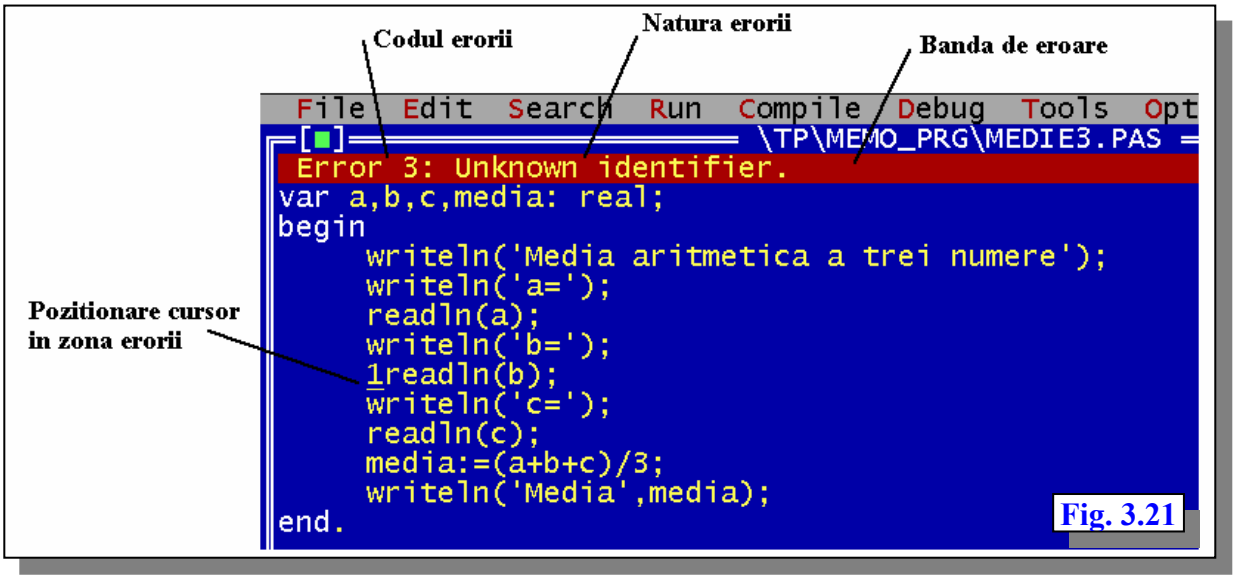


Fig. 3.21

Opțiunea **Make** din meniul **Compile** sau tasta **F9** compilează fişierul primar specificat sau, în lipsa acestuia, fişierul curent aflat în editare. De asemenea se compilează toate **unit**-urile asociate fişierului primar, funcție de data de creare / modificare a acestora. Compilarea se referă şi la fişierele incluse.

Opțiunea **Build** din meniul **Compile** compilează fişierul primar specificat sau, în lipsa acestuia, fişierul curent aflat în editare şi toate toate **unit**-urile asociate, fără a ține însă cont de data de creare / modificare a acestora. Aceasta este singura diferență față de opțiunea **Make**.

Opțiunea **Destination** din meniul **Compile** este o opțiune de tip comutator, deci are două poziții referitoare la destinația compilării, deci a locului depunerii codului executabil: în memorie, pentru opțiunea **Destination Memory** respectiv pe disk sub forma unui fişier cu extensia "**exe**", pentru opțiunea **Destination Disk**. Pentru destinația memorie, la ieşirea din mediul TurboPascal codul executabil se pierde, iar comenzile **Make** şi **Build** vor regenera pe disk toate fişierele cu extensia **TPU**.

Opțiunea **Primary File** din meniul **Compile** activează fereastra de dialog **Primary File**, fig. 3.22, care permite specificarea fișierului primar, subiect al compilării pentru comenzile de compilare **Make** respectiv **Build**. Numele fișierului primar poate fi specificat în câmpul **Primary program file** sau selectat din lista **Files**. Butonul **Clear** anulează fișierul primar curent, fără a specifica altul în loc, după care închide fereastra de dialog **Primary File**.

Fig. 3.22

Opțiunea **Clear primary file** din meniul **Compile** anulează fișierul primar curent, fără a specifica altul în loc.

Opțiunea **Information** din meniul **Compile** afișează fereastra de dialog **Information**, fig. 3.23, în care sunt prezentate informații referitoare la numărul liniilor compilate, dimensiunea codului generat, dimensiunea segmentului de date și de stivă, a heap-ului minim și maxim, spațiul de memorie reală sau expandată ocupat de DOS și mediul de programare.

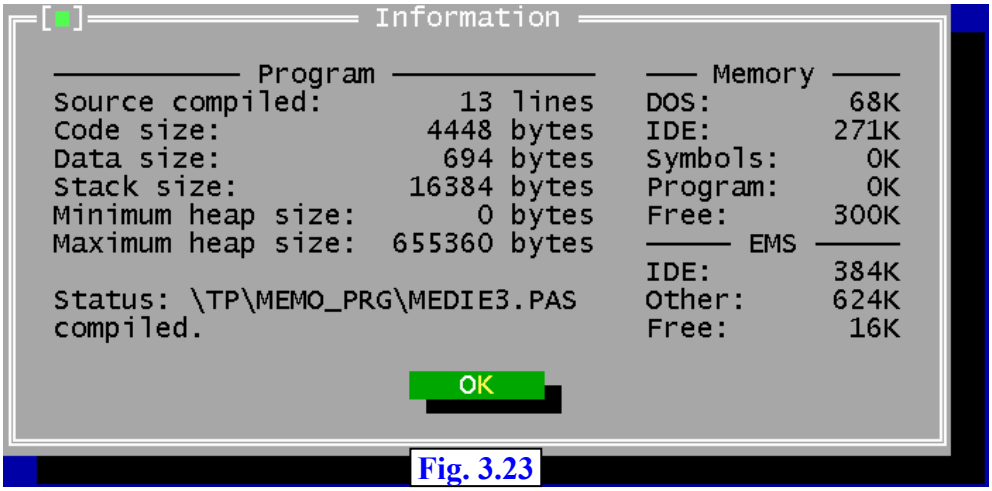


Fig. 3.23

3.7.6 Meniul Debug

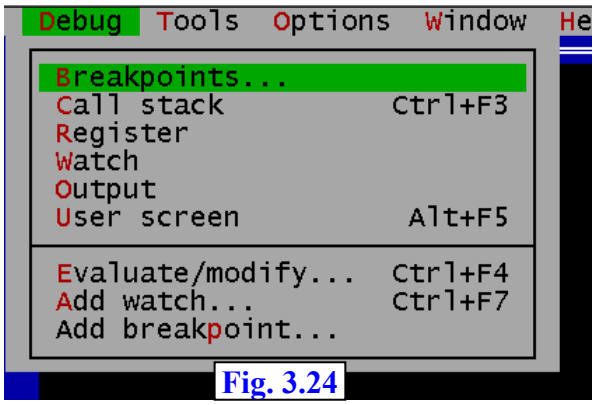


Fig. 3.24

Meniul **Debug** (fig. 3.24) oferă opțiuni pentru gestionarea depanării programelor.

Opțiunea **Breakpoints** din meniul **Debug** afișează fereastra de dialog **Breakpoints**, fig. 3.25, care permite gestionarea *punctelor de întrerupere*. Acestea sunt locuri impuse în program, concretizate prin instrucțiuni, la care se provoacă oprirea programului la momentul execuției. Liniile programului prevăzute cu punct de întrerupere sunt evidențiate printr-o altă culoare a fundalului sau cu o intensitate mai ridicată, comparativ cu instrucțiunile normale, neprevăzute cu puncte de întrerupere. La momentul execuției, când programul ajunge la punctul de întrerupere, se oprește execuția acestuia și se pot efectua de către utilizator diverse operații: evaluări, modificări, execuție pas cu pas, etc. Punctele de întrerupere pot fi fixate numai pe instrucțiuni executabile.

Fereastra **Breakpoints** conține o listă a punctelor de întrerupere impuse, pentru fiecare punct fiind specificate: numele programului sursă, numărul liniei în programul sursă, condiția care trebuie îndeplinită pentru a avea loc oprirea execuției precum și un contor de trecere, contor care precizează numărul de treceri admise pentru a nu se produce oprirea.

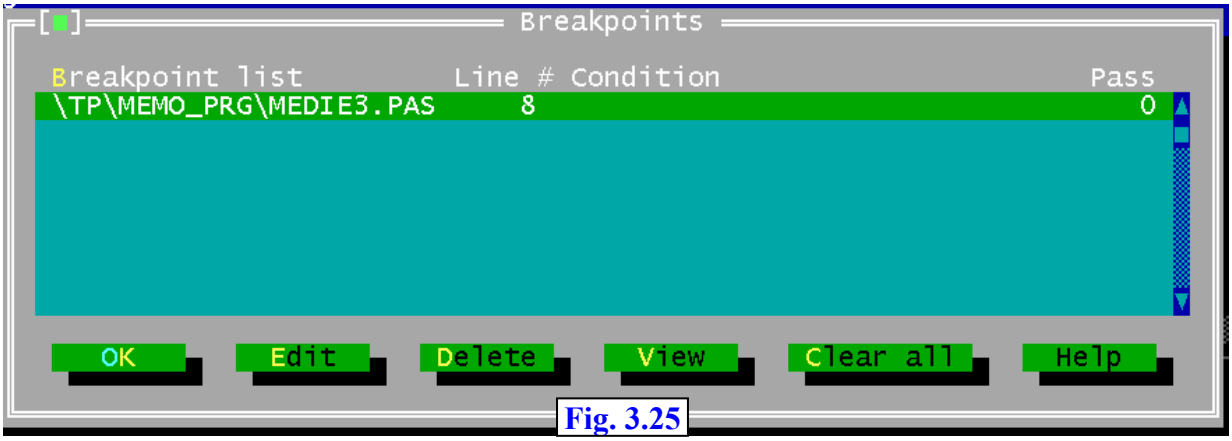


Fig. 3.25

Declanșarea efectivă a opririi depinde de:

- realizarea condiției impuse; condiția este de tip logic și respectă sintaxa limbajului TurboPascal; dacă la momentul execuției evaluarea condiției returnează valoarea adevărat, execuția programului se oprește, în caz contrar execuția continuă;
- realizarea valorii 0 a contorului de trecere; un contor intern, inițializat cu valoarea impusă a numărului de treceri, este decrementat cu 1 la fiecare trecere; dacă valoarea acestuia este pozitivă execuția continuă, iar dacă acesta devine egal cu 0 se produce oprirea.

Butoanele din fereastra **Breakpoints** au următoarele semnificații:

- butonul **OK** impune punctele de întrerupere definite în lista de întreruperi;
- butonul **Delete** șterge punctul de întrerupere selectat în lista de întreruperi;

- butonul **View** activează fereastra de editare şi deplasează cursorul la punctul de întrerupere selectat în lista de întreruperi;
- butonul **Clear All** şterge toate punctele de întrerupere din lista de întreruperi;
- butonul **Help** oferă informaţii referitoare la semnificaţia controalelor din fereastra de dialog **Breakpoints**;
- butonul **Edit** deschide o nouă fereastră de dialog **Edit Breakpoints**, fig. 3.26, prin intermediul căreia se pot modifica elementele unui punct de întrerupere (butonul **Modify**) sau se pot adăuga elementele unui nou punct de întrerupere (butonul **New**). Elementele unui punct de întrerupere se referă la condiţia (câmpul **Condition**) şi numărul de treceri (câmpul **Pass count**), calea fişierului sursă (câmpul **File name**) şi numărul liniei unde se impune oprirea (câmpul **Line number**).

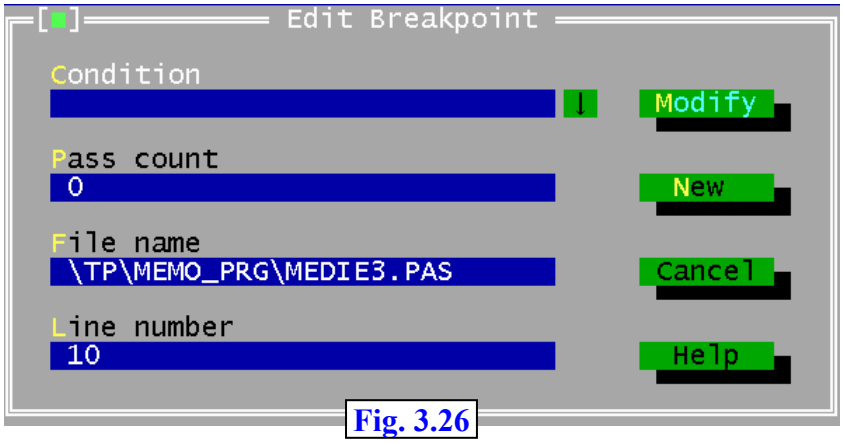


Fig. 3.26

Opţiunea **Call stack** din meniul **Debug** sau combinaţia de taste **Ctrl+F3** afişează fereastra de dialog **Call stack**, care permite vizualizarea stivei cu istoricul apelurilor subprogramelor, prin afişarea numelui subprogramelor şi a parametrilor actuali. Informaţiile referitoare la subprogramul actual sunt afişate în vârful stivei, iar la baza acesteia se afişează numele programului principal.

Opţiunea **Register** din meniul **Debug** afişează în colţul din dreapta sus a ferestrei de dialog **CPU**, care prezintă conţinutul regiştrilor procesorului. Fereastra se poate utiliza la depanarea secvenţelor de program scrise în limbaj de programare.

Opţiunea **Watch** din meniul **Debug** afişează fereastra de dialog **Watches**, fig. 3.27, care afişează valorile variabilelor şi expresiilor supravegheate.

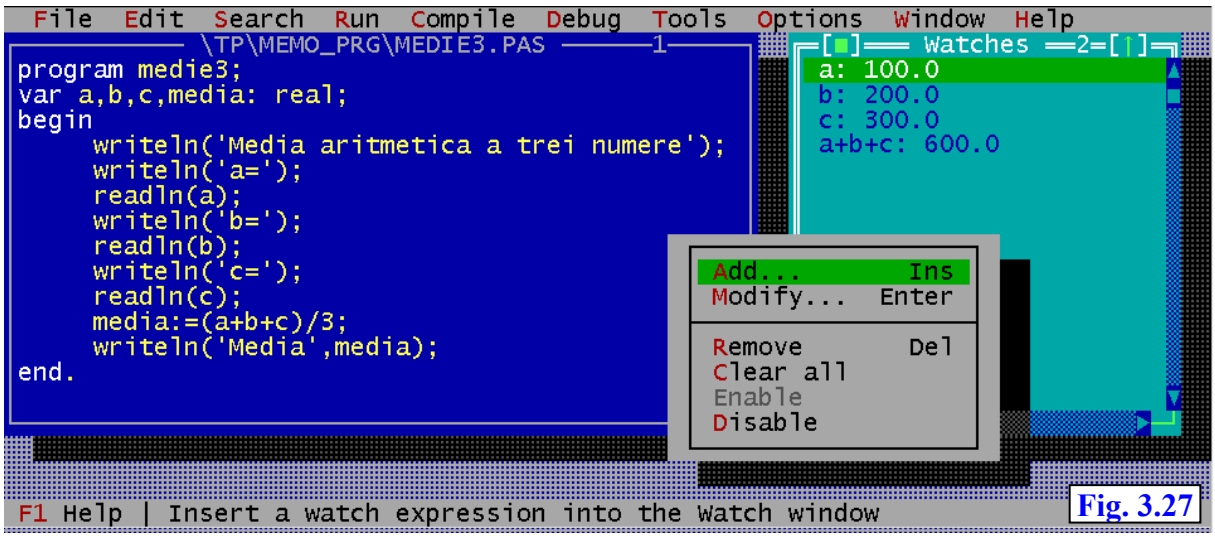


Fig. 3.27

Expresiile sau variabilele ale căror valori se doresc a fi supravegheate trebuie introduse prin denumire sau expresie matematică în lista din fereastra **Watches**. Fereastra dispune şi de un meniu asociat, fig. 3.27, care poate fi activat prin buton dreapta mouse pe fereastră sau combinaţia de taste **Alt+F10** şi care constă în următoarele opţiuni:

- opţiunea **Add** sau tasta **Ins** (dacă fereastra **Watches** este activă) - activează fereastra **Add Watch**, fig. 3.28, care permite adăugarea unei variabile sau expresii în vederea supravegherii, listă care dispune şi de istoricul **[↓]** pentru a putea reselecta o variabilă sau expresie introdusă anterior şi eliminată din listă; în câmpul **Watch expression** se introduce numele variabilei sau expresia de supravegheat şi se confirmă pe butonul **OK** (ceea ce va avea ca efect adăugarea variabilei sau expresiei în lista din fereastra **Watches**) sau se anulează supravegherea prin butonul **Cancel**;

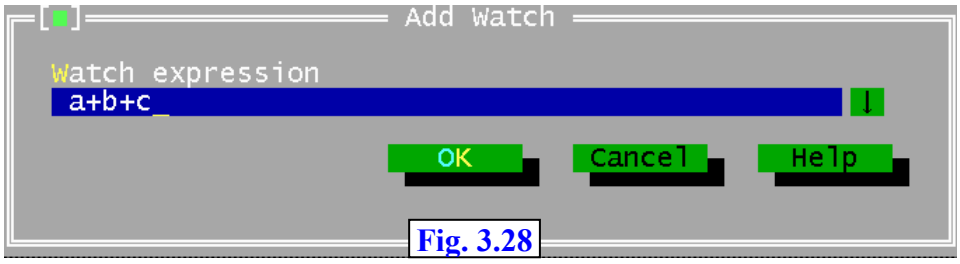


Fig. 3.28

- opţiunea **Modify** sau tasta **Enter** pe numele variabilei - reactivează fereastra **Add Watch**, fig. 3.28, care permite modificarea unei variabile sau expresii anterior introduse; acelaşi efect se obţine şi prin dublu click stânga pe numele variabilei sau expresiei;
- opţiunea **Remove** sau tasta **Del** pe numele variabilei – permite eliminarea, fără confirmare, a unei variabile sau expresii din lista de supraveghere;
- opţiunea **Clear all**– permite eliminarea, cu confirmare, a tuturor variabilelor sau expresiilor existente în lista de supraveghere;
- opţiunile **Enable** respectiv **Disable** – permit activarea sau dezactivarea supravegherii unei variabile sau expresii; dezactivarea este evidenţiată prin cuvântul **<disabled>** existent în dreptul variabilei sau expresiei, fig. 3.29.

Opţiunea **Output** din meniul **Debug** sau combinaţia de taste **Ctrl+F3** afişează fereastra de dialog **Output**, fig. 3.29, care permite vizualizarea textelor generate de linia de comandă DOS şi de programul în curs de execuţie. Nu pot fi vizualizate informaţii grafice în această fereastră. Informaţiile grafice pot fi urmărite doar în fereastra **User Screen**. Fereastra îşi dovedeşte utilitatea mai ales la depanarea programelor, deoarece se pot urmări simultan programul sursă în fereastra sa de editare, cât şi rezultatele acestuia.

Opţiunea **User Screen** din meniul **Debug** sau combinaţia de taste **Alt+F5** afişează un întreg ecran cu rezultate textuale şi grafice ale programului curent executat. Ieşirea din acest ecran şi revenirea în mediul de programare TurboPascal se face prin apăsarea unei taste arbitrare. Această fereastră nu poate fi redimensionată sau deplasată.

Opţiunea **Evaluate / Modify** din meniul **Debug** sau combinaţia de taste **Ctrl+F4** activează fereastra de dialog **Evaluate and Modify**, fig. 3.30, care permite evaluarea şi vizualizarea valorii unor expresii şi variabile precum şi modificarea valorii unor variabile simple. Operaţia se aplică la sfârşitul execuţiei programului sau în momentul întreruperii acestuia datorită punctelor de întrerupere. Variabila sau expresia care se doreşte a se evalua se

introduce în câmpul **Expression**, după care se acționează butonul **Evaluate**, valoarea rezultată fiind afișată în câmpul **Result**.

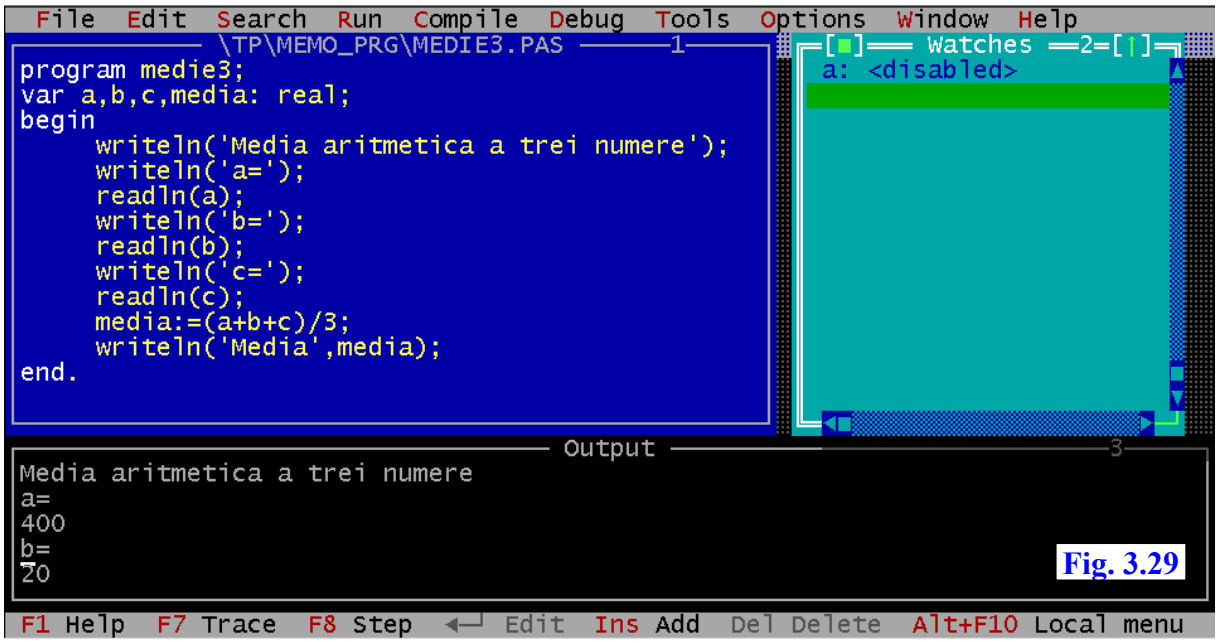


Fig. 3.29

Câmpul **New value** permite introducerea unei noi valori, dacă conținutul câmpului **Expression** corespunde unei variabile simple. Atribuirea acestei noi valori se face prin butonul **Modify**.

Fereastra **Evaluate and Modify** poate opera ca un calculator de buzunar, situație în care în câmpul **Expression** se pot introduce valori constante; exemplu 2*5+72.

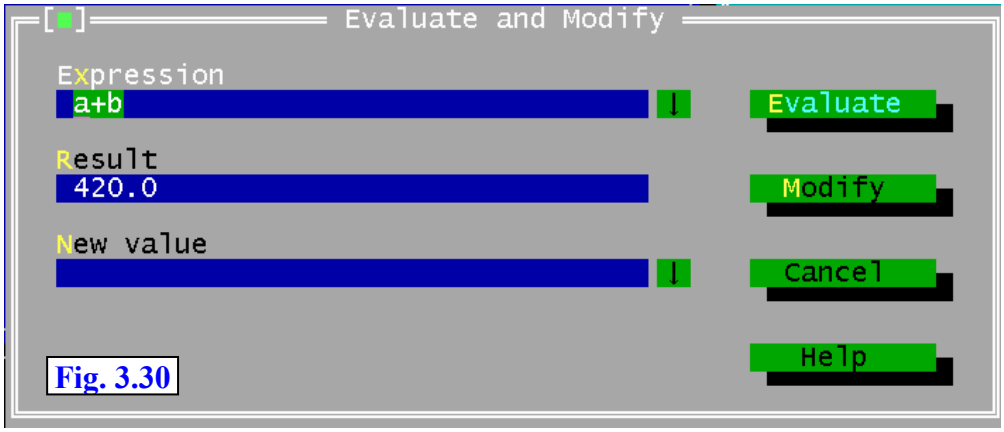


Fig. 3.30

Opțiunea **Add watch** din meniul **Debug** sau combinația de taste **Ctrl+F7** afișează fereastra de dialog **Add watch**, fig. 3.28, care permite definirea unor variabile sau expresii în vederea supravegherii valorii lor. Anterior definirii de variabile sau expresii pentru supraveghere, programul trebuie compilat, pentru ca identificatorii din program să devină cunoscuți mediului TurboPascal, în caz contrar se afișează mesaj de eroare "**Unknown**

identifier". Valorile vor fi alocate numai în timpul execuţiei, iar până în acel moment fereastra **Watches** va afişa mesajul "**Cannot access this symbol**".

Opţiunea **Add breakpoint** din meniul **Debug** afişează fereastra de dialog **Add breakpoint**, similară ca formă, conţinut şi funcţii cu fereastra de dialog **Edit breakpoint**, fig. 3.26, singura diferenţă fiind titlul ferestrei. Scopul ferestrei este specificarea punctelor noi de întrerupere şi a condiţiilor de oprire, cu rol de oprire a execuţiei programului, pentru a putea efectua următoarele acţiuni:

- se pot efectua evaluări şi modificări de valori prin comanda, **Evaluate and Modify (Ctrl+F4)**;
- se poate impune execuţia programului pas cu pas, prin comanda **Trace into** (**F7**) şi **Step Over** (**F8**) din meniul **Run**;
- prin deplasarea cursorului din locaţia punctului de întrerupere la o altă poziţie, se poate impune execuţia programului până la noua poziţie, prin comanda **Go to cursor** (**F4**) din meniul **Run**.

3.7.7 Meniul Tools

Meniul **Tools** (fig. 3.31) oferă posibilitatea lansării în execuţie a altor programe externe mediului de programare TurboPascal.

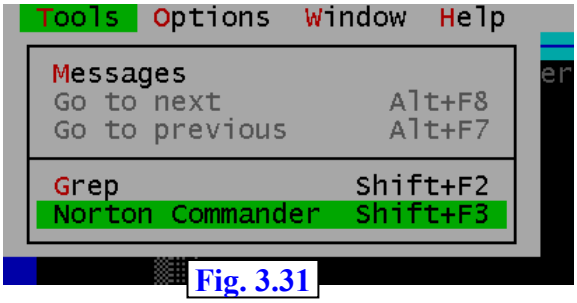


Fig. 3.31

Opţiunea **Messages** din meniul **Tools** afişează fereastra de dialog **Messages**, care permite vizualizarea diferitelor mesaje generate de programele externe către mediul de programare TurboPascal. Parcurgerea acestor mesaje se poate face linie cu linie, prin trecere la următorul mesaj, prin opţiunea de meniu **Go to next** din meniul **Tools** sau combinaţia de taste **Alt+F8** sau prin trecere la anteriorul mesaj, prin opţiunea de meniu **Go to previous** din meniul **Tools** sau combinaţia de taste **Alt+F7**.

Introducerea de noi opţiuni de meniu pentru lansare în execuţie de programe externe se poate realiza prin opţiunea de meniu **Tools** din bara de meniu **Options**. Numele programelor ce pot fi lansate în execuţie au regim identic cu al opţiunilor de meniu; de exemplu, utilitarul **Grep** permite căutarea diferitelor cuvinte în fişiere text, iar programul **Norton Commander** este un manager de fişiere.

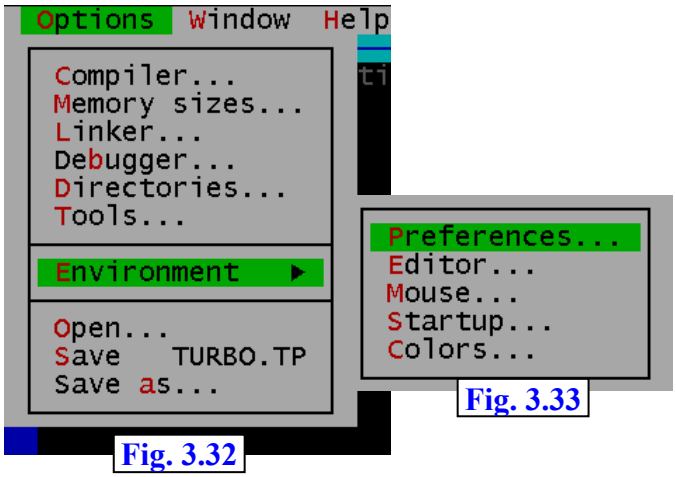


Fig. 3.32 Fig. 3.33

3.7.8 Meniul Options

Meniul **Options** (fig. 3.32, fig. 3.33) oferă posibilitatea configurării mediului de programare TurboPascal, prin specificarea unor opţiuni specifice.

3.7.9 Meniul Window

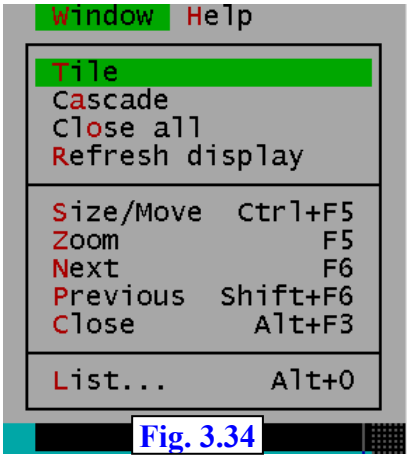


Fig. 3.34

Meniul **Window** (fig. 3.34) oferă manipulării ferestrelor de editare şi ale sistemului, prin deschiderea, închiderea dispunerea şi listarea acestora.

Opţiunea **Tile** din meniul **Window** provoacă dispunerea ferestrelor sub formă de "*cărămidă*", astfel încât să fie toate vizibile şi dispuse alăturat una lângă alta sau una sub alta.

Opţiunea **Cascade** din meniul **Window** provoacă dispunerea ferestrelor sub formă de "*cascadă*", astfel încât să fie dispuse suprapus una peste alta, fereastra activă fiind complet vizibilă şi ocupând întreaga zonă disponibilă, celelalte fiind aranjate dedesubt, fiind vizibilă doar bara de titlu a lor.

Opţiunea **Close all** din meniul **Window** provoacă închiderea tuturor ferestrelor deschise, cu confirmarea salvării modificărilor.

Opţiunea **Refresh display** din meniul **Window** provoacă restabilirea ecranului mediului de programare, situaţie care poate fi necesară atunci când acesta este suprascris de un program lansat în execuţie.

Opţiunea **Size/Move** din meniul **Window** sau combinaţia de taste **Ctrl+F5** oferă posibilitatea redimensionării şi deplasării ferestrei active. Redimensionarea se realizează prin acţionarea simultană a tastelor direcţionale împreună cu **Shift**, iar deplasarea numai prin acţionarea tastelor direcţionale, iar finalizarea acţiunii se obţine prin apăsarea tastei **Enter**. Există o dimensiune minim admisă a unei ferestre: lăţimea egală cu 3 linii plus chenarul acesteia, iar lungimea să poată afişa numele ferestrei. Dimensiunea maximă poate coincide cu cea a zonei de lucru.

Opţiunea **Zoom** din meniul **Window** sau tasta **F5** oferă posibilitatea maximizării (dispunerii pe întreaga suprafaţă a zonei de lucru) ferestrei active sau revenirii la dimensiunile de dinaintea maximizării.

Opţiunea **Next** din meniul **Window** sau tasta **F6** oferă posibilitatea activării următoarei ferestre din lista celor deschise la un moment dat. Acţiunea are un efect circular.

Opţiunea **Previous** din meniul **Window** sau tasta **Shift+F6** oferă posibilitatea activării anterioarei ferestre din lista celor deschise la un moment dat. Acţiunea are un efect circular.

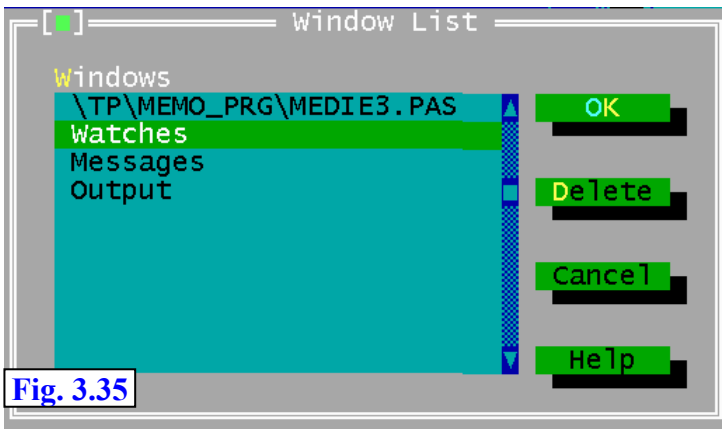


Fig. 3.35

Opţiunea **Close** din meniul **Window** sau tasta **Alt+F3** provoacă închiderea ferestrei active, cu confirmarea salvării modificărilor.

Opţiunea din meniul **Window List** sau tasta **Alt+0** provoacă afişarea ferestrei de dialog **Window list**, fig. 3.35, care conţine lista tuturor ferestrelor deschise la

un moment dat. Butonul de comandă **Delete** are ca efect închiderea ferestrei selectate în listă, cu confirmarea de salvare a modificărilor.

3.7.10 Meniul Help

Meniul **Help** (fig. 3.36) oferă posibilitatea accesului la documentația însoțitoare a mediului de programare TurboPascal. Documentația este furnizată în limba engleză şi constituie un ajutor prețios şi rapid în utilizarea mediului de programare.

Opțiunea **Contents** din meniul **Help** activează fereastra **Help**, fig. 3.37, care prezintă sumar capitolele documentației. Prin deplasarea cursorului pe capitolul dorit şi apăsarea tastei **Enter** sau prin dublu click stânga mouse se poate selecta capitolul dorit în vederea consultării acestuia.

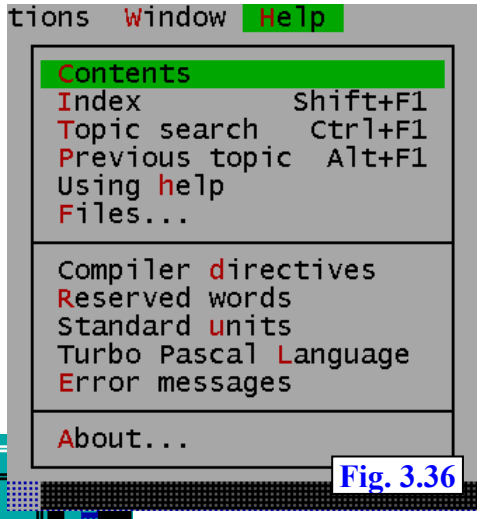


Fig. 3.36

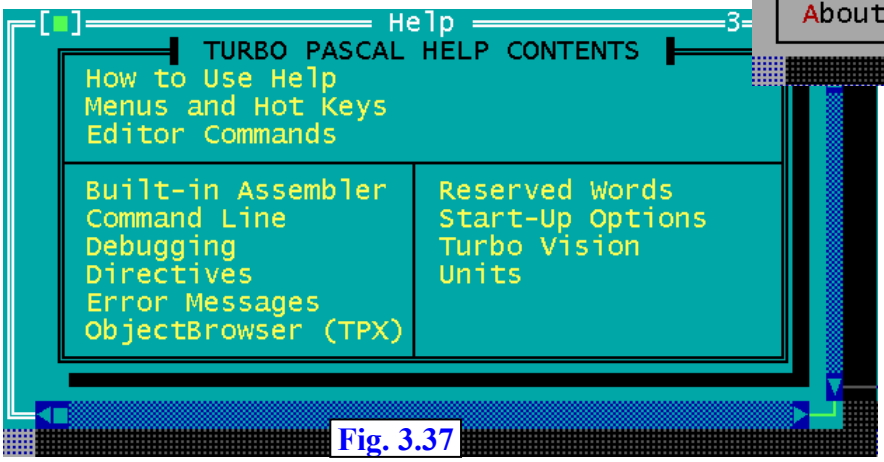


Fig. 3.37

Opțiunea **Index** din meniul **Help** sau combinația de taste **Shift + F1** activează fereastra **Help**, fig. 3.38, care oferă indexul noțiunilor ordonate alfabetic. Prin deplasarea cursorului pe capitolul dorit şi apăsarea tastei **Enter** sau prin dublu click stânga mouse se poate selecta indexul dorit în vederea consultării.

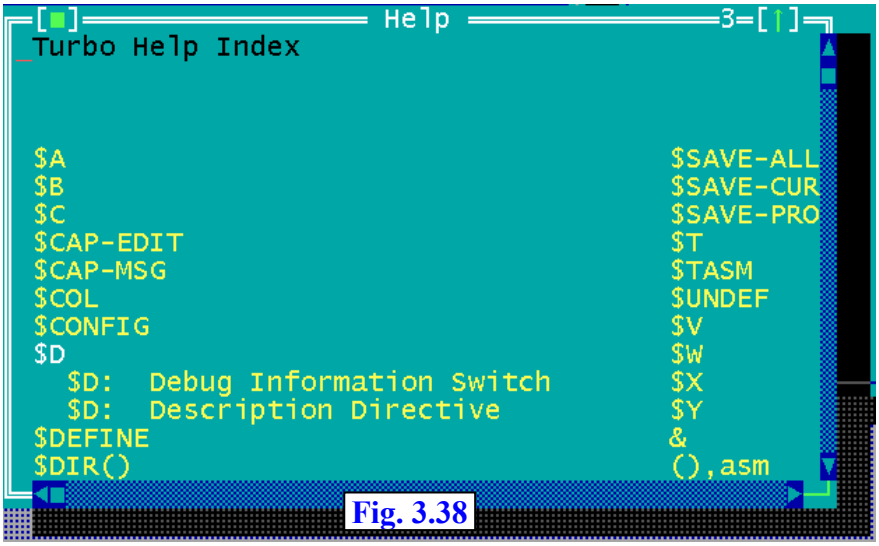


Fig. 3.38

Opțiunea **Topic search** din meniul **Help** sau combinația de taste **Ctrl + F1** oferă posibilitatea accesării rapide a informațiilor referitoare la sintaxa limbajului (cuvinte rezervate, funcții şi proceduri, tipuri de date, **unit**-uri, variabile, etc.). În fereastra de editare se poziționează cursorul pe cuvântul pentru care se dorește obținerea de informații şi se apasă combinația de taste **Ctrl + F1** sau se selectează opțiunea **Topic search** din meniul **Help**, ceea ce va provoca apariția ferestrei de informații asociate. Dacă cuvântul nu este găsit în baza de date, atunci se vor afişa informații referitoare la cel mai apropiat cuvânt din index.

Opțiunea **Previous topic** din meniul **Help** sau combinația de taste **Alt + F1** oferă posibilitatea revederii în sens invers a ultimelor 20 de subiecte **Help** accesate.

Opțiunea **Using help** din meniul **Help** oferă informații referitoare la modul de accesare a sistemului de **Help**.

Opțiunea **Files** din meniul **Help** deschide fereastra de dialog **Install Help Files**, fig. 3.39, pentru adăugarea de noi fişiere de **Help** (prin butonul **New**), eliminarea acestora din listă (prin butonul **Delete**). Declanşarea butonului **New** provoacă apariția ferestrei **Help Files**, fig. 3.40, prin care se poate specifica numele noului fişier de tip **Help** care se dorește a fi adăugat.

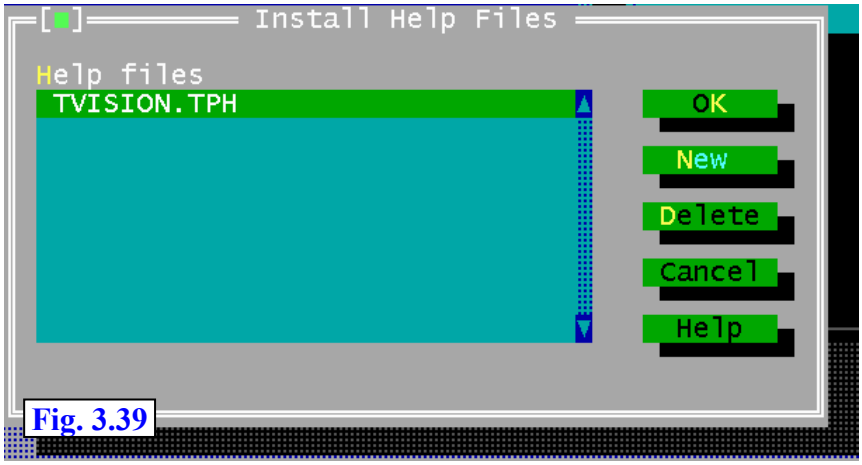


Fig. 3.39

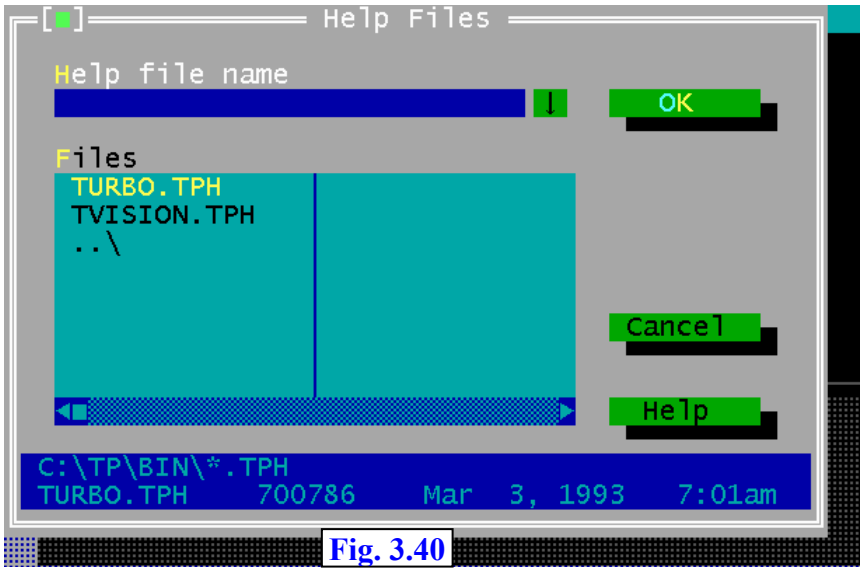


Fig. 3.40

Restul opțiunilor din meniul **Help**, oferă informații specifice referitoare la următoarele subiecte:

- **Compiler directives** – directive de compilare;
- **Reserved words** – cuvinte rezervate ale limbajului;
- **Standard units** – **unit**-uri standard;
- **Turbo Pascal Language** – limbajul TurboPascal;
- **Error messages** – lista mesajelor de eroare.

Opțiunea **About** din meniul **Help** deschide fereastra de dialog **About**, fig. 3.41, prin care se afișează informații referitoare la numele şi versiunea programului, anul fabricației şi numele producătorului.

Fig. 3.41

Meniul **Help** dispune şi de un submeniu asociat, fig. 3.42, activabil prin combinația de taste **Alt+F10** sau prin click stânga mouse şi disponibil atunci când este activă o fereastră de informații.

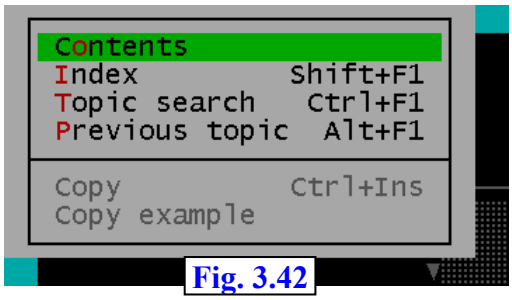


Fig. 3.42

Primele patru opțiuni sunt identice funcțional cu opțiunile corespondente din meniul principal **Help**.

Opțiunea **Copy** sau combinația de taste **Ctrl+Ins** permite copierea în **Clipboard** a informațiilor selectate într-un bloc (& 3.6), de unde pot fi recopiate într-o fereastră de editare, prin comanda **Paste** a meniului **Edit** (& 3.6), conform metodologiei detaliate în & 3.6.

Opțiunea Copy example permite copierea în **Clipboard** a exemplelor asociate subprogramelor, de unde pot fi recopiate într-o fereastră de editare, prin comanda **Paste** a meniului **Edit** (& 3.6), conform metodologiei detaliate în & 3.6.

4. CONCEPTE FUNDAMENTALE ALE LIMBAJULUI TurboPascal

4.1 VOCABULAR, DEFINIŢII, NOTAŢII

Vocabularul limbajului Pascal este format din *simboluri de bază*, clasificabile în:

- **litere** – literele mari şi mici ale alfabetului englez, în număr de 26:
  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
  a b c d e f g h i j k l m n o p q r s t u v w x y z

Se observă inexistența literele specifice vocabularului românesc (ă, â, î, ş, ţ).

- **cifre** – cifrele arabe 0, 1, 2, 3, 4, 5 ,6, 7, 8, 9;
- **simboluri speciale** care pot fi **operatori** şi **delimitatori**:

  \+ - * / = <> < > <= >= ( ) [ ] { } := . , ; : ^

- separatori spaţiul, tasta **ENTER**, tasta **TAB**, comentariile**.**

Pentru unele simboluri sunt prevăzute alternative de înlocuire, pentru a putea fi accesate pe tastaturi în care nu sunt incluse: de exemplu combinaţia de taste "(*" respectiv "*)" este echivalentul combinaţiei "{" respectiv "}", iar combinaţia de taste "(." respectiv ".)" este echivalentul combinaţiei "[" respectiv "]". Dacă un simbol special este reprezentat de 2 caractere, între acestea nu trebuie să apară spaţiu.

Cuvintele cheie sunt cuvinte rezervate ale limbajului Pascal, cu semnificaţie fixată şi care nu pot fi folosite decât în contextul stabilit de limbaj. Ele nu pot fi redefinite şi pot fi scrise cu majuscule sau cu minuscule. Lista cuvintelor cheie ale limbajului este următoarea:

And	Do	If	Nil	Repeat	Until
Asm	Downto	Implementation	Not	Set	Uses
Array	Else	In	Object	Shl	Var
Begin	End	Inherited	Of	Shr	While
Case	Exports	Inline	Or	String	With
Const	File	Interface	Packed	Then	Xor
Constructor	For	Label	Procedure	To	
Destructor	Function	Library	Program	Type	
Div	Goto	Mod	Record	Unit	

Programul Pascal este format din succesiuni de instrucţiuni scrise pe linii, instrucţiunile fiind formate din *unităţi sintactice* cu semnificaţii specifice, despărţite prin separatori (spaţii sau alte simboluri speciale). Unităţile sintactice sunt clasificabile în patru categorii: *identificatori, numere, şiruri de caractere, comentarii*.

Identificatorii reprezintă modalitatea de denumire a constantelor, variabilelor, tipurilor, procedurilor, funcțiilor. Un identificator este o secvență de caractere ce începe cu o literă sau caracterul "_" și poate fi urmat de o succesiune de maxim 63 litere sau cifre zecimale. Nu se face diferență între litere mari și mici. În TurboPascal există două tipuri de identificatori:

- *identificatorii definiți de către programator* - care se recomandă să reflecte prin denumire conținutul sau acțiunea asociată acestuia, pentru a fi mai ușoară identificarea funcției sale în cadrul programului;
- *identificatori standard sau predefiniți* – exemplu: **sin**, **cos**, care, spre deosebire de cuvintele cheie, pot fi redefiniți, deși aceasta nu este recomandabil.

Reguli impuse identificatorilor:

- *declarare* - identificatorii trebuie declarați, înainte de a fi utilizați, în caz contrar se semnalează eroare "**Error 3: Unknown identifier.**";
- *unicitate* – un identificator trebuie să reprezinte o singură entitate TurboPascal; deci două entități diferite nu pot avea aceeași denumire;
- *valabilitate* – orice identificator are valabilitate numai în blocul în care a fost definit, precum și în blocurile incluse acestuia;
- *corectitudine* - nu pot conține caracterul spațiu, indici matematici sau simboluri speciale.

Exemple de identificatori corecți și incorect definiți sunt date în tabelul 4.1.

Tabel 4.1

Identificatori corecți	**Identificatori incorecți**	
Identificator	**Identificator**	**Tipul erorii**
Alfa	2s	Începe cu o cifră
A1	Al fa	Conține caracterul spațiu
A1b	Begin	Coincide cu un cuvânt rezervat
_alfa	Alfa-beα	Conține caracterul – și α
_alfa234	Test.2	Conține caracterul .

Numerele pot fi de clasificate în două categorii:

- *numere întregi* – care corespund numerelor întregi din matematică, fiind o succesiune de cifre precedate de semnul "+", pentru numere pozitive, respectiv de semnul "-", pentru numere negative;
- *numere reale* – care corespund numerelor raționale din matematică și care pot fi exprimate, cu număr finit de zecimale, în două formate:
  - *format zecimal* – exprimate prin:
    - semnul "+", pentru numere pozitive, respectiv de semnul "-", pentru numere negative;
    - succesiune finită de cifre - constituie partea întreagă a numărului;
    - punctul zecimal – "." constituie delimitatorul între partea întreagă și partea zecimală a numărului
    - succesiune finită de cifre - constituie partea zecimală a numărului;

- *format exponențial* – exprimate prin:
  - *mantisa* - un număr întreg sau fracționar, pozitiv sau negativ;
  - *litera E* - care reprezintă baza 10;
  - *exponent* - care este un număr întreg pozitiv sau negativ;

iar valoarea numărului exprimat în format exponențial se obține astfel: valoarea mantisei înmulțită cu 10 ridicată la puterea exponentului.

Exemple de numere în cele trei formate sunt date în tabelul 4.2.

Tabel 4.2

Numere întregi	**Numere reale**		
	Format zecimal	**Format exponențial**	**Reprezentare echivalentă numărului real**
234	123.54	31415E-4	3.1415
+234	-123.54	314.15E-2	
-234	0.124578	0.31415E+1	
120	-0.5234	0.31415E1	

Datorită caracterului finit al memoriei calculatorului numărul de zecimale memorabil este finit, deci exprimarea numerelor iraționale se face cu aproximație. Exemple: numărul "e" sau numărul "π", care sunt numere iraționale.

Şirurile de caractere sau **mesajele** sunt secvențe de caractere imprimabile incluse între două simboluri apostrof. Pentru a include în şir chiar simbolul apostrof, acesta trebuie dublat. Valoarea şirului este succesiunea de caractere dintre apostroafe, mai puțin apostroafele. Un şir de caractere care nu include nici un alt caracter între apostroafe este *şirul vid*. Exemple de şiruri:

'Acesta este un şir' 'Acest şir include apostroful '' ca şi caracter al şirului'

Comentariul este o porțiune de text cuprinsă între caracterele "**{**" respectiv "**}**" sau "**(***" respectiv "***)**", care este ignorată de compilator, cu rol de uşurare a înțelegerii programelor. Comentariile pot fi introduse oriunde în cadrul programului: la început sau sfârşit, în fața sau la sfârşitul instrucțiunilor. Pentru programe de mare complexitate se recomandă plasarea de comentarii referitoare la algoritm şi modul de rezolvare, pentru a facilita rememorarea acestora la analiza unui program după o perioadă lungă de timp scursă de la elaborarea sa. Comentariu are rol de separator.

Constantele reprezintă date (informații) a căror valoare nu se modifică pe parcursul execuției programului. Tipuri de constante: întregi, reale, şiruri de caractere, constante desemnate prin identificatori.

Variabile reprezintă date (informații) a căror valoare se poate modifica pe parcursul execuției programului. Orice variabilă primeşte un nume simbolic (identificator) şi i se poate asocia o valoare sau un şir de valori aparținând unei mulțimi date. Mulțimea de valori căreia îi aparține variabila se numeşte tipul variabilei, care rămâne neschimbat pe tot parcursul execuției programului.

4.2 TIPURI DE DATE

Noțiunea de date reprezintă informațiile furnizate unui program în scopul obținerii rezultatelor şi sunt concretizate prin constante şi variabile.

Tipul de date definește:

- mulțimea valorilor pe care le poate accesa o variabilă sau constantă;
- mulțimea de operații care pot fi efectuate.

Valorile unui tip de date sunt accesate prin intermediul constantelor şi variabilelor. În TurboPascal există următoarele tipuri de date: simple (elementare), compuse (structurate), de referință (pointeri).

Tipurile de date standard sunt tipuri de date predefinite, pe care programatorul le utilizează, fără a fi nevoie să le mai definească. Tipurile standard simple de date utilizate de limbajul TurboPascal sunt: **INTEGER** – constituit din numere întregi; **BOOLEAN** – valorile **TRUE** (adevărat) sau **FALSE** (fals); **REAL** – numere cu parte zecimală; **CHAR** – un caracter din setul de caractere.

Tipurile ordinale reprezintă mulțimi finite şi ordonate de valori. Aceasta înseamnă că fiecare element are asociat un număr de ordine, iar ordonarea este considerată în raport cu acest număr de ordine. Asupra acestora se pot aplica următoarele funcții:

- **ORD()** – obținerea numărului de ordine al unui element;
- **SUCC()** – accesarea elementului cu numărul de ordine următor;
- **PRED()** – accesarea elementului cu numărul de ordine anterior.

Prin tip **simplu** de date se înțelege un tip de date ordinal sau real.

Prin tip **compus** de date se înțelege tipul de date şir de caractere, tablou, articol sau obiect.

În continuare vor fi descrise tipurile de date standard, precizând limitele maximale şi operațiile posibile de aplicat.

4.2.1 Tipul INTEGER

Tipul **INTEGER** permite reprezentarea, memorarea şi prelucrarea numerelor întregi. Limbajul Pascal nu permite operarea valori infinite, această restricție fiind concretizată prin identificatorii constantelor **MaxInt** şi **MaxLongInt**, a căror valori reprezintă cel mai mare număr ce poate fi atribuit datelor de tip **INTEGER** respectiv **LONGINT**. Aceasta este 32567 pentru date de tip **INTEGER** respectiv 2147483647 pentru date de tip **LONGINT**.

Domeniul datelor de tip întreg este dat în tabelul 4.3.

Tabel 4.3

Tip	Domeniu		Format
SHORTINT	$-2^7 \dots 2^7-1$	-128 ... 127	8 biți cu semn
INTEGER	$-2^{15} \dots 2^{15}-1$	-32768 ... 32567	16 biți cu semn
LONGINT	$-2^{31} \dots 2^{31}-1$	-2147483648 ... 2147483647	32 biți cu semn
BYTE	$0 \dots 2^8-1$	0 ... 255	8 biți fără semn
WORD	$0 \dots 2^{16}-1$	0 ... 65535	16 biți fără semn

Operatori aritmetici şi funcţiile standard ce operează asupra numerelor întregi sunt date în tabelul 4.4.

Tabel 4.4

Operatori aritmetici		
Operator	**Semnificaţie**	**Explicaţie**
*	Înmulţire	
DIV	Împărţire întreagă	Împărţire şi trunchiere rezultat la parte întreagă
/	Împărţire reală	Câtul real al împărţirii a două numere întregi
MOD	Modulo	Restul împărţirii a două numere întregi
+	Adunare	
-	Scădere	
Funcţiile standard		
ABS(x)	Valoarea absolută	
SQR(x)	Ridicare la pătrat	

Exemple: (+7) **DIV** (+2) => +3 (- 7) **DIV** (+2) => -3
(+18) **MOD** (+5) => +3 (-17) **MOD** (+4) => -1

Prioritatea operatorilor este: * / **DIV** **MOD** + -
Paritatea unui număr întreg "**m**" se poate exprima prin **m MOD 2 = 0**.
Divizibilitatea numărului întreg "**m**" cu numărul întreg "**n**" se poate exprima prin **m MOD n = 0**.

4.2.2 Tipul logic (BOOLEAN)

Tipul **BOOLEAN** este format din mulţimea valorilor logice **TRUE** şi **FALSE**. Cel mai adesea, valoarea unei variabile booleene rezultă în urma operaţiei de comparaţie. Operatorii de comparaţie (relaţionali) sunt prezentaţi în tabelul 4.5.

Tabel 4.5

Operator	**Semnificatie**	**Operator**	**Semnificatie**
=	egal cu	>	mai mare decat
<>	diferit de	<=	mai mic sau egal cu
<	mai mic decat	>=	mai mare sau egal cu

Operatorii logici sunt :

- **NOT** – negare logică;
- **AND** - conjuncţia logică ;
- **OR** – disjuncţia logică.

Prioritatea operatorilor este: **NOT AND OR**.
Tabelele de adevăr ale acestor operatori sunt prezentaţi în tabel 4.6 :

Tabel 4.6

AND	False	True
False	False	False
TRUE	False	**True**

OR	False	True
False	False	**True**
True	**True**	**True**

NOT	False	True
	True	False

Funcţii care returnează valori logice:

- **ODD**(x) – rezultatul este **TRUE** dacă x este un întreg impar;
- **EOLN(f) -** rezultatul este **TRUE** dacă s-a ajuns la sfârşitul unei linii a fişierului "f";
- **EOF(f) -** rezultatul este **TRUE** dacă s-a ajuns la sfârşitul fişierului "f".

4.2.3 Tipul REAL

Tipul real este reprezentat de o submulţime a mulţimii numerelor raţionale, reprezentând aproximaţii a numerelor reale. Există deci o limită a *mărimii* numerelor reale şi o limită a *preciziei* acestora. Domeniul datelor de tip real este prezentat în tabelul 4.7.

Tabel 4.7

Tip	Domeniu	Format	Precizie
REAL	2.9E-39 ... 1.7E38	6 octeţi	11-12
SINGLE	1.5E-45 ... 3.4E38	4 octeţi	7-8
DOUBLE	5.0E+324 ... 1.7E308	8 octeţi	15-16
EXTENDED	3.4E-4932 ... 1.1E4932	10 octeţi	19-20
COMP	0 ... 2^{63}-1	8 octeţi	19-20
OBS: În cazul **COMP** datele sunt de tip întregi, dar în cazul lor acţionează operaţiile cu tipuri reale.			

Simbolul separatorului între partea întreagă şi partea zecimală este ".". Operatorii aritmetici şi funcţiile standard ce operează asupra numerelor reale sunt date în tabelul 4.8.

Tabelul 4.8

Operatori aritmetici			
Operator	**Semnificaţie**	**Explicaţie**	
*	Înmulţire	+	Adunare
/	Împărţire	-	Scădere
OBS: Prioritatea operatorilor este * / + -			
Funcţiile standard			
ABS(x)	Valoarea absolută		
SQR(x)	Ridicare la pătrat		
SIN(x), COS(x)	sinus respectiv cosinus de argument exprimat în radiani		
ARCTAN(x)	arctangenta unghiului x exprimat în radiani		
LN(x)	logaritm natural		
EXP(x)	funcţia exponenţială **EXP**(x)=e^x, unde e=2.71828...		
SQRT(x)	radical		
INT(x)	parte întreagă a numărului x, generează un număr real		
FRAC(x)	parte fracţionară a numărului x		
TRUNC(X)	Partea întreagă a numărului real X, partea fracţionară fiind îndepărtată, generează un număr întreg de tip LongInt		
ROUND(X)	Rotunjirea numărului real X la cel mai apropiat întreg, generează un număr întreg de tip LongInt4		
OBS: pentru X >= 0 avem **ROUND**(X) = **TRUNC**(X+0.5) pentru X < 0 avem **ROUND**(X) = **TRUNC**(X-0.5)			

4.2.4 Tipul CHAR

Reprezintă un caracter din mulțimea caracterelor, inclus între apostroafe. Caracterul apostrof însuşi trebuie dublat pentru a putea fi inclus în set. Convenția de încadrare a datelor caracter între apostroafe se foloseşte numai la scrierea sursei programului, dar nu şi în timpul execuției programului la introducerea acestora de la tastatură. Codul **ASCII** codifică seturile de caractere, fiecare caracter având asociat un număr de ordine, cuprins între 0 şi 255 (vezi Anexa 3).

Funcțiile standard ce se aplică datelor de tip caracter sunt:

- **ORD(c)** – numărul de ordine al caracterului "c" în cadrul setului de caractere, **ORD**('A')=65;
- **CHR(i)** – furnizează caracterul al cărui număr de ordine este "i"; exemplu **CHR**(65)='A';
- **PRED(c)** – funcția predecesor, furnizează ca rezultat caracterul precedent caracterului "c"; exemplu: **PRED**('B')='A';
- **SUCC(c)** – funcția succesor, furnizează ca rezultat caracterul următor caracterului "c"; exemplu: **SUCC**('B')='C'.

4.3 EXPRESII

Expresiile reprezintă "*echivalentul computerizat*" al formulelor matematice.

Expresiile reprezintă combinație de operanzi (constante, variabile, apel de funcții) legate prin operatori (simbolurile operațiilor). Expresiile se scriu linearizat (pe aceeaşi linie).

Prin *evaluarea* unei expresii se înțelege efectuarea operațiilor care intervin în expresie şi generarea unui rezultat. Regulile de evaluare sunt cele obişnuite din matematică:

- prioritatea operatorilor;
- evaluarea de la stânga la dreapta în cazul priorităților egale;
- expresiile din paranteze sunt evaluate ca şi cum ar fi simpli operanzi.

În cadrul expresiilor, prioritatea tuturor operatorilor prezentați anterior sunt prezentați în continuare, cu observația că aceasta se poate modifica prin intermediul parantezelor.

1. **NOT**
2. *** / AND DIV MOD**
3. **+ - OR**
4. **= <> <= >= < >.**

4.4 STRUCTURA GENERALĂ A UNUI PROGRAM PASCAL

Structura unui program TurboPascal conține următoarele componente, fig. 4.1:

- antetul programului
- clauza **USES**
- secțiunea declarativă
- secțiunea de instrucțiuni.

Prima linie reprezintă **antetul programului** – format din cuvântul cheie **PROGRAM** şi denumirea acestuia; această linie este opțională.

Clauza **USES** – este opțională şi specifică numele tuturor **unit**-urilor utilizate de program.

Secțiunea declarativă conține 6 secțiuni, grupate astfel:

1. **secțiunea de declarare a etichetelor** – prin cuvântul cheie **LABEL** se definesc etichetele utilizate în program de instrucțiunile de salt necondiționat **GOTO**;
2. **secțiunea de declarare a constantelor** – prin cuvântul cheie **CONST** se definesc constantele din program și se declară variabilele inițializate ale programului;
3. **secțiunea de declarare a tipurilor** – prin cuvântul cheie **TYPE** se definesc tipuri de date specifice utilizatorului (exceptând cele predefinite standard: integer, real, char, boolean);
4. **secțiunea de declarare a variabilelor** – prin cuvântul cheie **VAR** se definesc variabilele neinițializate ale programului și tipul acestora; variabile nedeclarate în program nu pot fi utilizate;
5. **secțiunea de declarare a funcțiilor** – permite definirea subprogramelor prin cuvintul cheie **FUNCTION**.
6. **secțiunea de declarare a procedurilor** – permite definirea subprogramelor prin cuvintul cheie **PROCEDURE**.

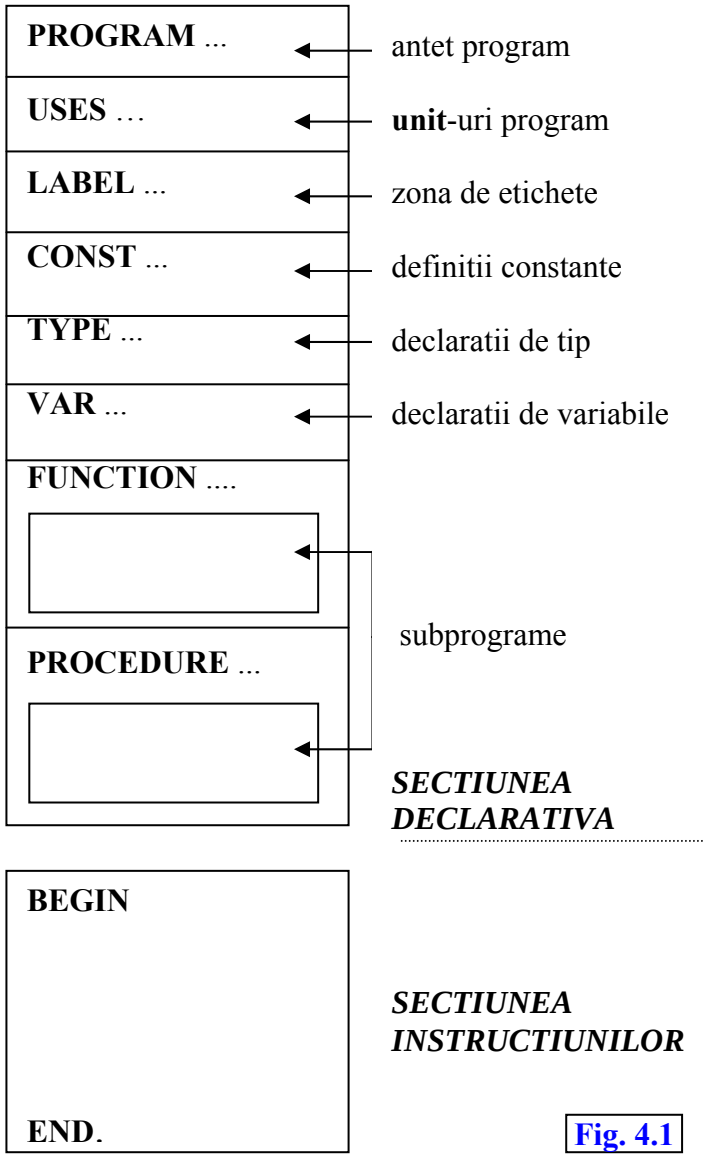


Fig. 4.1

Subprogramele **FUNCTION** sau **PROCEDURE** au structura similară celei din fig. 4.1.

Dacă programul nu necesită vreuna din secțiuni, atunci aceasta poate lipsi. Ordinea acestor secțiuni este arbitrară.

Secțiunea instrucțiunilor – este delimitată de cuvintele cheie **BEGIN** și **END** urmat de "**.**" și cuprinde toate instrucțiunile programului, inclusiv apelul de proceduri și funcții. Această secțiune este obligatorie. Un program nul, deci care nu face nimic arată astfel:

```
BEGIN

END.
```

4.5 INSTRUCȚIUNI TurboPascal

Instrucțiunile descriu partea algoritmică a unui program. Sintaxa instrucțiunilor și modul lor de asamblare în formarea programului este impusă de limbajul de programare. În limbajul de programare TurboPascal există două categorii de instrucțiuni:

- **instrucțiuni simple** (care nu conțin alte instrucțiuni):
  - instrucțiunea de atribuire
  - instrucțiunea de apel de procedură
  - instrucțiunea de salt necondiționat
  - instrucțiunea **INLINE**.
- **instrucțiuni structurate** (care sunt construite din alte instrucțiuni)
  - instrucțiunea compusă **BEGIN-END**
  - intrucțiunile repetitive **FOR**, **WHILE**, **REPEAT**
  - instrucțiunile condiționale **IF** și **CASE**
  - instrucțiunea **WITH**.

Separarea instrucțiunilor se realizează prin caracterul "**;**" . Acest caracter este destinat separării instrucțiunilor și nu terminării unei instrucțiuni.

Instrucțiunile pot fi prefixate de etichete, care pot fi referite prin instrucțiunea de salt necondiționat **GOTO**. Etichetele pot fi numere din domeniul 0 ... 9999 sau identificatori și trebuie declarate în mod obligatoriu prin secțiunea **LABEL**.

Pentru evidențierea restricțiilor impuse de limbaj, cuvintele cheie ale limbajului vor fi scrise cu litere mari, spre deosebire de entitățile definibile de către programator, care vor fi afișate cu litere mici. Această diferențiere nu este impusă de limbaj, ci constituie numai o metodă vizuală de atenționare utilizată în cadrul prezentei lucrări.

4.6 ETAPE ÎN PROGRAMAREA TurboPascal

Rezolvarea unei probleme utilizând programarea în limbajul TurboPascal impune parcurgerea următoarelor etape:

1. **Enunțul problemei** – problema poate rezulta dintr-o aplicație inginerească, din domeniu matematic sau alte domenii tehnice sau nu și trebuie precizată în mod explicit, astfel încât cerințele și obiectivele acesteia să fie definite în totalitate;
2. **Rezolvarea algoritmică a problemei** – desigur că rezolvarea unei probleme se poate realiza în mai multe moduri posibile, dintre care unele pot fi mai simple sau mai complicate; rezolvarea optimă constituie țelul care trebuie atins, care însă nu se poate obține decât printr-o experiență acumulată în timp; concretizarea metodei de rezolvare se poate realiza printr-o schemă logică, limbaj pseudocod sau alte metode de reprezentare a algoritmilor (cap. 2); desigur că gradul de detaliere a rezolvării în format algoritmic descrește, mergând chiar până la eliminarea completă a acestei etape, dar numai ca o consecință a experienței acumulate prin elaborarea de algoritmi; în faza incipientă de

abordarea a acestui domeniu această etapă este absolut obligatorie, pentru acomodarea intelectuală cu noile concepte specifice;

3. **Elaborarea programului în forma sursă** – etapa de transpunere a problemei într-o forma accesibilă calculatorului impune cunoaşterea unui limbaj de programare, în cazul de faţă TurboPascal; utilizând instrucţiunile acestui limbaj se generează un program, în forma sursă a sa, program constituit dintr-o secvenţă de instrucţiuni; în calculator programul se concretizează printr-un fişier, iar introducerea instrucţiunilor se realizează sub mediul de programare TurboPascal, cu ajutorul editorului de texte încorporat (& 3.6);
4. **Compilarea programului** – forma sursă a programului constituie un instrument de comunicare între utilizator şi ansamblul calculator - TurboPascal, care însă nu poate fi lansat în execuţie în această formă; programele pot fi executate numai în cod maşină; de aceea se impune etapa de compilare a acestora în format compatibil din punct de vedere a execuţiei (& 3.7.5); destinaţia compilării poate fi fixată atât în memorie, cât şi pe disk, în această ultimă situaţie rezultatul final al compilării va fi memorat într-un fişier cu extensia "**EXE**", comanda de compilare cea mai uzuală fiind combinaţia de taste **Alt + F9**, existând însă şi alte posibilităţi (& 3.7.5);
5. **Execuţia programului** – lansarea în execuţie a programului sub mediul TurboPascal este declanşată de combinaţia de taste **Ctrl+F9** (& 3.7.5); este momentul în care sunt citite sau introduse datele de intrare, aplicat algoritmul de rezolvare şi generate rezultate, sub forma afişării pe display, la imprimantă, salvate în fişier sau alte modalităţi concrete;
6. **Verificarea rezultatelor** – desigur că simpla generare a unui program nu constituie simpla garanţie a bunei sale funcţionări; din acest motiv această etapă este obligatorie, mai ales în programe de mare complexitate; scopul acestei etape este depistarea greşelilor, ceea ce nu întotdeauna este simplu sau rapid; mediul de programare ne pune la dispoziţie metode de verificare: execuţia pas cu pas a programelor (& 3.7.5), puncte de întrerupere (& 3.7.6), supravegherea valorii variabilelor în timpul execuţiei (& 3.7.6); însă aceste posibilităţi de depanare a programelor nu se referă şi la depistarea erorilor de logică în algoritm, care pot genera rezultate alterate sau chiar greşite fundamental, nefiind astfel respectată caracteristica de corectitudine a algoritmului (& 2.1); de asemenea, se recomandă rularea programului pe mai multe seturi de date de intrare pentru a verifica comportarea programului în diverse variante; există astfel trei tipuri de erori:
   - **erori de compilare** – sunt consecinţa erorilor lexicale sau sintactice (nerespectarea restricţiilor impuse ale limbajului); sunt cele mai uşor de depistat, fiind semnalate de către compilator (vezi Anexa 2);
   - **erori în execuţia programului** – provin din erori generate datorită calculelor; exemple clasice sunt: extragere de radical din număr negativ, împărţire cu 0;
   - **erori de logică sau de programare** – de exemplu înlocuirea neintenţionată a numelui unei variabile cu alta, poate provoca alterarea insesizabilă a rezultatelor finale; sunt cele mai greu de depistat.
7. **Corectarea erorilor şi finalizarea programelor** – desigur că scopul final al oricărui program este obţinerea unor date de ieşire corespunzătoare cerinţelor impuse, astfel încât în urma corecţiilor aplicate programul să poată fi executat pentru setul de date de intrare corect specificat.

5. INSTRUCȚIUNI DE BAZĂ

5.1 DECLARAREA VARIABILELOR ŞI CONSTANTELOR

Declararea tuturor constantelor şi variabilelor trebuie făcută înainte de utilizarea acestora. Fiecărei variabile i se alocă la compilare un spațiu de memorie dependent de tipul ei. Declarațiile de variabile şi constante sunt poziționate în secțiunea declarativă a programului (& 4.4).

Definiția de variabilă asociază unei locații de memorie un identificator şi un tip de date. Valorile tipului specificat vor fi memorate în locația respectivă.

Declararea variabilelor se face prin intermediul cuvântului cheie **VAR** sub forma :

```
VAR   v1, v2, v3: tip1;
      w1, w2, w3: tip2;
      .....
```

unde :

- **v1, v2, v3, w1, w2, w3** - reprezintă identificatori de variabile
- **tip1, tip2** - reprezintă tipul variabilei.

Exemplu:

```
VAR   a, b, c : REAL;
      i, j: INTEGER;
```

Identificatori succesivi de acelaşi tip sunt despărțiți prin separatorul ",".

Definiția unei constante introduce un identificator echivalent prin denumire unei valori constante.

Declararea constantelor se face prin intermediul cuvântului cheie **CONST** sub forma :

```
CONST  id_c1 = exp_c1;
       id_c2 = exp_c2;
       .....
```

unde :

- **id_c1** - este identificatorul asociat constantei **c1**
- **val_c1** - este o expresie sau valoare asociată constantei **c1**.

Exemplu:

```
CONST a = 15.5;
      b = 18;
      alfa = a*b;
```

Expresiile care intervin în definiția constantelor trebuie să fie evaluabile în timpul compilării (nu în timpul execuției).

La concepția programelor se recomandă definirea constantelor şi utilizarea acestora identificate prin denumire, în loc de a lucra direct cu valorile lor în interiorul instrucțiunilor, motivul fiind posibilitatea grupării inițiale a tuturor constantelor programului şi modificări minimale în codul programului la modificarea valorilor constantelor.

Limbajul TurboPascal permite şi declararea *variabilelor inițializate* care sunt variabile ale căror valori inițiale sunt cunoscute la intrarea în blocul în care sunt definite. *Declarația variabilelor inițializate specifică atât tipul variabilei, cât şi o constantă, cu care este inițializată variabila.*

Declararea acestor variabile se face în secțiunea de declarare a constantelor, prin declararea **CONST**, astfel:

CONST id_var : tip_var = valoare

unde

- **id_var** - este identificatorul variabilei
- **tip_var** - este tipul variabilei
- **valoare** - este valoarea inițială a variabilei.

Exemplu: **CONST** a: Integer = 1.

Variabilele inițializate pot fi utilizate similar variabilelor propriu-zise de acelaşi tip şi pot să apară şi în membrul stâng al unei instrucțiuni de atribuire. Deoarece aceste variabile sunt inițializate numai o singură dată – la începutul programului, rezultă că ele nu vor fi reinițializate la reapelarea procedurilor sau funcțiilor.

5.2 INSTRUCȚIUNEA DE ATRIBUIRE

Este instrucțiunea cea mai frecventă a limbajului şi are sintaxa exprimabilă prin una din următoarele forme:

variabilă1 := variabilă2

variabilă := expresie

nume de funcție := expresie

Execuția instrucțiunii decurge astfel : valoarea **variabilei2** sau valoarea calculată a expresiei (din partea dreaptă a semnului ":=") se atribuie variabilei sau numelui de funcție (din partea stângă a semnului ":="). În urma acestei atribuiri, variabila din stânga (care primeşte valoarea) îşi pierde valoarea anterioară, prin înlocuire cu noua valoarea rezultată din atribuire. Expresia din membrul drept trebuie să fie compatibilă din punct de vedere al atribuirii cu tipul variabilei sau funcției din membrul stâng.

Se accentuează sensul de atribuire a instrucțiunii (în sensul transmiterii unei valori, & 2.3.1 litera "e") şi nu de identitate matematică, motiv pentru care se utilizează simbolul ":=" în locul simbolului "=", iar caracterul spațiu nu este admis între simbolul "**:**" respectiv "**=**".

În momentul execuției instrucțiunii de atribuire toate variabilele care intervin în expresie trebuie să aibe valori.

Expresiile din dreapta semnului ":=" trebuie să respecte regulile precizate la & 4.3.

5.3 INSTRUCȚIUNEA COMPUSĂ

Instrucțiunea compusă este o secvență de instrucțiuni, despărțite prin separatorul "**;**" și încadrate între cuvintele cheie **BEGIN** și **END**.

Semnificația acestei instrucțiuni este execuția instrucțiunilor componente în ordinea scrierii lor. Instrucțiunile componente sunt considerate ca o singură instrucțiune.

Sintaxa instrucțiunii compuse este:

```
BEGIN
        instrucțiune ;
        ...……..   ;
        instrucțiune ;
END
```

Cuvintele cheie **BEGIN** și **END** joacă rolul unor *'paranteze'*, deoarece, din punct de vedere al limbajului, mulțimea instrucțiunilor cuprinse între acestea formează o singură instrucțiune. Instrucțiunea compusă se va folosi atunci când sunt necesare mai multe instrucțiuni în locul unde sintaxa impune utilizarea unei singure instrucțiuni;

Exemple:

- ramura **THEN** și **ELSE** ale instrucțiunii **IF**
- ciclurile **WHILE**, **FOR**.

Observații:

- O instrucțiune compusă poate fi vidă, caz în care se reduce la instrucțiunile:
  BEGIN
  END
- Corpul programului principal sau al unui subprogram constă într-o instrucțiune compusă (& 4.4).
- Instrucțiunile interioare instrucțiunii compuse se execută în ordinea apariției lor.
- Instrucțiunea compusă se consideră sintactic o singură instrucțiune.
- O instrucțiune compusă poate fi inclusă ca instrucțiune în altă instrucțiune compusă.

5.4 INSTRUCȚIUNEA VIDĂ

Este o instrucțiune care nu conține nimic. Dacă doi separatori de tip "**;**" sunt scriși succesiv, deci de forma "**;;**" atunci între cei doi se creează o instrucțiune vidă. Există posibilități de utilizarea a acestei instrucțiuni (exemplu la instrucțiunea **IF**).

5.5 INSTRUCȚIUNI DE INTRARE – IEȘIRE

Finalitatea oricărui program este calcularea unuia sau mai multor seturi de date de ieşire corespunzătoare unor date de intrare. Interacțiunea dintre utilizator şi program se realizează prin intermediul unor instrucțiuni specializate. Una din cele mai frecvente posibilități este introducerea datelor de intrare de la tastatură respectiv afişarea datelor de ieşire pe display, dar există şi alte posibilități de intrare-ieşire.

Pentru operațiilor de intrare-ieşire se utilizează procedurile de citire a datelor **READ** şi **READLN**, respectiv de scriere a datelor **WRITE** şi **WRITELN**.

5.5.1 Instrucțiuni de citire a datelor

Sintaxa instrucțiunii de citire a datelor este:

READLN (V_1, V_2, ..., V_n) sau **READLN**

care are efectul citirii de la tastatură a valorii variabilelor V_1, V_2, ..., V_n, în sensul alocării valorilor introduse de la tastatură variabilelor din listă.

Practic, instrucțiunea aşteaptă introducerea, de către utilizator, a uneia sau mai multor valori (numărul lor fiind egal cu cel al variabilelor din listă), valori care sunt transmise variabilelor din lista instrucțiunii, în ordinea apariției lor. Deci "*citirea*" nu se referă de fapt la operația de introducere a datelor, care este efectuată de către utilizator prin intermediul tastaturii, ci se referă la preluarea acestor valori în locații de memorie definite prin variabilele din listă.

Valorile introduse de la tastatură trebuie să corespundă ca *tip, număr şi ordine* cu variabilele din listă. Variabilele din listă nu pot fi de tip boolean sau de tip enumerare (sau un subdomeniu al acestuia).

Introducerea mai multor valori se poate face prin separare prin **spațiu** sau **Enter**. La sfârşitul valorilor trebuie tastat **Enter**, cu efectul trecerii cursorului ecranului pe rândul următor. Procedura **READLN** va citi şi sfârşitul de linie (ultimul **Enter)**.

Exemplu:

Dacă în secțiunea declarativă a unui program (& 4.4) sunt definite următoarele variabile (& 5.1)

```
VAR   a, b : INTEGER ;
      c : REAL ;
```

în secțiunea instrucțiunilor programului instrucțiunea

```
READLN (a, b, c)
```

aşteaptă introducerea a trei valori numerice, dintre care primele 2 trebuie să fie de tip **INTEGER** (& 4.2.1), iar ultima de tip **REAL** (& 4.2.3). Presupunând că cele 3 valori sunt numerele: **120 45 9.78** introducerea lor poate fi operată în mai multe moduri:

- scrierea individuală a valorilor numerelor (de la tastatură), după fiecare număr se apasă tasta **Enter**, astfel:

120 Enter
45 Enter
9.78 Enter

- scrierea simultană a celor 3 numere, pe acelaşi rând, dar despărțite prin **spațiu** şi finalizarea introducerii prin tasta **Enter**, astfel:

120 45 9.78 Enter

- combinarea celor două metode, astfel:

120 45 Enter
9.78 Enter

Oricare ar fi modalitatea de introducere a valorilor, prima dintre ele (**120**) va fi memorată în variabila "**a**", a doua dintre ele (**45**) va fi memorată în variabila "**b**", iar ultima (**9.78**) va fi memorată în variabila "c", deci valorile vor fi memorate în variabilele din lista **READLN**, în ordinea în care apar în listă.

Procedura **READLN** se poate folosi, fără nici un argument, la finalul programelor, înainte de instrucțiunea "**END.**", efectul fiind aşteptarea apăsării tastei **Enter** cu menținerea afişării ecranului cu rezultate şi revenirea în mediul TurboPascal, evitând astfel necesitatea reafişării ecranului de rezultate prin combinația de taste **Alt+F5**.

Există şi următoarea formă a instrucțiunii:

READ (V_1, V_2, ..., V_n)

care are acelaşi efect ca şi **READLN**, cu diferența că nu citeşte sfârşitul de linie (ultimul **Enter)**. La execuția acestei instrucțiuni cursorul ecran rămâne poziționat pe linia de introducere a valorilor, la sfârşitul acesteia, astfel încât, o eventuală nouă instrucțiune de citire va începe citirea datelor de pe linia în curs, de la poziția rămasă anterior.

În cazul şirurilor de caractere se recomandă utilizarea formei **READLN**.

Datele sunt memorate într-o zonă tampon înainte de a fi alocate variabilelor. Această zonă tampon se goleşte când apare **READLN.** Apelul simplu **READLN** fără argumente are deci ca efect golirea zonei tampon. Deci secvența **READLN**(x,z,...) este echivalentă cu secvența **READ** (x,z,...); **READLN**.

Eventualele valori ale variabilelor existente înainte de apelul procedurii **READ** sau **READLN** se pierd prin execuția acesteia. Valorile citite ale variabilelor se vor conserva până la modificarea lor printr-o nouă citire sau printr-o instrucțiune de atribuire.

5.5.2 Instrucţiuni de scriere a datelor

Rolul acestor instrucțiuni este de afişare a rezultatelor (datelor de ieşire). Prin aceste instrucțiuni pot fi afişate pe display următoarele entități:

- *variabile* – scrierea unei variabile constă în afişarea valorii ei curente şi nu a numelui acesteia;
- *expresii* – scrierea unei expresii constă în afişarea valorii expresiei, valoare rezultată în urma efectuării tuturor calculelor expresiei;
- *mesaje* – scrierea unui mesaj constă în afişarea succesiunii de caractere dintre apostroafe.

Oricare din aceste entități pot fi afişate individual sau oricare combinație a lor, prin includerea între paranteze a listei de entități de afişat (variabile, mesaje sau expresii) *separate prin virgulă*. Pot fi afişate numai date de tip întreg, real, char, şir de caractere sau boolean (caz în care se afişează cuvintele **True** sau **False**), dar nu pot fi de tip enumerare. După afişare, valoarea variabilelor nu se pierde şi nu se modifică.

Sintaxa instrucțiunii de scriere a datelor este:

WRITELN (lista de entități) sau **WRITE** (lista de entități)

Singura diferență dintre cele două forme ale instrucțiunii este că, la sfârşitul afişării listei de entități, cursorul ecran este mutat pe rândul următor (pentru forma **WRITELN**) respectiv cursorul rămâne poziționat pe rândul de afişare, astfel încât o nouă instrucțiune va provoca afişarea în continuarea aceluiaşi rând (pentru forma **WRITE**).

Exemplu 5.1 Dacă în secțiunea declarativă a unui program (& 4.4) sunt definite următoarele variabile (& 5.1):

```
VAR   c : INTEGER ;
      a, b : REAL ;
```

în secțiunea instrucțiunilor programului, următoarea instrucțiune afişează o listă de entități, listă compusă din 3 mesaje, 1 expresie şi 1 variabilă.

mesaj *mesaj* *mesaj*

WRITELN (‘Solutia =’, a * b, ‘ obținută în ‘, c , ‘ pasi’)

expresie *variabilă*

Pentru următoarele valori ale variabilelor a=1.5, b=2.8, c=11 rezultă afişarea:

mesaj *mesaj* *mesaj*

```
Solutia = 4.2000000000E+00 obtinuta în 11 pasi
```

valoare expresie *valoare variabilă*

din care rezultă afişarea:

- pentru mesaje - a şirurilor de caractere dintre apostroafe, dar nu şi apostroafele;
- pentru expresii – valoarea calculată a acestora, **a * b = 4.2** ;
- pentru variabile - valoarea acestora.

Exemplu 5.2

Combinația de instrucțiuni:

WRITE (‘Solutia =’, a * b, ‘ obtinută în ‘);
WRITE (c , ‘ pasi’);

are acelaşi efect, din punct de vedere al afişării, ca şi unica instrucțiune de afişare din exemplul 5.1, deoarece instrucțiunea **WRITE** conservă poziția cursorului pe rândul curent scris, deci după afişarea entităților argument al primei instrucțiuni **WRITE**, cursorul ecran rămâne poziționat pe aceeaşi linie ecran, astfel că ce-a de-a doua instrucțiune **WRITE**, continuă afişarea începând de la ultima poziție scrisă de anterioara instrucțiune **WRITE**.

Exemplu 5.3

Este un obicei frecvent utilizat asocierea unei instrucțiuni de introducere a datelor (**READLN**) cu o instrucțiune prealabilă de afişare a unui mesaj (**WRITELN**) al cărui rol este informare asupra tipului / funcției / caracteristicii datei ce urmează a fi introdusă.

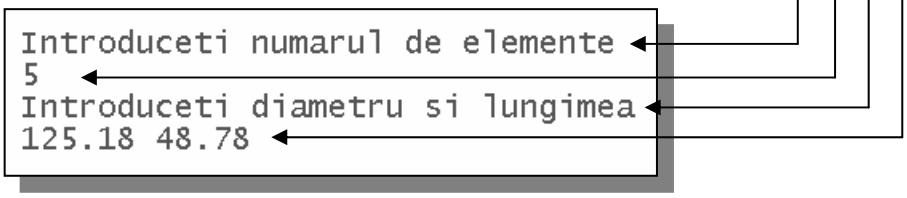
Dacă în secțiunea declarativă a unui program (& 4.4) sunt definite următoarele variabile (& 5.1)

```
VAR   n : INTEGER ;
      d, lungime : REAL ;
```

în secțiunea instrucțiunilor programului, instrucțiunile de introducere a datelor (**READLN**) precedate de instrucțiuni de afişare de mesaje informative asociate acestora (**WRITELN**) ar putea fi:

```
WRITELN(‘Introduceti numarul de elemente’);
READLN(n);
WRITELN(‘Introduceti diametru si lungimea’);
READLN(d, lungime);
```

iar la execuția programului, ecranul de afişare / introducere date se prezintă astfel:



Trebuie menționat faptul că citirea propriu-zisă a datelor nu se produce ca o consecință a mesajelor generate prin instrucțiunile **WRITELN** (care au numai rol de afişare a unor mesaje) ci *numai* datorită instrucțiunilor **READLN**, care au rol de citire a datelor (& 5.5.1). Introducerea datelor este deci independentă de afişarea de mesaje asociate, atât din punct de vedere al programării instrucțiunilor cât şi din punct de vedere al operării, totuşi precedarea instrucțiunilor de citire cu mesaje asociate ajută utilizatorul la introducerea datelor, în sensul luării la cunoştință a tipului datelor şi a ordinii în care acestea trebuie introduse.

După cum se observă din exemplul 5.1, modul implicit de afişare a datelor de tip real este formatul exponențial (& 4.1). Se poate impune însă afişarea datelor reale / întregi într-un *format* impus. Astfel, în lista argumentelor procedurilor **WRITE** şi **WRITELN**, variabilele şi/sau expresiile pot fi urmate de *formatul de afişare*, definit astfel:

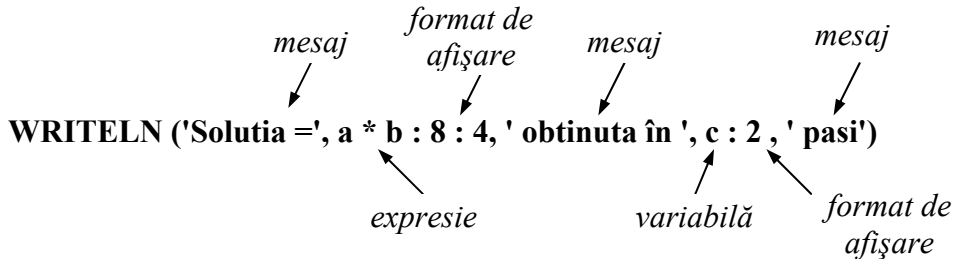
ve : m : n

unde :

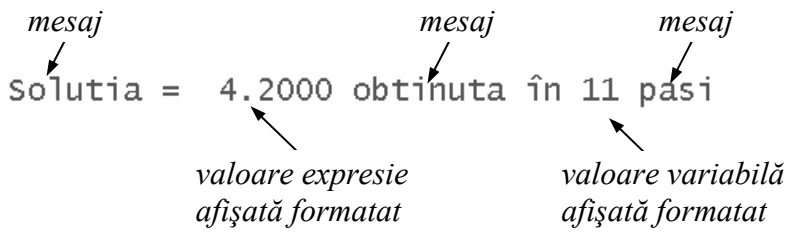
- **ve** este variabila sau expresia de afişat:
- **m** şi **n** sunt expresii cu valori întregi, cu semnificațiile:

- **m** – numărul de poziții pe care se va afişa expresia; dacă valoarea este mai mare decât numărul de poziții rezervat, atunci **m** este ignorat, iar dacă este mai mic, afişarea se va completa cu spații până la completarea numărului **m** impus;
- **n** – numărul de zecimale cu care se doreşte scrierea valorii; dacă se afişează un număr întreg sau un şir de caractere **n** trebuie să lipsească; din cele **m** poziții, una este ocupată de punctul zecimal, dacă **n** este specificat.

Exemplu 5.4 Dacă pentru datele specificate la exemplul 1 se utilizează instrucțiunea următoarea instrucțiune de afişare:



rezultatele afişării vor fi următoarele:



Se remarcă afişarea în format zecimal a valorii reale a rezultatului calculat al expresiei **a * b**, pe un total de 8 poziții, din care 4 sunt zecimale; deoarece 1 poziție este ocupată de punctul zecimal, rămâne disponibil pentru partea întreagă un număr de 3 poziții; deoarece partea întreagă conține numai o cifră, 2 poziții vor completate cu spații; de asemenea, pentru afişarea valorii întregi a variabilei "**c**" s-au impus prin formatul de afişare 2 poziții.

Alte exemple de afişare formatată sunt date în tabelul 5.1, unde prin semnul "-" se înțelege caracterul **spațiu**, de unde rezultă:

- autocompletarea afişărilor cu spații până la atingerea numărului maxim de poziții impus prin format (liniile 1,2,3);
- rotunjirea părții zecimale, la impunerea afişării unui număr mai mic de zecimale în raport cu cel real (linia 2);
- autocompletarea cu zero-uri a părții zecimale până la atingerea numărului de zecimale impus de format (linia 4).

Tabel 5.1

Nr. Crt.	Variabilă	Tip de date	Valoare	Format de afişare impus	Afişare formatată
1	a	INTEGER	123	a : 5	- -123
2	b	REAL	145.78	b : 8 : 1	- - -145.8
3	b	REAL	145.78	b : 8 : 2	- - 145.78
4	b	REAL	145.78	b : 8 : 4	145.7800

5.6 APLICAȚII

5.6.1 Media aritmetică a trei numere

E 5.5 Program TurboPascal pentru calcul mediei aritmetice a trei numere

Prezentarea teoretică şi rezolvarea algoritmului problemei au fost analizate în & 2.5.1, exemplul 2.12 şi exprimat prin limbaj pseudocod, fig. 2.31 respectiv schemă logică, fig. 2.32.

Prima linie a programului, fig. 5.1, defineşte numele său (& 4.4), iar a doua reprezintă un comentariu, care exprimă conținutul său (& 4.1). În a treia linie sunt declarate (& 5.1) variabilele programului ca fiind de tip real (& 4.2.3), respectiv datele de intrare **a, b, c**, precum şi data de ieşire **media**.

```
PROGRAM medie3;
{Calcul medie aritmetica a 3 numere}
VAR a, b, c, media: REAL;
BEGIN
   WRITELN('Media aritmetica a 3 numere');
   WRITELN('a=');
   READLN(a);
   WRITELN('b=');
   READLN(b);
   WRITELN('c=');
   READLN(c);
   MEDIA := (a+b+c)/3;
   WRITELN('Media=',media:10:5);
   READLN
END.
```

Fig. 5.1

```
Media aritmetica a 3 numere
a=
123.45
b=
184.1
c=
99.784
Media=  135.77800
```

Fig. 5.2

Secțiunea programului, inclusă între **BEGIN** şi **END.**, conține o instrucțiune de afişare pe display a conținutului programului, 3 instrucțiuni de citire corespunzătoare datelor de intrare (& 5.5.1), fiecare fiind precedată de câte o instrucțiune de scriere (& 5.5.2) asociată, cu rol de informare asupra valorilor care trebuie introduse; prin instrucțiunea de atribuire (& 5.2) se calculează valoarea mediei aritmetice, depozitată în variabila **media**, valoare afişată prin următoarea instrucțiune. Instrucțiunea finală **READLN** are rol de aşteptare a apăsării tastei **Enter**, pentru a menține afişarea ecranului utilizator cu rezultatele programului, fig. 5.2.

5.6.2 Calcul arie triunghi prin formula lui Heron

E 5.6 Program TurboPascal pentru calcul arie triunghi prin formula Heron

Prezentarea teoretică şi rezolvarea algoritmului problemei au fost analizate în & 2.4.1, exemplul 2.7 şi exprimat prin limbaj pseudocod, fig. 2.11 respectiv schemă logică, fig. 2.12.

```
PROGRAM aria_heron;
VAR a, b, c, p, s : REAL;
BEGIN
   WRITELN('Calcul arie prin formula Heron');
   WRITELN('Introduceti cele trei laturi:');
   READLN(a, b, c);
   p := (a+b+c)/2;
   s := SQRT(p*(p-a)*(p-b)*(p-c));
   WRITELN('Aria=', s : 10: 4);
   READLN
END.
```

Fig. 5.3

```
Calcul arie prin formula Heron
Introduceti cele trei laturi:
3 4 5
Aria=     6.0000
```

Fig. 5.4

Prima linie a programului, fig. 5.3, defineşte numele său (& 4.4), iar în a doua linie sunt declarate (& 5.1) variabilele programului ca fiind de tip real (& 4.2.3), respectiv datele de intrare: laturile triunghiului - **a, b, c**, data intermediară de calcul: semiperimetrul - **p**, precum şi data de ieşire: aria - **s**.

Secțiunea programului, inclusă între **BEGIN** şi **END.,** conține două instrucțiuni de afişare pe display cu rol informativ (conținutului programului respectiv atenționarea introducerii a trei valori de laturi), o instrucțiune de citire corespunzătoare datelor de intrare (& 5.5.1); prin instrucțiuni de atribuire (& 5.2) se calculează valoarea semiperimetrului respectiv ariei, depozitate în variabilele **p** respectiv **s**, valoarea calculată a ariei este afişată prin următoarea instrucțiune. Instrucțiunea finală **READLN** are rol de aşteptare a apăsării tastei **Enter**, pentru a menține afişarea ecranului utilizator cu rezultatele programului, fig. 5.4.

Spre deosebire de exemplul anterior - exemplul 5.5, se remarcă faptul că citirea variabilelor se realizează într-o singură instrucțiune **READLN** şi nu prin trei instrucțiuni separate, ceea ce este mai eficient din punct de vedere al programării.

Variabila **p** este o variabilă intermediară de calcul, a cărei valoare nu este interesantă din punct de vedere al cerințelor problemei, motiv pentru care nu se afişează.

Funcția **SQRT** este funcția radical (& 4.2.3), iar argumentul acesteia, adică expresia din care trebuie extras radicalul, trebuie încadrat între paranteze rotunde.

Valoarea ariei este afişată formatat (& 5.5.2), pe 10 poziții maximale, dintre care 4 sunt ocupate de zecimale, una de punctul zecimal, deci mai rămân 5 poziții pentru partea întreagă; pentru valoarea întreagă a ariei **S** = 6 completarea acestor 5 poziții impune autocompletarea cu 4 spații, după cum rezultă din fig. 5.4.

5.6.3 Inversarea valorii a două variabile

E 5.7 Program TurboPascal pentru inversarea valorii a două variabile

Prezentarea teoretică şi rezolvarea algoritmului problemei au fost analizate în & 2.5.2, exemplul 2.13 şi exprimat prin limbaj pseudocod, fig. 2.33 respectiv schemă logică, fig. 2.34.

```
PROGRAM invers;
{Inversarea valorii a doua variabile}
VAR a,b,c: REAL;
BEGIN
   WRITELN('Introduceti doua valori');
   READLN(a, b);
   c := a;
   a := b;
   b := c;
   WRITELN('Valorile inversate ', a: 9: 4, b: 9: 4);
   READLN
END.
```

Fig. 5.5

```
Introduceti doua valori
123.45
28
Valorile inversate   28.0000 123.4500
```

Fig. 5.6

Prima linie a programului, fig. 5.5, defineşte numele său (& 4.4), a doua instrucțiune este un comentariu, care exprimă conținutul său (& 4.1), iar în a treia linie sunt declarate (& 5.1) variabilele programului ca fiind de tip real (& 4.2.3), respectiv datele de intrare: variabilele - **a, b**, data intermediară de calcul: variabila - **c**; datele de ieşire sunt aceleaşi variabile **a** şi **b**, dar cu valori inversate.

Secțiunea programului, inclusă între **BEGIN** şi **END.**, conține o instrucțiune de afişare pe display cu rol informativ (atenționarea introducerii a două valori), o instrucțiune de citire corespunzătoare datelor de intrare (& 5.5.1); prin trei instrucțiuni de atribuire (& 5.2) se execută inversarea valorilor celor două variabile **a** şi **b**, folosind variabila intermediară **c**; valoarea celor două variabile este afişată prin următoarea instrucțiune. Instrucțiunea finală **READLN** are rol de aşteptare a apăsării tastei **Enter**, pentru a menține afişarea ecranului utilizator cu rezultatele programului, fig. 5.6.

Reamintim logica instrucțiunii de atribuire (& 5.2): valoarea variabilei din dreapta semnului egal este atribuită variabilei din stânga semnului egal, deci sensul transmiterii valorii este întotdeauna de la dreapta la stânga şi nu invers. Din acest motiv este foarte important care variabilă este în stânga (cea care primeşte valoarea) şi care este în dreapta (cea care transmite valoarea). Tabelul 5.2 reflectă transferul valorilor între variabile.

Tabel 5.2

Exemplificare numerică execuție algoritm			
Instrucțiune	**a**	**b**	**c**
Valori inițiale	123.45	28	
c := a			123.45
a := b	28		
b := c		123.45	
Valori finale: **a=28 ; b**=123.45			

5.6.4 Calcul perimetru şi arie cerc

E 5.8 Program TurboPascal pentru calcul perimetru şi arie cerc

Presupunând cunoscută raza unui cerc **R**, perimetrul **P** respectiv aria **A** se calculează prin relațiile consacrate:

$$P = 2 * \pi * R$$

$$A = \pi * R^2$$

Fig. 5.7

```
PROGRAM calcul_arie;                                  {Titlu program}
VAR raza, perim, aria: REAL;                          {Declarare variabile tip real}
BEGIN                                                 {Marcare inceput bloc de comenzi}
  WRITE('Raza cerc =');                               {Afisare mesaj Raza cerc= }
  READLN(raza);                                       {Citire valoare raza}
  perim := 2*PI*raza ;                                {Calcul perimetru cerc}
  aria := PI*SQR(raza);                               {Calcul arie cerc}
  WRITELN ('Perimetru cerc = ', perim : 10:3);        {Afisare valoare perimetru}
  WRITELN ('Suprafata cerc = ', aria : 10 : 3);       {Afisare valoare arie}
  READLN                                              {Aşteptarea apăsare tasta ENTER}
END.                                                  {Marcare sfarsit program}
```

Fig. 5.8

```
Raza cerc =12.5
Perimetru cerc =     78.540
Suprafata cerc =    490.874
```

Prima linie a programului, fig. 5.7, defineşte numele său (& 4.4), iar în a doua linie sunt declarate (& 5.1) variabilele programului ca fiind de tip real (& 4.2.3), respectiv data de intrare: variabila - **raza** şi datele de ieşire: variabilele **perim** şi **aria**.

Secțiunea programului, inclusă între **BEGIN** şi **END.,** conține o instrucțiune de afişare pe display cu rol informativ (atenționarea introducerii valorii razei); deoarece afişarea se realizează prin instrucțiunea **WRITE** (& 5.5.2), după afişarea mesajului cursorul rămâne poziționat pe acest rând, astfel încât instrucțiunea de citire **READLN** a valorii variabilei **raza** (& 5.5.1) execută citirea de pe acelaşi rând, de la ultima poziție a cursorului ; prin două instrucțiuni de atribuire (& 5.2) se execută calculul celor două variabile **perim** şi **aria**; în expresia de calcul a ariei intervine funcția **SQR** care returnează pătratul argumentului; valoarea celor două variabile este afişată formatat prin următoarele două instrucțiuni **WRITELN**. Instrucțiunea finală **READLN** are rol de aşteptare a apăsării tastei **Enter**, pentru a menține afişarea ecranului utilizator cu rezultatele programului, fig. 5.8. Se observă descrierea fiecărei instrucțiuni printr-un comentariu asociat (& 4.1).

Variabila **PI** nu este declarată în secțiunea declarațiilor programului, deoarece este o constantă predefinită şi utilizabilă în programe sub identificatorul **PI** şi valoarea **3.14159265358979323 85**, fără a fi nevoie de declararea explicită a acestuia.

6. INSTRUCȚIUNI CONDIȚIONALE

O **instrucțiune condițională (IF** sau **CASE)** realizează selectarea, în vederea execuției, a unei singure instrucțiuni din două sau mai multe posibile.

6.1 INSTRUCȚIUNEA DE DECIZIE

Instrucțiunea condițională **IF** selectează, pentru execuție, o singură instrucțiune din două alternative posibile. Sintaxa instrucțiunii este concretizată prin oricare din formele de mai jos:

IF *expresie logică* **THEN**
instrucțiune1
ELSE
instructiune2

IF *expresie logică* **THEN** *instrucțiune1*
ELSE *instrucțiune2*

IF *expresie logică* **THEN** *instrucțiune1* **ELSE** *Instrucțiune2*

Expresie logică reprezintă condiția ce trebuie testată şi al cărei rezultat trebuie să fie de tip logic (& 4.2.2). Modul de execuție al acestei instrucțiuni este următorul:

1. Se evaluează condiția plasată după cuvântul cheie **IF**.
2. Dacă valoarea condiției este **TRUE** (adevărată) atunci se execută instrucțiunea care apare după cuvântul cheie **THEN** (*instrucțiune1*) şi apoi se execută pasul 4.
3. Dacă valoarea condiției este **FALSE** (fals) atunci se execută instrucțiunea care apare după cuvântul cheie **ELSE** (*instrucțiune2*) şi apoi se execută pasul 4.
4. Se continuă execuția următoarei instrucțiuni plasată după instrucțiunea **IF**.

Este corectă şi oricare din următoarele forme:

IF *expresie logică* **THEN**
instrucțiune1

IF *expresie logică* **THEN** *instrucțiune1*

în care ramura **ELSE** lipseşte, care se execută astfel:

1. Se evaluează condiția plasată după cuvântul cheie **IF**.
2. Dacă valoarea condiției este **TRUE** (adevărată) atunci se execută instrucțiunea care apare după cuvântul cheie **THEN** (*instrucțiune1*).
3. Se continuă execuția următoarei instrucțiuni plasată după instrucțiunea **IF**.

În ansamblul ei, instrucțiunea de decizie este considerată o singură instrucțiune, chiar dacă ea include în sintaxă una sau două instrucțiuni executabile (*instrucțiune1* şi/sau *instrucțiune2*), deoarece aceste instrucțiuni sunt incluse în instrucțiunea IF, nu sunt separate de aceasta.

Instrucțiunea de decizie reprezintă concretizarea în TurboPascal a instrucțiunii de decizie analizată la descrierea algoritmilor prin reprezentare în schemă logică (& 2.3.1, pct. f) respectiv limbaj pseudocod (& 2.3.2, pct. d).

Deoarece atât pe ramura **THEN** cât şi pe ramura **ELSE** poate să apară orice instrucțiune, inclusiv un alt **IF-THEN-ELSE**, ambiguitatea unei expresii de forma:

IF *condiție1* **THEN**
IF *condiție2* **THEN** *instrucțiune1*
ELSE *instrucțiune2*

în sensul deciderii cărui **IF** îi aparține ramura "**ELSE** *instrucțiune2*" (primului **IF** sau celui de-al doilea interior), se rezolvă după următoarea regulă a sintaxei limbajului: *construcția **ELSE** se asociază cu cel mai apropiat cuplu **IF-THEN** anterior care nu a fost împerecheat cu un **ELSE***.

Deci în construcția anterioară ramura **ELSE** se va împerechea cu cel de-al doilea **IF**, interpretându-se astfel:

IF condiție1 **THEN**
IF condiție2 **THEN** *instrucțiune1*
ELSE *instrucțiune2*

În toate cazurile în care pot apare ambiguități se recomandă utilizarea cuvintelor cheie **BEGIN** – **END**, care încadrează construcția **IF-THEN-ELSE** corect împerecheată, formând astfel o instrucțiune compusă (& 5.3).

IF *condiție1* **THEN**
BEGIN
IC
IF *condiție2* **THEN** *instrucțiune1*
ELSE *instrucțiune2*
END

Instrucțiunea compusă se foloseşte în mod obligatoriu, pe oricare din cele două ramuri, atunci când algoritmul necesită execuția mai multor instrucțiuni, intrând în contradicție cu sintaxa instrucțiunii **IF**, care impune execuția unei singure instrucțiuni. În aceste situații, grupul de instrucțiuni se încadrează între cuvintele cheie **BEGIN** – **END**, formând o instrucțiune compusă (& 5.3), care este consideră de către compilator din punct de vedere sintactic ca şi cum ar fi o singură instrucțiune.

OBSERVAȚII

1. *Instrucțiune1* sau *Instrucțiune2* poate fi de atribuire (& 5.2), apel de procedură **READ**, **READLN** (& 5.5.1), **WRITE**, **WRITELN** (& 5.5.2), altă instrucțiune **IF**, instrucțiune compusă (& 5.3), instrucțiune repetitivă sau altă instrucțiunea validă TurboPascal.

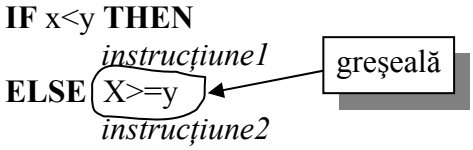
2. Este o greşeală plasarea separatorului "**;**" după **THEN**, ceea ce este interpretat ca fiind o instrucțiune vidă (& 5.4) pentru **IF**, urmat de o altă instrucțiune de sine stătătoare. Deci, în acest caz, *instrucțiune* nu va mai fi asociată ramurii **THEN**, ci va fi considerată ca o instrucțiune separată, ceea ce va provoca execuția ei indiferent de valoarea adevărată sau nu a condiției, modificând astfel logica inițială a algoritmului.

greşeală
IF condiție **THEN** ;
instrucțiune

3. Cuvântul cheie **ELSE** nu este precedat de separatorul punct şi virgulă, deoarece acesta separă şi nu încheie instrucțiunile, iar cuvântul cheie **ELSE** este parte componentă a instrucțiunii **IF-THE-ELSE**.

4. Condiția **a = 0** nu se pune sub forma **a := 0**, deoarece reprezintă o comparație şi nu o atribuire (& 5.2); deci se compara valoarea lui **a** cu **0**, nu se atribuie lui **a** valoarea **0**. În urma unei comparații, mărimile comparate îşi conservă valorile, pe când în urma unei atribuiri variabila din stânga simbolului de atribuire (**:=**) îşi modifică valoarea. Deci în expresiile condițiilor se vor folosi operatorii relaționali (& 4.2.2, tabel 4.5).

5. Este inutilă şi incorectă sintactic plasarea unei condiții suplimentare de forma:

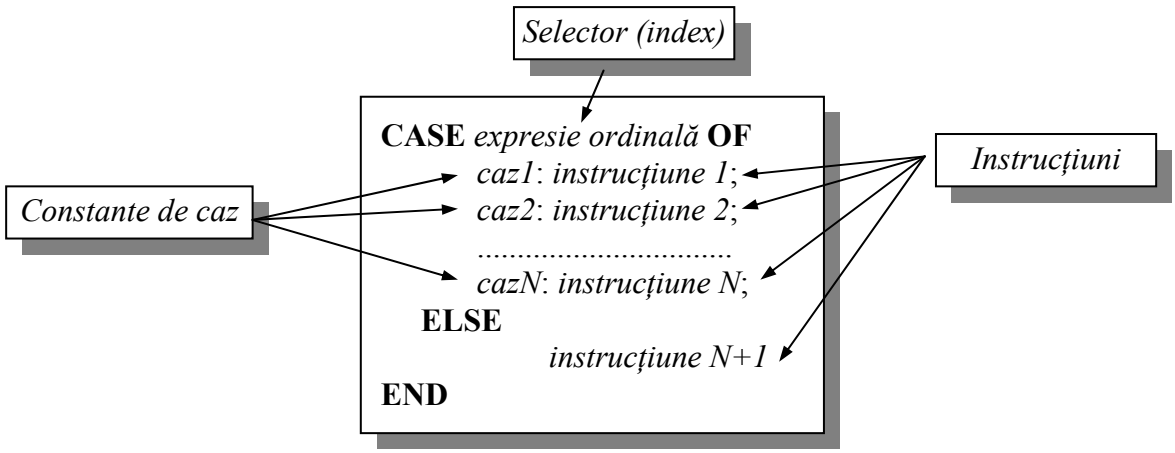


deoarece condiția asociată ramurii **ELSE** este contrariul condiției asociate ramurii **IF**, iar sintaxa instrucțiunii "*subânțelege*" condiția contrară prin însăşi definiția ei.

6.2 SELECȚIA MULTIPLĂ. INSTRUCȚIUNEA "CASE"

Instrucțiunea de selecție multiplă **CASE** permite selecția, pentru execuție, a unei singure alternative din mai multe posibile. Ea conține o expresie ordinală numită *selector* (*index*) şi o listă de instrucțiuni, fiecare fiind precedată de una sau mai multe constante, numite *constante de caz* sau de cuvântul cheie **ELSE**.

Sintaxa instrucțiunii **CASE** este:



Modul de execuție al instrucțiunii este următoarea: după evaluarea expresiei se execută o singură instrucțiune şi anume cea prefixată de constanta sau domeniul de constante care conține valoarea expresiei, după care execuția continuă cu prima instrucțiune de după **END**; dacă valoarea expresiei nu se încadrează în nici una din valorile constantelor de caz se execută instrucțiunea de pe ramura **ELSE** (dacă aceasta există), iar în caz contrar se execută prima instrucțiune după **END**-ul instrucțiunii **CASE**.

RESTRICȚII:

- *selectorul* trebuie să fie o expresie de tip ordinal (& 4.2), adică să aparțină unei mulțimi ordonate şi finite de valori; tipurile ordinale sunt: tipul logic, caracter, întreg, interval şi enumerare;
- toate *constantele de caz* trebuie să fie unice şi compatibile din punct de vedere al atribuirii cu tipul *selectorului*.

OBSERVAȚII

- înaintea unei instrucțiuni pot fi plasate mai multe constante de caz, despărțite prin separatorul ","; variantă care se utilizează atunci când se doreşte execuția aceleiaşi instrucțiuni pentru mai multe valori posibile ale *selectorului*;

Val1, Val2, Val3 : **instrucțiune**

- înaintea unei instrucțiuni poate fi specificat un domeniu de valori consecutive, prin următoarea formulare sintactică:

ValMin .. ValMax : **instrucțiune**

unde **ValMin** reprezintă valoarea minimă a domeniului, **ValMax** reprezintă valoarea maximă a domeniului, variantă care se utilizează atunci când se doreşte execuția aceleiaşi instrucțiuni pentru mai multe valori posibile ale *selectorului*, valori care aparțin unui domeniu de valori de tip ordinal (& 4.2);

- este de asemenea admisă şi combinarea celor două metode, sub forma:

ValMin .. ValMax, Val1, Val2 : **instrucțiune**

variantă care se utilizează atunci când se doreşte execuția aceleiaşi instrucțiuni pentru mai multe valori posibile ale *selectorului*, valori care aparțin unui domeniu de valori de tip ordinal sau pot fi specificate individual;

- ramura **ELSE** este opțională, iar separatorul ";" plasat înainte de **ELSE** sau **END** nu este o greşeală pentru această instrucțiune;
- constantele de caz nu trebuie confundate cu etichetele, care pot fi definite prin declarația **LABEL** şi pot fi referite prin instrucțiunea **GOTO**; este deci interzisă referirea constantelor de caz prin instrucțiunile de salt **GOTO**;

EXEMPLU: Secvența de instrucțiuni din exemplele următoare reprezintă extrase dintr-un posibil program.

```
VAR i: INTEGER;
.....
CASE i OF
        0: a:=0
        1: a:=10
        2: a:=20;
ELSE
        a:=50;
END;
```

Presupunând cunoscută valoarea variabilei de tip întreg "**i**" (& 4.2.1), prin citire sau atribuire, pentru instrucțiunea **CASE** aceasta reprezintă *selectorul,* iar valorile 0, 1, 2 *constantele de caz*; funcție de valoarea lui "**i**" (0, 1 respectiv 2), variabila "**a**" poate primi prin atribuire una din valorile 0, 10 sau 20. Dacă valoarea lui "**i**" este diferită de 0, 1 sau 2, variabila "**a**" va primi pe ramura **ELSE** valoarea 50.

6.3 APLICAȚII

6.3.1 Calculul modulului unui număr real

E 6.1 Program TurboPascal pentru calculul modulului unui număr real

Prezentarea teoretică şi rezolvarea algoritmului problemei au fost analizate în & 2.3.1, exemplul 2.3 şi exprimat prin schemă logică, fig. 2.5, respectiv, exprimat prin limbaj pseudocod, în & 2.3.2, exemplul 2.5, fig. 2.8.

```
PROGRAM modul;
{Calcul modul numar real}
VAR x, y: REAL;
BEGIN
   WRITELN('Introduceti un numar');
   READLN(x);
   IF x >= 0 THEN
      y := x
   ELSE
      y := -x;
   WRITELN('Modulul numarului ',x: 7: 3,' este',y: 7: 3);
   READLN;
END.
```

Prima linie a programului, fig. 6.1, defineşte numele său (& 4.4), iar a doua reprezintă un comentariu, care exprimă conținutul său (& 4.1). În a treia linie sunt declarate (& 5.1) variabilele programului ca fiind de tip real (& 4.2.3), respectiv data de intrare **x**, precum şi data de ieşire **y**.

Fig. 6.1

```
Introduceti un numar
3.25
Modulul numarului   3.250 este  3.250

Introduceti un numar
-17.58
Modulul numarului  -17.580 este 17.580
```

Fig. 6.2

Secțiunea programului, inclusă între **BEGIN** şi **END.**, conține o instrucțiune de afişare pentru atenționarea introducerii unui număr, o instrucțiune de citire a datei de intrare (& 5.5.1); funcție de valoarea adevărată sau nu a condiției din instrucțiunea de decizie se atribuie variabilei **y** (& 5.2) valoarea **x** respectiv **-x**, valoare afişată prin următoarea instrucțiune. Instrucțiunea finală **READLN** are rol de aşteptare a apăsării tastei **Enter**, pentru a menține afişarea ecranului utilizator cu rezultatele programului, fig. 6.2, figură în care sunt prezentate două exemple ale execuției programului: prima variantă, care corespunde parcurgerii ramurii **THEN** a instrucțiunii **IF**, iar a doua corespunde parcurgerii ramurii **ELSE**.

6.3.2 Rezolvarea ecuației de gradul I

E 6.2 Program TurboPascal pentru rezolvarea ecuației de grad I

Prezentarea teoretică şi rezolvarea algoritmului problemei au fost analizate în & 2.3.1, exemplul 2.4 şi exprimat prin schemă logică, fig. 2.6, respectiv, exprimat prin limbaj pseudocod, în & 2.3.2, exemplul 2.6, fig. 2.9.

Fig. 6.3

```
PROGRAM ecuatie1;
{Rezolvarea ecuatiei de grad I}
VAR a,b,x: REAL;
BEGIN
   WRITELN('Introduceti coeficientii ec. gr. 1');
   READLN(a, b);
   IF a <> 0 THEN
      BEGIN
IC       x := -b/a;
         WRITELN('Solutia ecuatiei=', x: 8: 3)
      END
   ELSE
        IF b=0 THEN WRITELN('Identitate')
               ELSE WRITELN('Imposibil');
   READLN
END.
```

Prima linie a programului, fig. 6.1, defineşte numele său (& 4.4), iar a doua reprezintă un comentariu, care exprimă conținutul său (& 4.1). În a treia linie sunt declarate (& 5.1) variabilele programului ca fiind de tip real (& 4.2.3), respectiv datele de intrare **a**, **b** precum şi data de ieşire **x**.

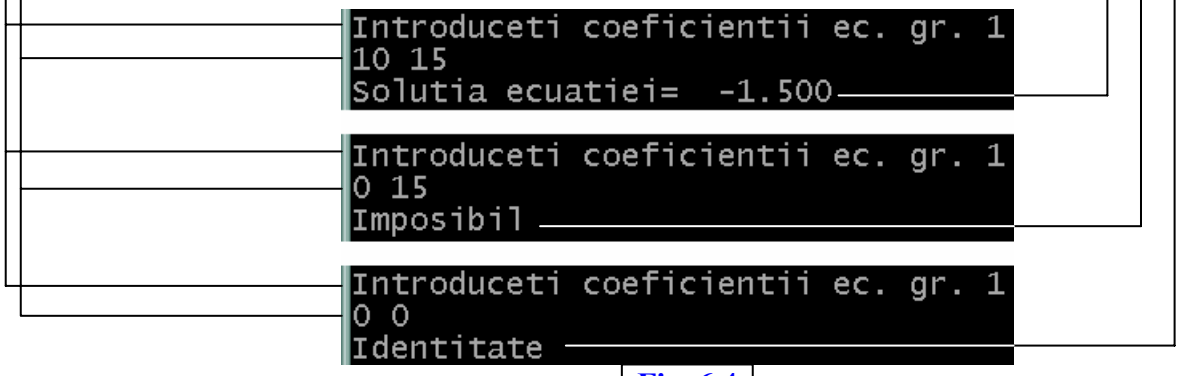


Fig. 6.4

Secțiunea programului, inclusă între **BEGIN** şi **END.**, conține o instrucțiune de citire a datelor de intrare (& 5.5.1), precedată de o instrucțiune de afişare cu rol de atenționare a introducerii datelor; funcție de valoarea adevărată sau nu a condiției **a <> 0** din instrucțiunea de decizie, pe ramura **THEN** se execută instrucțiunea compusă (formată din instrucțiunea de atribuire pentru calculul rădăcinii respectiv afişarea acesteia), iar pe ramura **ELSE** se execută

altă instrucțiune de decizie, care afișează un mesaj al cărui conținut este funcție de evaluarea condiției **b=0**. Instrucțiunea finală **READLN** are rol de așteptare a apăsării tastei **Enter**, pentru a menține afișarea ecranului utilizator cu rezultatele programului, fig. 6.4, figură în care sunt prezentate trei exemple ale execuției programului: prima variantă, care corespunde parcurgerii ramurii **THEN** a primei instrucțiunii **IF**, iar a doua și a treia corespunde parcurgerii ramurii **ELSE** respectiv **THEN** a celei de-a doua instrucțiuni **IF**.

6.3.3 Rezolvarea ecuației de gradul II

E 6.3 Program TurboPascal pentru rezolvarea ecuației de grad II

Prezentarea teoretică și rezolvarea algoritmului problemei au fost analizate în & 2.4.2, exemplul 2.8 și exprimat prin limbaj pseudocod în fig. 2.17, respectiv prin schemă logică, în fig. 2.18.

Prima linie a programului, fig. 6.5, definește numele său (& 4.4), iar a doua reprezintă un comentariu, care exprimă conținutul său (& 4.1). În a treia linie sunt declarate (& 5.1) variabilele programului ca fiind de tip real (& 4.2.3), respectiv datele de intrare **a**, **b, c**, datele de ieșire **x1, x2, x**, precum și data intermediară de calcul **delta**.

Secțiunea programului, inclusă între **BEGIN** și **END.**, conține o instrucțiune de citire a datelor de intrare (& 5.5.1), precedată de o instrucțiune de afișare cu rol de atenționare a introducerii datelor.

În continuare programul se ramifică în două alternative, funcție de condiția **a <> 0**:

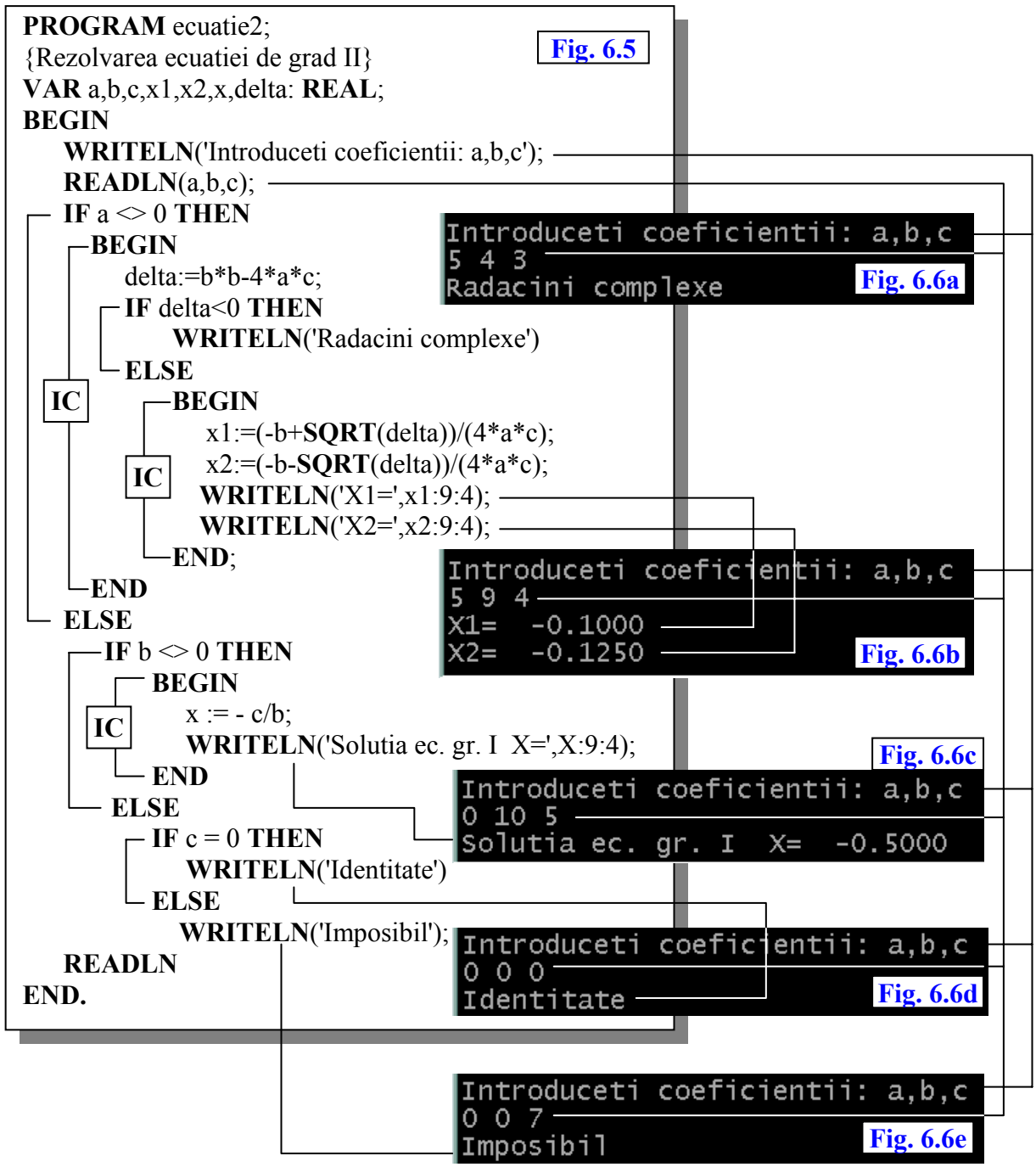
- dacă condiția **a <> 0** este adevărată, suntem în cazul ecuației de grad II, pentru a cărei rezolvare se impune calculul determinantului **delta**, funcție de valoarea căruia rezultă rădăcini complexe, pentru **delta < 0**, sau, în caz contrar, se pot calcula respectiv afișa cele două soluții;
- dacă condiția **a <> 0** este falsă, suntem în cazul ecuației de grad I, deci se parcurge aceeași logică cu cea prezentată în exemplul 6.2.

Instrucțiunea finală **READLN** are rol de așteptare a apăsării tastei **Enter**, pentru a menține afișarea ecranului utilizator cu rezultatele programului, fig. 6.6a ÷ 6.6b, figuri în care sunt prezentate cinci exemple ale execuției programului, care diferă funcție de datele introduse și care acoperă tot spectrul de variante posibile ale programului, mai puțin varianta calculării părților reale și respectiv imaginare ale soluției complexe.

În fig. 6.5 sunt marcate explicativ instrucțiunea compusă **IC** (& 5.3), iar ramurile instrucțiunii de decizie (& 6.1) cu linii asociate, cu observația că marcatorii grafici au un rol informativ și nu fac parte din programul propriu-zis. Se observă gruparea mai multor instrucțiuni asociate aceleiași ramuri a instrucțiunii de decizie, prin încadrarea între cuvintele cheie **BEGIN** respectiv **END** și formarea instrucțiunii compuse.

Se remarcă absența separatorului de instrucțiuni ";" în fața cuvântului cheie **ELSE** din instrucțiunile de decizie (& 6.1).

Cuvântul cheie **SQRT** reprezintă "*echivalentul informatic*" al funcției radical (& 4.2.3), iar argumentul acesteia trebuie încadrat între paranteze rotunde.



Desigur că, pentru un set de date de intrare, corespunde un set unic de date de ieşire, rezultat din parcurgerea alternativelor prevăzute în program. Sarcina programatorului este deci de a prevedea în algoritmul de rezolvare nu numai metoda de rezolvare propriu-zisă, ci şi tratarea eventualelor consecințe care pot rezulta dintr-o set particular de date de intrare, care poate influența tipul rezultatului obținut.

6.3.4 Determinare a minimului / maximului dintre două numere

E 6.4 Program TurboPascal de calcul a minimului/maximului a două numere

Prezentarea teoretică şi rezolvarea algoritmului problemei au fost analizate în & 2.5.3, exemplul 2.14 şi exprimat prin limbaj pseudocod în fig. 2.35 şi fig. 2.37 respectiv prin schemă logică în fig. 2.36 respectiv fig. 2.38.

Fig. 6.7

```
PROGRAM minim2;
VAR a,b,min: REAL;
BEGIN
   WRITELN('Introduceti doua numere');
   READLN(a,b);
   IF a<b THEN
      min:=a
   ELSE
      min:=b;
   WRITELN('Minim=',min: 9: 3);
END.
```

Fig. 6.8a

```
Introduceti doua numere
124 15.18
Minim=    15.180
```

```
Introduceti doua numere
16.8 92.14
Minim=    16.800
```

Fig. 6.8b

Fig. 6.9

```
PROGRAM maxim2;
VAR a,b,max: REAL;
BEGIN
   WRITELN('Introduceti doua numere');
   READLN(a,b);
   IF a>b THEN
      max:=a
   ELSE
      max:=b;
   WRITELN('Maxim=',max: 9: 3);
END.
```

Fig. 6.10a

```
Introduceti doua numere
123.5 14
Maxim=   123.500
```

```
Introduceti doua numere
11.98 45.44
Maxim=    45.440
```

Fig. 6.10b

Prima linie a programului, fig. 6.7, fig. 6.9, defineşte numele său (& 4.4), iar în a doua linie sunt declarate (& 5.1) variabilele programului ca fiind de tip real (& 4.2.3), respectiv datele de intrare **a**, **b,** şi data de ieşire **min** respectiv **max**. Secțiunea programului, inclusă între **BEGIN** şi **END.**, conține o instrucțiune de citire a datelor de intrare (& 5.5.1), precedată de o instrucțiune de afişare cu rol de atenționare a introducerii datelor; valoarea mărimii căutate rezultă prin compararea celor două numere şi atribuirea valorii minime / maxime variabilei corespunzătoare. Fig. 6.8 respectiv fig. 6.10 prezintă câte două variante ale execuției programului. Se remarcă faptul că condiția din instrucțiunea **IF** este diferența fundamentală dintre cele două programe, în rest modificările nu sunt substanțiale.

6.3.5 Intersecţia a două intervale

E 6.5 Program TurboPascal de calcul a intersecției a două intervale

Prezentarea teoretică şi rezolvarea algoritmului problemei au fost analizate în & 2.5.4, exemplul 2.15 şi exprimat prin limbaj pseudocod în fig. 2.40 respectiv prin schemă logică în fig. 2.41.

Sunt prezentate 4 variante de execuție ale programului, cu datele de intrare din tabelul asociat paragrafului & 2.5.4.

Fig. 6.11

```
PROGRAM intersec;
{Intersectia a doua intervale [a,b] si [c,d]}
VAR a,b,c,d,e,f: REAL;
BEGIN
   WRITELN('Intersectia intervalelor [a,b] si [c,d]');
   WRITELN('Introduceti limitele intervalelor ');
   READLN(a, b, c, d);
   IF a > c THEN e := a ELSE e := c;
   IF b < d THEN f := b ELSE f := d;
   IF e <= f THEN
      WRITELN('Intersectia este [',e: 9: 3,f:9 :3,']')
   ELSE
      WRITELN('Multime vida');
   READLN
END.
```

```
Intersectia intervalelor [a,b] si [c,d]
Introduceti limitele intervalelor
3 5 7 9
Multime vida

Intersectia intervalelor [a,b] si [c,d]
Introduceti limitele intervalelor
3 8 5 10
Intersectia este [    5.000    8.000]

Intersectia intervalelor [a,b] si [c,d]
Introduceti limitele intervalelor
5 9 1 7
Intersectia este [    5.000    7.000]

Intersectia intervalelor [a,b] si [c,d]
Introduceti limitele intervalelor
5 8 1 3
Multime vida
```

Fig. 6.12

6.3.6 Determinarea minimului dintre patru numere

E 6.6 Program TurboPascal pentru determinarea minimului a 4 numere

Prezentarea teoretică şi rezolvarea algoritmului problemei au fost analizate în & 2.5.5, exemplul 2.16 şi exprimat prin limbaj pseudocod în fig. 2.42 şi prin schemă logică în fig. 2.43, varianta 1, respectiv prin limbaj pseudocod în fig. 2.44, varianta 2.

```
PROGRAM minim4_varianta1;
VAR a,b,c,d,min: REAL;
BEGIN
   WRITELN('Introduceti 4 numere');
   READLN(a, b, c, d);
   min := a;
   IF b < min THEN min := b;
   IF c < min THEN min := c;
   IF d < min THEN min := d;
   WRITELN('Minim=',min:9 :3);
   READLN
END.
```

Fig. 6.13

```
PROGRAM minim4_varianta2;
VAR a,b,c,d,min1, min2, min: REAL;
BEGIN
   WRITELN('Introduceti 4 numere');
   READLN(a, b, c, d);
   IF a < b THEN min1 := a ELSE min1 := b;
   IF c < d THEN min2 := c ELSE min2 := d;
   IF min1 < min2 THEN min := min1
                  ELSE min := min2;
   WRITELN('Minim=',min:9 :3);
   READLN
END.
```

Fig. 6.14

```
Introduceti 4 numere
1 5 -3 12.8
Minim=   -3.000
```

Fig. 6.15

Prima linie a programului, fig. 6.13, fig. 6.14, defineşte numele său (& 4.4), iar în a doua linie sunt declarate (& 5.1) variabilele programului ca fiind de tip real (& 4.2.3), respectiv datele de intrare **a**, **b, c**, **d,** şi data de ieşire **min**; pentru varianta 2 sunt declarate reale şi variabilele intermediare **min1** şi **min2**. Secțiunea programului, inclusă între **BEGIN** şi **END.**, conține o instrucțiune de citire a datelor de intrare (& 5.5.1), precedată de o instrucțiune de afişare cu rol de atenționare a introducerii datelor. În continuare:

- pentru prima variantă, valoarea minimă căutată rezultă inițializarea variabilei **min** cu prima din cele patru valori, urmat de compararea celorlalte trei numere cu variabila **min** şi memorarea în variabila **min** a valorii oricăreia din acestea trei valori, dacă este mai mică decât a valorii curente a variabilei **min**;
- pentru a doua variantă, valoarea minimă căutată rezultă prin găsirea minimului local **min1** respectiv **min2**, ca valori minime între **a** şi **b** respectiv **c** şi **d**, urmând ca valoarea minimă absolută **min** să fie determinată ca fiind minimul între **min1** şi **min2**.

Valoarea minimă memorată în variabila **min** este afişată prin instrucțiunea de scriere. Instrucțiunea finală **READLN** are rol de aşteptare a apăsării tastei **Enter**, pentru a menține afişarea ecranului utilizator cu rezultatele programului, fig. 6.15.

6.3.7 Rezolvarea sistem două ecuații cu două necunoscute

E 6.7 Program TurboPascal pentru rezolvare sistem 2 ecuații 2 necunoscute

Prezentarea teoretică şi rezolvarea algoritmului au fost analizate în & 2.5.7, exemplul 2.18 şi exprimat prin limbaj pseudocod în fig. 2.47 şi schemă logică în fig. 2.48.

```
PROGRAM sistem2;
{Sistem 2 ecuatii 2 necunoscute}
VAR a,b,c,d,e,f,N,X,Y: REAL;
BEGIN
   WRITELN('Coeficientii a,b,c,d,e,f ?');
   READLN(a, b, c, d, e, f);
   N := a*e-b*d;
   IF N <> 0 THEN
      BEGIN
         X:=(c*e-b*f)/N;
         Y:=(a*f-c*d)/N;
         WRITELN('X=',x: 9: 3,'Y=',y:9 :3);
      END
   ELSE
      WRITELN('Sistemul nu are solutie');
   READLN
END.
```

IC

Fig. 6.16

Fig. 6.17a

```
Coeficientii a,b,c,d,e,f ?
2.5 3 17 0.1 -0.5 -1.8
X=    2.000  Y=    4.000
```

```
Coeficientii a,b,c,d,e,f ?
1 2 4 1 2 9
Sistemul nu are solutie
```

Fig. 6.17b

Prima linie a programului, fig. 6.16, defineşte numele său (& 4.4), a doua linie reprezintă un comentariu referitor la conținutul său (& 4.1), iar în a treia linie sunt declarate (& 5.1) variabilele programului ca fiind de tip real (& 4.2.3), respectiv datele de intrare **a**, **b, c**, **d, e**, **f** şi datele de ieşire **x** şi **y**; este declarată de asemenea şi variabila intermediare **N**. Secțiunea programului, inclusă între **BEGIN** şi **END.**, conține o instrucțiune de citire a datelor de intrare (& 5.5.1), precedată de o instrucțiune de afişare cu rol de atenționare a introducerii datelor. În continuare se calculează prin atribuire (& 5.2) valoarea numitorului **N**, valoarea care se testează comparativ cu 0;

- dacă condiția **N <> 0** este adevărată, se pot calcula prin atribuire soluțiile şi se pot afişa; aceste patru instrucțiuni sunt alocate ramurii THEN a condiției şi de aceea trebuie încadrate între cuvintele cheie **BEGIN – END**, formând o instrucțiune compusă (& 5.3);
- dacă condiția **N <> 0** este falsă, soluțiile nu pot fi calculate, deoarece o împărțire cu 0 generează eroare la execuție, motiv pentru care se va afişa un mesaj informativ.

Instrucțiunea finală **READLN** are rol de aşteptare a apăsării tastei **Enter**, pentru a menține afişarea ecranului utilizator cu rezultatele programului, fig. 6.17, unde sunt prezentate două variante ale execuției programului.

6.3.8 Identificare număr introdus de la tastatură

E 6.8 Program TurboPascal pentru identificare număr introdus de la tastatură

Se introduce de la tastatură un număr întreg şi se cere identificarea domeniului în care acesta se încadrează: număr par sau impar mai mic decât 10, număr cuprins între 10-20 sau 30-40, respectiv număr mai mare ca 40 sau negativ.

```
PROGRAM identif_numar;
{Identificare numar introdus de la tastatura}
VAR numar: INTEGER;
BEGIN
   WRITELN('Introduceti un numar');
   READLN(numar);
   CASE numar OF
      0,2,4,6,8: WRITELN('Numar par');
      1,3,5,7,9: WRITELN('Numar impar');
      10..20,30..40: WRITELN('Nr. între 10-20 sau 30-40');
   ELSE
      WRITELN('Numar mai mare ca 40 sau negativ');
   END;
READLN
END.
```

Fig. 6.18

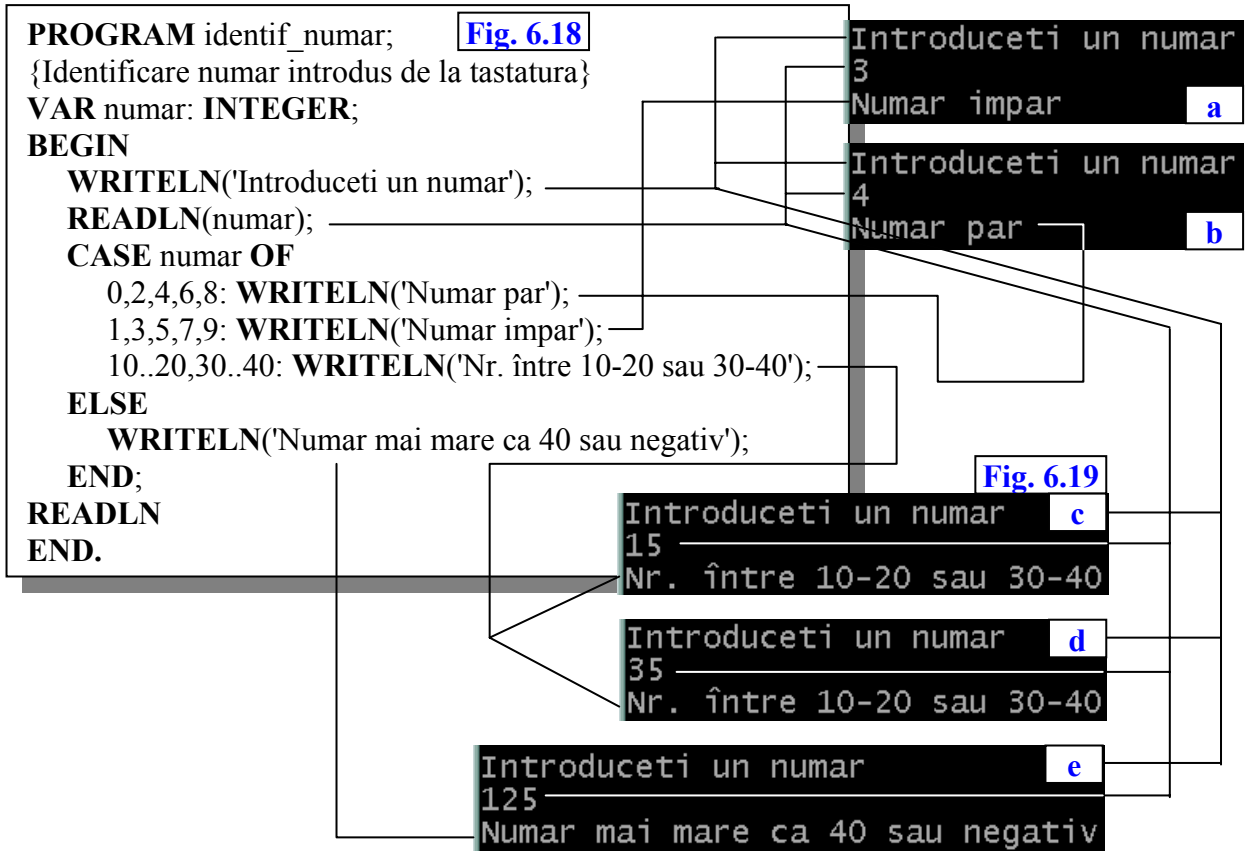


Fig. 6.19

Programul prezintă un exemplu didactic de aplicare a instrucțiunii de selecție multiplă **CASE** (& 6.2), unde, valoarea citită a variabilei de tip întreg "**i**", reprezintă *selectorul*, iar valorile / domeniile din fața instrucțiunilor **WRITELN** reprezintă *constantele de caz* :

- pentru oricare din valorile selectorului egală cu una din valorile constantelor de caz 0,2,4,6,8, se va afișa mesajul 'Număr par', fig. 6.19b;
- pentru oricare din valorile selectorului egală cu una din valorile constantelor de caz 1,3,5,7,9, se va afișa mesajul 'Număr impar', fig. 6.19a;
- dacă valoarea selectorului se încadrează în domeniul de la 10 până la 20 sau de la 30 până la 40, se va afișa mesajul 'Nr. între 10-20 sau 30-40', fig. 6.19c, d;
- dacă valorile selectorului nu se încadrează în nici una din valorile analizate anterior (**ELSE**), se va afișa mesajul 'Număr mai mare ca 40 sau negativ', fig. 6.19e.

6.3.9 Identificare caracter introdus de la tastatură

6.9 Program TurboPascal pentru identificare caracter introdus de la tastatură

Se introduce de la tastatură un caracter şi se cere identificarea domeniului în care acesta se încadrează: majusculă, minusculă, cifră, spațiu sau semn special.

```
PROGRAM identif_caracter;
{Identificare caracter introdus de la tastatura}
VAR c: CHAR;
BEGIN
   WRITELN('Introduceti un caracter');
   READLN(c);
   CASE c OF
      'A'..'Z' : WRITELN('Majuscule');
      'a'..'z' : WRITELN('Minuscule');
      '0'..'9' : WRITELN('Cifra');
      ' '      : WRITELN('Spatiu');
   ELSE
      WRITELN('Semn special');
   END;
READLN
END.
```

Fig. 6.20

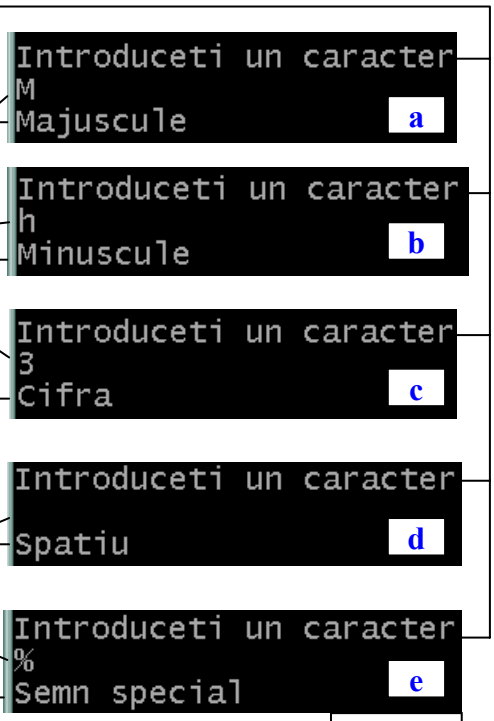


Fig. 6.21

Programul prezintă un exemplu didactic de aplicare a instrucțiunii de selecție multiplă **CASE** (& 6.2), unde, valoarea citită a variabilei de tip caracter "c", reprezintă *selectorul*, iar valorile / domeniile din fața instrucțiunilor **WRITELN** reprezintă *constantele de caz* :

- pentru oricare din valorile selectorului cuprinse în domeniul de la "**A**" la "**Z**", se va afișa mesajul 'Majuscule', fig. 6.21a;
- pentru oricare din valorile selectorului cuprinse în domeniul de la "**a**" la "**z**", se va afişa mesajul 'Minuscule', fig. 6.21b;
- pentru oricare din valorile selectorului cuprinse în domeniul de la "**0**" la "**9**", se va afişa mesajul 'Cifră, fig. 6.21c;
- dacă se apasă tasta **SPACE** (spațiu), se va afișa mesajul 'Spatiu, fig. 6.21d;
- dacă valorile selectorului nu se încadrează în nici una din valorile analizate anterior (ramura **ELSE**), se va afişa mesajul 'Semn special', fig. 6.21e.

7. INSTRUCȚIUNI REPETITIVE

Instrucțiunile repetitive specifică execuția de mai multe ori a unei singure instrucțiuni sau a unui grup de instrucțiuni. Dacă numărul de repetiții se cunoaşte se poate folosi instrucțiunea **FOR**, în caz contrar se folosesc instrucțiunile **WHILE** şi **REPEAT**.

7.1 INSTRUCȚIUNEA "WHILE"

Instrucțiunea **WHILE** conține o expresie logică ce controlează printr-o condiție execuția repetată a unei instrucțiuni (care poate fi şi compusă).

Sintaxa instrucțiunii este următoarea:

WHILE *condiție* **DO**
instructiune

WHILE *condiție* **DO**
BEGIN
Instrucțiune1;
Instrucțiune2;
……
END

IC

Instrucțiunea *instrucțiune* se execută repetat, atât timp cât *condiție* este adevărată. Dacă aceasta are valoarea falsă sau devine falsă, instrucțiunea situată după **DO** nu se mai execută. Este deci posibil ca instrucțiunea să nu se execute nici măcar o dată, deoarece evaluarea condiției se face chiar şi înainte de prima execuție a buclei. De aceea această instrucțiune se mai numeşte şi ciclu (buclă) cu test inițial.

Condiția se evaluează la fiecare parcurgere a buclei, deci instrucțiunea componentă trebuie scrisă astfel ca să modifice condiția, pentru a o putea trece din starea adevărat **TRUE** în starea **FALSE**, în caz contrar ciclul se repetă la infinit.

Instrucțiunea **WHILE** reprezintă concretizarea în TurboPascal a structurii repetitive – ciclu cu test inițial, analizată la descrierea algoritmilor (& 2.4.3.1).

Instrucțiunea compusă se foloseşte în mod obligatoriu atunci când algoritmul necesită execuția repetată a unui grup de instrucțiuni, intrând în contradicție cu sintaxa instrucțiunii **WHILE**, care impune execuția unei singure instrucțiuni. În aceste situații, grupul de instrucțiuni se încadrează între cuvintele cheie **BEGIN** – **END**, formând o instrucțiune compusă (& 5.3), care este consideră de către compilator din punct de vedere sintactic ca şi cum ar fi o singură instrucțiune.

O eroare care trebuie evitată este plasarea unui separator "**;**" după cuvântul cheie **DO**. TurboPascal va interpreta aceasta ca un ciclu **WHILE** cu o instrucțiune vidă, urmat de o altă instrucțiune, iar compilatorul nu va genera eroare sintactică.

greşeală

WHILE *condiție* **DO**;
instrucțiune

7.2 INSTRUCȚIUNEA "REPEAT-UNTIL"

Instrucțiunea **REPEAT** conține o expresie logică ce permite execuția unei secvențe de instrucțiuni situate între cuvintele cheie **REPEAT** şi **UNTIL**. Sintaxa instrucțiunii este următoarea:

REPEAT
Instrucțiune1;
Instrucțiune2;
……
UNTIL *condiție*

Instrucțiunile cuprinse între cuvintele cheie **REPEAT** şi **UNTIL** se execută atâta timp cât condiția este falsă. Când condiția devine adevărată, se va trece la execuția instrucțiunii de după **UNTIL**.

Execuția instrucțiunilor are loc cel puțin odată, deoarece evaluarea condiției se face după execuția secvenței de instrucțiuni, motiv pentru care acest ciclu se numeşte ciclu (buclă) cu test final. Instrucțiunea **REPEAT – UNTIL** reprezintă concretizarea în TurboPascal a structurii repetitive – ciclu cu test final, analizată la descrierea algoritmilor (& 2.4.3.2).

În cadrul ciclului pot fi plasate mai multe instrucțiuni, încadrate între cuvintele cheie **REPEAT – UNTIL**, deci aici nu este neapărat necesară utilizarea instrucțiunii compuse (& 5.3), însă încadrarea instrucțiunilor între cuvintele cheie **BEGIN-END** nu este însă o eroare.

Condiția se evaluează la fiecare parcurgere a buclei, deci instrucțiunea componentă trebuie scrisă astfel ca să modifice condiția, pentru a o putea trece din starea **FALSE** în starea **TRUE**, în caz contrar ciclul se repetă la infinit.

Repetițiile **REPEAT – UNTIL** pot fi îmbricate (incluse unele în altele), situație în care ele se grupează după principiul parantezelor.

7.3 INSTRUCȚIUNEA "FOR"

Instrucțiunile repetitive **FOR** se folosesc acolo unde se execută instrucțiuni în mod repetat, de un număr de ori cunoscut. Se mai numesc şi ciclu (buclă) cu contor, deoarece utilizează o variabilă numită *contor*, care trebuie să fie de tip ordinal (& 4.2), trebuie declarată în blocul care conține ciclul şi care ține evidența numărului de repetiții efectuate. Sintaxa instrucțiunii este următoarea:

FOR *contor* := V_i **TO** V_f **DO**
Instrucțiune

FOR *contor* := V_i **DOWNTO** V_f **DO**
Instrucțiune

FOR *contor* := V_i **TO** V_f **DO**
BEGIN
IC *Instrucțiune1;*
Instrucțiune2;
……
END

FOR *contor* := V_i **DOWNTO** V_f **DO**
BEGIN
IC *Instrucțiune1;*
Instrucțiune2;
……
END

Contorul este inițializat cu o valoare inițială **Vi**, este incrementat (forma **TO**) sau decrementat (forma **DOWNTO**) la fiecare execuție a ciclului, iar ciclul se încheie atunci când contorul atinge valoarea finală **Vf** prestabilită.

Instrucțiunea compusă se foloseşte în mod obligatoriu atunci când algoritmul necesită execuția repetată a unui grup de instrucțiuni, intrând în contradicție cu sintaxa instrucțiunii **FOR**, care impune execuția unei singure instrucțiuni. În aceste situații, grupul de instrucțiuni se încadrează între cuvintele cheie **BEGIN – END**, formând o instrucțiune compusă (& 5.3), care este consideră de către compilator din punct de vedere sintactic ca şi cum ar fi o singură instrucțiune.

Instrucțiunea **FOR** reprezintă concretizarea în TurboPascal a structurii repetitive – ciclu cu contor, analizată la descrierea algoritmilor (& 2.4.3.3).

OBSERVATII

1. Ciclul se execută de **N** ori, unde **N** se evaluează o singură dată, la începutul ciclului, după formula:

$N = V_f - V_i + 1$ în cazul ciclului **TO**
$N = V_i - V_f + 1$ în cazul ciclului **DOWNTO**

2. V_i şi V_f trebuie să fie compatibile cu tipul variabilei contor; ele sunt evaluate o singură dată, la începutul ciclului.

3. Pentru varianta **TO** se impune ca $V_i <= V_f$, iar valorile contorului sunt incrementate la fiecare execuție a ciclului, adică se trece la succesorul valorii contorului. Dacă $V_i > V_f$ ciclul nu se execută niciodată.

4. Pentru varianta **DOWNTO** se impune ca $V_i < V_f$, iar valorile contorului sunt decrementate la fiecare execuție a ciclului, adică se trece la predecesorul valorii contorului. Dacă $V_i < V_f$ ciclul nu se execută niciodată.

5. Dacă $V_i = V_f$, ciclul se execută o dată, indiferent de forma sintactică a ciclului **FOR**.

6. Modificarea contorului nu este semnalată ca eroare, dar poate produce erori la execuție. Contorul poate fi însă folosit în expresii sau condiții. Nu se permite:

- modificarea contorului în interiorul ciclului, prin atribuire, citire.
- contorul nu poate fi parametru actual corespunzător unui parametru formal variabil **var**, dintr-o procedură.
- contorul unui ciclu nu poate fi contor şi pentru alt ciclu.

8. Ciclurile pot fi îmbricate (incluse unul în altul), dar cu contoare diferite. În această situație, regula de execuție este următoarea: ciclurile interioare se execută mai repede decât ciclurile exterioare.

7.4 APLICAȚII

7.4.1 Calculul radical prin recurență (varianta ciclu cu test inițial)

E 7.1 Program TurboPascal pentru calculul radical prin recurență

Prezentarea teoretică şi rezolvarea algoritmului problemei au fost analizate detaliat în & 2.4.3.1, exemplul 2.9 şi exprimat prin limbaj pseudocod, în fig. 2.21 respectiv prin schemă logică, fig. 2.22, motiv pentru care toate raționamentele rămân valabile.

Fig. 7.1

```
PROGRAM Radical1;
{ Calcul radical prin relatie de recurenta }
VAR x, eps, Xant, Xcrt : REAL;
BEGIN
   WRITELN('Introduceti numarul');
   READLN(x);
   WRITELN('Precizia de calcul');
   READLN(eps);
   Xant := 1;
   Xcrt := 0.5 * (Xant + X / Xant);
   WRITELN('Xant=',Xant: 9: 6, 'Xcrt=',Xcrt: 9 : 6);
   WHILE ABS(Xcrt - Xant) >= eps DO
     BEGIN
IC     Xant := Xcrt;
       Xcrt := 0.5 * (Xant + X/Xant);
       WRITELN('Xant=',Xant: 9: 6, 'Xcrt=',Xcrt: 9 : 6);
     END;
   WRITELN(' radical final =', Xcrt: 9 : 6);
   READLN
END.
```

Fig. 7.2

```
Introduceti numarul
13
Precizia de calcul
0.000001
Xant= 1.000000Xcrt= 7.000000
Xant= 7.000000Xcrt= 4.428571
Xant= 4.428571Xcrt= 3.682028
Xant= 3.682028Xcrt= 3.606345
Xant= 3.606345Xcrt= 3.605551
Xant= 3.605551Xcrt= 3.605551
radical final = 3.605551
```

După declararea variabilelor programului (& 5.1), sunt citite datele de intrare **x** şi **eps** prin instrucțiuni de citire, precedate de instrucțiuni de afişare cu rol de atenționare a introducerii datelor. Prin instrucțiuni de atribuire se inițializează variabila **Xant** cu valoarea 1, se calculează (prin relația de recurență) şi se afişează prima valoarea a variabilei **Xcrt**, valori care intră în condiția ciclului **WHILE**. Instrucțiunile repetitive sunt incluse în instrucțiunea compusă şi constau în: două atribuiri, prin care variabila **Xant** preia valoarea variabilei **Xcrt**, respectiv recalcularea lui **Xcrt** prin relația de recurență şi o instrucțiune de afişare a valorilor momentane ale variabilelor **Xant** şi **Xcrt**. Se iese din repetiție la trecerea condiției din instrucțiunea **WHILE** din starea **TRUE** în starea **FALSE**, deci la obținerea unei diferențe între **Xcrt** şi **Xant** (în valoare absolută) mai mică decât valoarea **eps** impusă. Afişarea din interiorul repetiției nu este necesară, ci de control, pentru a evidenția evoluția numerică a calculelor în timpul repetiției.

7.4.2 Calculul radical prin recurență (varianta ciclu cu test final)

E 7.2 Program TurboPascal pentru calculul radical prin recurență

Prezentarea teoretică şi rezolvarea algoritmului problemei au fost analizate detaliat în & 2.4.3.2, exemplul 2.10 şi exprimat prin limbaj pseudocod, în fig. 2.25 respectiv prin schemă logică, fig. 2.26, motiv pentru care toate raționamentele rămân valabile.

```
PROGRAM Radical2;
{ Calcul radical prin relatie de recurenta }
VAR x, eps, Xant, Xcrt : REAL;
BEGIN
   WRITELN('Introduceti numarul');
   READLN(x);
   WRITELN('Precizia de calcul');
   READLN(eps);
   Xcrt := 1;
   REPEAT
      Xant := Xcrt;
      Xcrt := 0.5 * (Xant + X/Xant);
      WRITELN('Xant=',Xant: 9: 6, 'Xcrt=',Xcrt: 9 : 6);
   UNTIL ABS(Xcrt - Xant) < eps;
   WRITELN('radical final=', Xcrt: 9 : 6);
   READLN
END.
```

Fig. 7.3

```
Introduceti numarul
13
Precizia de calcul
0.000001
Xant= 1.000000Xcrt= 7.000000
Xant= 7.000000Xcrt= 4.428571
Xant= 4.428571Xcrt= 3.682028
Xant= 3.682028Xcrt= 3.606345
Xant= 3.606345Xcrt= 3.605551
Xant= 3.605551Xcrt= 3.605551
radical final= 3.605551
```

Fig. 7.4

După declararea variabilelor programului (& 5.1), sunt citite datele de intrare **x** şi **eps** prin instrucțiuni de citire, precedate de instrucțiuni de afişare cu rol de atenționare a introducerii datelor. Prin instrucțiune de atribuire se inițializează variabila **Xcrt** cu valoarea 1. Instrucțiunile repetitive sunt incluse între cuvintele cheie **REPEAT - UNTIL** şi constau în: două atribuiri, prin care variabila **Xant** preia valoarea variabilei **Xcrt**, respectiv recalcularea lui **Xcrt** prin relația de recurență şi o instrucțiune de afişare a valorilor momentane ale variabilelor **Xant** şi **Xcrt**. Se iese din repetiție la trecerea condiției din instrucțiunea repetitivă din starea **FALSE** în starea **TRUE**, deci la obținerea unei diferențe între **Xcrt** şi **Xant** (în valoare absolută) mai mică decât valoarea **eps** impusă. Afişarea din interiorul repetiției nu este necesară, ci de control, pentru a evidenția evoluția numerică a calculelor în timpul repetiției.

În ambele exemple, exemplul 7.1 şi exemplul 7.2, se utilizează repetiții condiționate, adică repetiții al căror număr nu este cunoscut apriori, ci este determinat de o condiție impusă. Evoluția valorică a variabilelor **Xant** şi **Xcrt** provoacă îndeplinirea condițiilor ciclurilor, astfel încât acestea să parcurgă un număr finit de paşi.

7.4.3 Determinarea maximului a N numere

E 7.3 Program TurboPascal pentru determinarea maximului a N numere

Prezentarea teoretică şi rezolvarea algoritmului problemei au fost analizate detaliat în & 2.4.3.3, exemplul 2.11 şi exprimat prin limbaj pseudocod, în fig. 2.29 respectiv prin schemă logică, fig. 2.30, motiv pentru care toate raționamentele rămân valabile.

Fig. 7.5

```
PROGRAM maxim_N;
{Maxim a N numere introduse de la tastatura}
VAR i, N: INTEGER;
      x, maxim: REAL;
BEGIN
   WRITELN('Maxim a N numere');
   WRITELN('Cate numere ?');
   READLN(N);
   WRITELN('X1=');
   READLN(X);
   maxim:=x;
   FOR i:=2 TO N DO
      BEGIN
         WRITELN('X',i,'=');
IC       READLN(X);
         IF x > maxim THEN maxim := x;
      END;
   WRITELN('Maxim=',maxim:9:4);
   READLN
END.
```

Fig. 7.6

```
Maxim a N numere
Cate numere ?
5
X1=
3
X2=
6
X3=
2
X4=
9
X5=
4
Maxim=    9.0000
```

Domeniu ciclu **FOR**

Principiul programului este următorul: cu valoarea primului număr se inițializează variabila **maxim**; apoi, de la al doilea număr la ultimul, numerele se citesc succesiv şi se compară cu valoarea momentană a variabilei **maxim**; numai dacă valoarea numărului curent este mai mare decât **maxim**, atunci **maxim** va prelua această valoare curentă. La sfârşitul parcurgerii tuturor numerelor, în variabila **maxim** va rămâne cea mai mare valoare.

În interiorul ciclului **FOR**, instrucțiunea compusă conține 3 instrucțiuni:

- afişarea unui mesaj cu rol de atenționare a introducerii numerelor; se observă utilizarea contorului de ciclu "**i**" în interiorul instrucțiunii **WRITELN**, pentru a genera numărul de ordine al numerelor de introdus;
- citirea succesivă a numerelor în aceeaşi variabila **x**;
- compararea prin instrucțiunea **IF** (& 6.1) a lui **x** cu **maxim** şi preluarea de către **maxim** a valorii lui **x**, dar numai dacă **x > maxim**.

7.4.4 Afișare numere impare mai mici decât N

E 7.4 Program TurboPascal pentru afișare numere impare mai mici decât N

Prezentarea teoretică şi rezolvarea algoritmului problemei au fost analizate detaliat în & 2.5.8, exemplul 2.19 şi exprimat prin limbaj pseudocod, în fig. 2.51 pentru varianta ciclului cu test inițial, respectiv în fig. 2.52 pentru varianta ciclului cu test final, motiv pentru care toate raționamentele rămân valabile.

```
PROGRAM impare1;
{Numere impare mai mici ca N}
VAR i,N:INTEGER;
BEGIN
   WRITE('N=');
   READLN(N);
   i:=1;
   WHILE i <= N DO
   BEGIN
IC    WRITELN(i);
      i:=i+2;
   END;
   READLN
END.
```

Fig. 7.7

```
PROGRAM impare2;
{Numere impare mai mici ca N}
VAR i,N:INTEGER;
BEGIN
   WRITE('N=');
   READLN(N);
   i:=1;
   REPEAT
      WRITELN(i);
      I:=i+2;
   UNTIL i > N;
   READLN
END.
```

Fig. 7.8

Fig. 7.9

```
N=15
1
3
5
7
9
11
13
15
```

Domeniu repetitie

Principiul programului este următorul: la valoarea numărul 1 se adaugă rația 2, generând astfel numerele impare, în aceeaşi variabilă "**i**". Repetitiv, deci se vor executa următoarele două acțiuni: afişarea valorii curente a variabilei "**i**" respectiv mărirea cu rația 2 a valorii sale, prin instrucțiunea de atribuire (& 5.2). Utilizarea structurii de repetiție permite deci generarea numerelor impare căutate.

Cele două variante de program diferă prin structura de repetiție utilizată, respectiv ciclul cu test inițial (& 7.1) – fig. 7.7 şi ciclul cu test final (& 7.2) – fig. 7.8, dar rezultatele programului, fig. 7.9, sunt identice. Se remarcă de asemenea faptul că condiția de finalizare a ciclului **REPEAT** este tocmai condiția contrară celei din ciclul **WHILE**.

Pentru ciclul **WHILE** instrucțiunile repetitive trebuie incluse într-o instrucțiune compusă (& 5.3).

În orice problemă, *atunci când datele pot fi generate printr-o exprimare repetitivă*, întotdeauna se va utiliza structura de repetiție, în locul introducerii manuale a datelor, pentru a evita o operație obositoare, consumatoare de timp şi susceptibilă de erori de introducere.

7.4.5 Suma a N numere citite succesiv

E 7.5 Program TurboPascal pentru suma a N numere citite succesiv

Prezentarea teoretică şi rezolvarea algoritmului problemei au fost analizate detaliat în & 2.5.9, exemplul 2.20 şi exprimat prin limbaj pseudocod, în fig. 2.53 respectiv prin schemă logică în fig. 2.54, motiv pentru care toate raționamentele rămân valabile.

```
PROGRAM suma_N;
{Suma a N numere citite succesiv}
VAR i, N: INTEGER;
   x, suma: REAL;
BEGIN
   WRITE('Cate numere insumati ?');
   READLN(N);
   suma:=0;
   FOR i:=1 TO N DO
   BEGIN
      WRITE('X',i,'=');
IC READLN(X);
      suma:= suma+X;
   END;
   WRITELN('SUMA=',suma:10:4);
   READLN
END.
```

Fig. 7.10

Fig. 7.11

```
Cate numere insumati ?5
X1=10
X2=5
X3=11
X4=6
X5=-3
SUMA=    29.0000
```

Domeniu ciclu **FOR**

Principiul programului este următorul: numerele introduse de la tastatură, în aceeaşi variabilă **X**, vor fi însumate succesiv în variabila **suma**. Repetitiv, deci se vor executa următoarele trei acțiuni:

- afişarea unui mesaj cu rol de atenționare a introducerii numerelor; se observă utilizarea contorului de ciclu "**i**" în interiorul instrucțiunii **WRITE**, pentru a genera numărul de ordine al numerelor de introdus;
- citirea valorii numerice în variabila **X**; utilizarea instrucțiunii de afişare **WRITE** provoacă conservarea poziției cursorului după semnul egal, deci citirea se va produce începând de la această poziție pe acelaşi rând, spre deosebire de instrucțiunea **WRITELN**, care provoacă citirea de pe rânduri diferite, & 7.4.3, exemplul 7.3, fig. 7.5, fig. 7.6;
- aplicarea relației de însumare relația 2.5, & 2.5.9, concretizată prin instrucțiunea de atribuire (& 5.2).

Pentru ciclul **FOR** instrucțiunile repetitive trebuie incluse într-o instrucțiune compusă (& 5.3).

7.4.6 Suma numerelor pozitive şi negative dintr-un şir de N numere

E 7.6 Program TurboPascal pentru suma nr. pozitive şi negative dintr-un şir de N numere

Prezentarea teoretică şi rezolvarea algoritmului problemei au fost analizate detaliat în & 2.5.10, exemplul 2.21 şi exprimat prin limbaj pseudocod, în fig. 2.55 respectiv prin schemă logică în fig. 2.56, motiv pentru care toate raţionamentele rămân valabile.

```
PROGRAM suma_sir;
{Suma numerelor pozitive/negative din sir}
VAR i,N: INTEGER;
   x, spoz, sneg: REAL;
BEGIN
   WRITE('Cate numere ?'); READLN(N);
   spoz:=0;
   sneg:=0;
   FOR i:=1 TO N DO
     BEGIN
       WRITE('X',i,'=');READLN(x);
       IF X>0 THEN
         spoz:=spoz+x
       ELSE
          sneg:=sneg+x;
     END;
   WRITELN('Suma pozitive=',spoz:10:4);
   WRITELN('Suma negative=',sneg:10:4);
   READLN
END.
```

IC

Fig. 7.12

```
Cate numere ?6
X1=2.5
X2=-2
X3=-1.5
X4=3
X5=4
X6=0.3
Suma pozitive=    9.8000
Suma negative=   -3.5000
```

Fig. 7.13

Domeniu ciclu **FOR**

Programul este similar celui anterior, cu diferența calculării a două sume, separat pentru numerele pozitive şi separat pentru cele negative, motiv pentru care se utilizează două variabile separate pentru depozitarea sumelor **spoz** şi **sneg**.

Numerele sunt introduse succesiv, prin citire în aceeaşi variabilă **X**. Identificarea pozitivității numerelor se realizează printr-o instrucțiune de decizie (& 6.1), prin comparația cu 0. Funcție de realizarea sau nu a condiției se aplică diferențiat, pentru cele două variabile **spoz** şi **sneg**, relația de însumare relația 2.5, & 2.5.9, concretizată prin instrucțiunile de atribuire corespondente (& 5.2). Pentru ciclul **FOR** instrucțiunile repetitive (afişare mesaj informativ, citire număr şi instrucțiunea de decizie) trebuie incluse într-o instrucțiune compusă (& 5.3), prin încadrarea acestora între cuvintele cheie **BEGIN-END**.

Se observă instrucțiunile de inițializare a variabilelor de însumare anterior ciclului repetitiv în care se execută însumarea.

Se accentuează importanța inițializării variabilelor de însumare; lipsa acestei etape poate genera erori la execuție în cazul în care suma se calculează de mai multe ori într-un ciclu repetitiv; eroarea este generată de faptul că, în lipsa inițializării, la fiecare repetiție, suma continuă valoarea obținută în repetiția anterioară, în loc să "*plece*" de la 0.

7.4.7 Produsul unui şir de N numere

E 7.7 Program TurboPascal pentru calculul produsului a N numere

```
PROGRAM produs_N;
{Produsul a N numere reale}
VAR i,N: INTEGER;
   x, P: REAL;
BEGIN
   WRITELN('Cate numere ? ');
   READLN(N);
   P:=1; {initializare produs}
   FOR i:=1 TO N DO
     BEGIN
IC     WRITE('X',i,'=');
       READLN(x);
       P:=P*x;
     END;
   WRITELN('Produsul=',P:12:3);
   READLN
END.
```

Fig. 7.14

Prezentarea teoretică şi rezolvarea algoritmului problemei au fost analizate detaliat în & 2.5.11, exemplul 2.22 şi exprimat prin limbaj pseudocod, în fig. 2.57 respectiv prin schemă logică în fig. 2.58, motiv pentru care toate raționamentele rămân valabile.

Fig. 7.15

```
Cate numere ?
5
X1=3
X2=6
X3=1
X4=2
X5=2
Produsul=     72.000
```

Principiul programului este următorul: numerele introduse de la tastatură, în aceeaşi variabilă **X**, vor fi înmulțite succesiv în variabila **P**. Repetitiv, deci se vor executa următoarele trei acțiuni:

- afişarea unui mesaj cu rol de atenționare a introducerii numerelor; se observă utilizarea contorului de ciclu "**i**" în interiorul instrucțiunii **WRITE**, pentru a genera numărul de ordine al numerelor de introdus;
- citirea valorii numerice în variabila **X**; utilizarea instrucțiunii de afişare **WRITE** provoacă conservarea poziției cursorului după semnul egal, deci citirea se va produce începând de la această poziție pe acelaşi rând, spre deosebire de instrucțiunea **WRITELN**, care provoacă citirea de pe rânduri diferite, & 7.4.3, exemplul 7.3, fig. 7.5, fig. 7.6;
- aplicarea relației de calcul a produsului relația 2.7, & 2.5.11, concretizată prin instrucțiunea de atribuire (& 5.2).

Pentru ciclul **FOR** instrucțiunile repetitive trebuie incluse într-o instrucțiune compusă (& 5.3).

Se observă instrucțiunea de inițializare a variabilei produs **P** anterior ciclului repetitiv în care se execută calculul produsului.

Se accentuează importanța inițializării variabilei produs; lipsa acestei etape poate genera erori la execuție, eroarea fiind generată de faptul că, în lipsa inițializării, variabila produs are valoarea implicită 0 şi deci inclusiv produsul final va avea ca rezultat final valoarea 0, cu rezultate în general nefavorabile asupra execuției.

7.4.8 Contorizare numere pozitive, negative şi nule dintr-un şir de N numere

E 7.8 Program TurboPascal contorizare numere pozitive, negative şi nule din N numere

```
PROGRAM cont_sir;
{Contorizare numere intregi
pozitive/negative/nule din sir}
VAR i,N,x,poz,nule,neg:INTEGER;
BEGIN
   WRITE('Cate numere ? '); READLN(N);
   poz:=0;
   nule:=0;
   neg:=0;
   FOR i:=1 TO N DO
     BEGIN
       WRITE('X',i,'=');READLN(x);
       IF X>0 THEN
         poz:=poz+1
       ELSE
         IF X=0 THEN
           nule:=nule+1
         ELSE
           neg:=neg+1;
     END;
   WRITELN('Numere pozitive=',poz);
   WRITELN('Numere nule=',nule);
   WRITELN('Numere negative=',neg);
   READLN
END.
```

IC

Domeniu ciclu **FOR**

```
Cate numere ? 5
X1=3
X2=-6
X3=0
X4=2
X5=4
Numere pozitive=3
Numere nule=1
Numere negative=1
```

Fig. 7.17

Fig. 7.16

Prezentarea teoretică şi rezolvarea algoritmului problemei au fost analizate detaliat în & 2.5.12, exemplul 2.23 şi exprimat prin limbaj pseudocod, în fig. 2.59 respectiv prin schemă logică în fig. 2.60, motiv pentru care toate raționamentele rămân valabile.

Programul "*numără*" câte numere pozitive, negative şi nule sunt într-un şir de N numere, motiv pentru care se utilizează trei variabile separate pentru depozitarea contorilor: **poz, nule** şi **neg**.

Numerele sunt introduse succesiv, prin citire, în aceeaşi variabilă **X**. Identificarea pozitivității numerelor se realizează printr-o instrucțiunea de decizie prin comparația **X>0** (& 6.1). La realizarea condiției se aplică variabilei **poz** relația de contorizare relația 2.8, & 2.5.12, concretizată prin instrucțiunea de atribuire **poz:=poz+1** (& 5.2). În caz contrar, pe ramura **ELSE** a primei instrucțiuni **IF** se execută o nouă instrucțiune de decizie prin comparația **X=0**. La realizarea condiției se aplică relația de contorizare variabilei **nule**, în caz contrar (pe ramura **ELSE** a celei de-a doua instrucțiune **IF**) se aplică relația de contorizare variabilei **neg**. Cea de-a două instrucțiune **IF** se constituie ca o unică instrucțiune asociată ramurii **ELSE** a primei instrucțiuni **IF**, după cum arată şi marcatorii grafici, din fig. 7.16. Pentru ciclul **FOR** instrucțiunile repetitive (afişare mesaj informativ, citire număr şi instrucțiunile de decizie) trebuie incluse într-o instrucțiune compusă (& 5.3), prin încadrarea acestora între cuvintele cheie **BEGIN-END**. Se observă instrucțiunile de inițializare a variabilelor contor anterior ciclului repetitiv în care se execută contorizarea. De fapt contorizarea este o sumare, în care termenul de însumat este în general 1 şi deci constituie un caz particular al sumării.

7.4.9 Sumare serie

E 7.9 Program TurboPascal sumare serie

```
PROGRAM suma_ser;
{Sumare serie:
1+1/(1*2)+2/(1*2*3)+..+(n-1)/(1*2*3*..*n)}
VAR i,j,N: INTEGER;
   prod,suma: REAL;
BEGIN
   WRITE('Cati termeni se insumeaza ? ');
   READLN(N);
   suma:=1; {initializare suma}
   FOR i:=1 TO N DO
   BEGIN
      prod:=1;
      FOR J:=1 TO i+1 DO
      BEGIN
         prod:=prod*j;
         WRITELN('Pasul j=',j,' Produs=',prod:12:4)
      END;
      suma:=suma+i/prod;
      WRITELN('   Pasul i=',i,' Suma=',suma:12:4)
   END;
   WRITELN('Suma finala serie=',suma:12:4);
   READLN;
END.
```

IC

IC

Fig. 7.18

```
Cati termeni se insumeaza ? 4
Pasul j=1 Produs=      1.0000
Pasul j=2 Produs=      2.0000
   Pasul i=1 Suma=      1.5000
Pasul j=1 Produs=      1.0000
Pasul j=2 Produs=      2.0000
Pasul j=3 Produs=      6.0000
   Pasul i=2 Suma=      1.8333
Pasul j=1 Produs=      1.0000
Pasul j=2 Produs=      2.0000
Pasul j=3 Produs=      6.0000
Pasul j=4 Produs=     24.0000
   Pasul i=3 Suma=      1.9583
Pasul j=1 Produs=      1.0000
Pasul j=2 Produs=      2.0000
Pasul j=3 Produs=      6.0000
Pasul j=4 Produs=     24.0000
Pasul j=5 Produs=    120.0000
   Pasul i=4 Suma=      1.9917
Suma finala serie=      1.9917
```

Fig. 7.19

Prezentarea teoretică şi rezolvarea algoritmului problemei au fost analizate detaliat în & 2.5.13, exemplul 2.24 şi exprimat prin limbaj pseudocod, în fig. 2.61 respectiv prin schemă logică în fig. 2.62, motiv pentru care toate raționamentele rămân valabile.

În această problemă, datele de intrare pot fi generate printr-o exprimare repetitivă, de aceea s-au utilizat structurile de repetiție, în locul introducerii manuale a datelor, pentru a evita o operație obositoare, consumatoare de timp şi susceptibilă de erori de introducere. Acesta program este un exemplu de includere a unui ciclu (ciclu interior de contor "**j**") în alt ciclu (ciclu exterior de contor "**i**"), iar fig. 7.19 exemplifică numeric regula de funcționare a ciclurilor îmbricate (& 7.3): ciclul interior se execută mai repede decât cel exterior. Cu alte cuvinte, contorul ciclul interior "**j**" parcurge complet domeniul de valori impus (de la **1** la **i+1**, unde valoarea "**i**" este variabilă) înainte de a se trece la un nou pas pentru ciclul exterior. Desigur că aceasta înseamnă, că, la fiecare pas al ciclului exterior, ciclul interior este reluat din nou de la valoarea **1** şi parcurs complet până la valoarea curentă a contorului ciclului exterior "**i+1**".

Afişarea din interiorul repetiţiilor nu este necesară, ci de control, pentru a evidenția evoluția numerică a calculelor în timpul repetiției.

Pentru fiecare ciclul **FOR**, instrucțiunile repetitive trebuie incluse într-o instrucțiune compusă (& 5.3), prin încadrarea acestora între cuvintele cheie **BEGIN-END**. Astfel, primul ciclu **FOR** include următoarele instrucțiuni repetitive:

- inițializarea variabilei produs cu valoarea **1**;
- execuția ciclului interior, de contor "**j**", pentru calculul produsului;
- aplicarea relației de însumare relația 2.5, & 2.5.9, concretizată prin instrucțiunea de atribuire (& 5.2) **suma:=suma+i/prod**;
- instrucțiunea de afişare a contorului "**i**" şi a valorii momentane a variabilei **suma**, aceasta nefiind obligatorie pentru logica programului.

Al doilea ciclu **FOR** include următoarele instrucțiuni repetitive:

- aplicarea relației de calcul a produsului relația 2.7, & 2.5.11, concretizată prin instrucțiunea de atribuire (& 5.2) **prod:=prod*j**.
- instrucțiunea de afişare a contorului "**j**" şi a valorii momentane a variabilei **prod**, aceasta nefiind obligatorie pentru logica programului.

Se observă instrucțiunea de inițializare a variabilei de însumare, anterior ciclului repetitiv în care se execută însumarea. De asemenea, la fiecare pas al ciclului de contor "**i**", se execută inițializarea variabilei produs **prod** cu valoarea **1**, ceea ce este foarte important, deoarece, în lipsa inițializării, valoarea finală a produsului este 0, ceea ce ar provoca oprirea cu eroare **Division by zero** a programului, la execuția instrucțiunii de însumare, **suma:=suma+i/prod,** datorită încercării de împărțire cu **0,** lucru ce rezultă din fig. 7.20, unde, pentru a forța eroarea, instrucțiunea de inițializarea s-a inclus în comentariu, fiind deci ignorată la compilarea şi execuția programului.

```
 Error 200: Division by zero.          Semnalizare eroare
VAR i,j,N: INTEGER;
    prod,suma: REAL;
BEGIN
     WRITE('Cati termeni se insumeaza ? ');
     READLN(N);
     suma:=1; {initializare suma}
     FOR i:=1 TO N DO
         BEGIN
              {prod:=1;} ———— Comentariu
              FOR J:=1 TO i+1 DO
                  BEGIN
                     prod:=prod*j;
                     WRITELN('Pasul j=',j,' Produs=',prod:12:4)
                  END;
              suma:=suma+i/prod; ————————————
              WRITELN('   Pasul i=',i,' Suma=',suma:12:4);
         END;
     WRITELN('Suma finala serie=',suma:12:4);
     READLN;
END.
```

Fig. 7.20

7.4.10 Produs a două numere întregi prin adunări repetate

E 7.10 Program TurboPascal pentru calcul produs a două numere întregi prin adunări repetate

Calculul produsului a două numere întregi poate fi efectuat conform algoritmului clasic de înmulțire sau prin procedeul adunărilor repetate, exemplificat mai jos:

$$12 = 3 \times 4 = \overbrace{3+3+3+3}^{\text{De 4 ori}} = \overbrace{4+4+4}^{\text{De 3 ori}}$$

Produsul **12** poate fi obținut în mod direct, înmulțind valoarea **3** cu valoarea **4**, sau prin adunări repetate, adunând de **4** ori numărul **3** sau adunând de **3** ori numărul **4**. Vom exemplifica a doua variantă de calcul prin program TurboPascal. Ideea de bază este că înmulțirea se calculează prin intermediul unei sume, deci vom aplica algoritmul de sumare (& 7.4.5, exemplul 7.5), iar sumarea se realizează printr-o repetiție de tip **FOR**, deoarece cunoaștem numărul de repetiții asociat sumării. Astfel, pentru două numere întregi **N1** și **N2**, sumarea constă în adunarea numărului **N1** repetată de **N2** ori.

```
PROGRAM produs_prin_adunare;
{Calculul produs a doua numere
intregi prin adunari repetate}
VAR N1,N2,i,suma: INTEGER;
BEGIN
   WRITELN('Produs prin adunari repetate');
   WRITELN('Introduceti doua numere intregi');
   READLN(N1, N2);
   suma:= 0; {Initializare variabila suma}
   FOR i:=1 TO N2 DO suma:=suma+N1;
   WRITELN('Suma prin adunare=',suma:6);
   WRITELN('Suma prin produs =',N1*N2:6);
   READLN
END.
```

Fig. 7.21

```
Produs prin adunari repetate
Introduceti doua numere intregi
3 4
Suma prin adunare=    12
Suma prin produs =    12
```

Fig. 7.22

După declararea variabilelor programului (& 5.1), sunt citite datele de intrare **N1**, **N2** instrucțiuni de citire, precedate de două instrucțiuni de afișare cu rol de atenționare a introducerii datelor. Prin instrucțiunea de atribuire se inițializează cu 0 variabila **suma**; ciclul repetitiv **FOR** are o singură instrucțiune asociată și anume aplicarea relației de sumare relația 2.5, & 2.5.9. Se remarcă faptul că cantitatea de însumat este **N1** iar ciclul se execută de **N2** ori.

În final se afișează formatat, prin instrucțiunile **WRITELN**, valoarea produsului calculată prin sumare comparativ cu cea calculată prin produsul celor două numere întregi. Scopul algoritmului a fost de exemplificare practică a repetiției asociată cu algoritmul sumării și nu de recomandare a utilizării acestui procedeu ca înlocuitor al procedeului clasic de înmulțire a două numere întregi.

7.4.11 Împărţire a două numere întregi prin scăderi repetate

E 7.11 Program TurboPascal pentru împărţire a două numere întregi prin scăderi repetate

Calculul împărțirii a două numere întregi poate fi efectuat conform algoritmului clasic de împărțire sau prin procedeul scăderilor repetate, exemplificat mai jos. Astfel, pentru două numere întregi **N1**=16 şi **N2**=3, calculul direct oferă câtul 5 respectiv restul 1. Procedeul scăderilor repetate pleacă de la valoarea numărului **N1** din care se scade numărul **N2**, obținând un rest inițial, la se aplică repetitiv scăderea lui **N2**, după regula : restul curent este egal cu cel anterior din care se scade **N2**; algoritmul continuă cât timp restul obținut prin scădere este mai mare sau cel mult egal cu **N2**. Numărul de scăderi efectuate este câtul împărțirii, iar ultimul rest este cel final. Elementele "*informatice*" ce trebuie evidențiate sunt: structura repetitivă a operației şi necesitatea contorizării numărului de scăderi efectuate.

N1 = 16 N2 = 3
Cat = 5 Rest = 1

16 – 3 = 13	
13 – 3 = 10	
10 – 3 = 7	5 scăderi
7 – 3 = 4	
4 - 3 = 1 ←	rest

```
PROGRAM cat_prin_scadere;
{Impartire a doua numere intregi prin
scaderi repetate cu afisare Cat si Rest}
VAR N1,N2,cat,rest:INTEGER;
BEGIN
WRITELN('Introduceti Deimpartit si Impartitor');
READLN(N1,N2);
IF N1<N2 THEN
  BEGIN
    cat:=0;
    rest:=N1;
  END
ELSE
  BEGIN
    rest:=N1-N2;  {Calculul primului rest}
    cat:=1;  {Contorizarea primei scaderi}
    WHILE rest >= N2 DO
      BEGIN
        rest:=rest-N2;
        cat:=cat+1;
      END;
  END;
WRITELN('Prin scadere: Catul=',CAT:5,'  Restul=',REST:5);
WRITELN('Direct:      Catul=',N1 DIV N2:5,'  Restul=',N1 MOD N2:5);
READLN
END.
```

IC

IC

IC

Fig. 7.23

Fig. 7.24a

```
Introduceti Deimpartit si Impartitor
16 3
Prin scadere: Catul=    5  Restul=    1
Direct:       Catul=    5  Restul=    1
```

Fig. 7.24b

```
Introduceti Deimpartit si Impartitor
3 10
Prin scadere: Catul=    0  Restul=    3
Direct:       Catul=    0  Restul=    3
```

După declararea variabilelor programului (& 5.1), sunt citite datele de intrare **N1** şi **N2** printr-o instrucțiune de citire, precedată de o instrucțiune de afişare cu rol de atenționare a introducerii datelor.

Algoritmul prezintă o excepție, fig. 7.24b: dacă **N1** < **N2**, câtul este 0, iar restul este egal cu **N1**, caz evidențiat prin ramura **THEN** a instrucțiunii **IF**, prin două instrucțiuni de atribuire (& 5.2) încadrate într-o instrucțiune compusă (& 5.3).

Pe ramura **ELSE** a instrucțiunii **IF** se parcurge varianta normală a algoritmului fig. 7.24a, printr-o instrucțiune compusă. Astfel, se efectuează prima scădere pentru obținerea primului rest şi în consecință se inițializează cu 1 valoarea variabilei **cat** (variabilă care contorizează numărul de scăderi efectuate). Evaluarea condiției din ciclul **WHILE** necesită precalcularea primului rest şi contabilizarea acestei prime operații de scădere, prin inițializarea cu valoarea 1 a contorului **cat**. În continuare, se aplică o repetiție de tip **WHILE** (& 7.1) unei instrucțiuni compuse, în care se execută următoarele două instrucțiuni: aplicarea regulii de scădere: **rest**-ul curent este egal cu **rest**-ul anterior din care se scade **N2** şi "*numărarea*" scăderilor efectuate, adică creşterea cu 1 a variabilei **cat** la fiecare scădere efectuată. Condiția continuării ciclului **WHILE** este "**rest >= N2"**, altfel spus cât timp această condiție este îndeplinită se continuă procedeul scăderilor şi contorizarea lor. Din forma condiției se observă că această continuare se aplică inclusiv la egalitatea variabilei **rest** cu **N2**.

Variabila **cat** semnifică din punct de vedere matematic un rest al unei împărțiri, dar în program valoarea acesteia rezultă dintr-o operație de contorizare (& 2.5.12), impusă de logica algoritmului.

În final se afişează valorile variabilelor **cat** şi **rest**, calculate prin scădere comparativ cu cele obținute prin procedeul de împărțire.

În ultima instrucțiune **WRITELN** se remarcă aplicarea operatorilor **MOD** (împărțire numere întregi şi trunchiere la parte întreagă) respectiv **DIV (**restul împărțirii a două numere întregi**)** (& 4.2.1, tabel 4.4). Desigur că întreg programul poate fi rezumat la această linie din punct de vedere al calculelor, însă algoritmul exemplificat îşi propune să evidențieze o asocierea unei structuri de tip ciclic **WHILE** (& 7.1) cu o operație de contorizare (& 2.5.12) şi în aceasta constă utilitatea lui. De altfel ciclările şi procedeul contorizării sunt des utilizate în domeniul programării, iar exemplele următoare vor confirma acest lucru.

În fig. 7.24c se prezintă un exemplu de execuție a programului pentru valoarea lui **N1** egală cu **N2**.

```
Introduceti Deimpartit si Impartitor
15
15
Prin scadere: Catul=    1  Restul=    0
Direct:       Catul=    1  Restul=    0
```

Fig. 7.24c

```
Introduceti Deimpartit si Impartitor
16 3
cat=1  rest=13
cat=2  rest=10
cat=3  rest=7
cat=4  rest=4
cat=5  rest=1
Prin scadere: Catul=    5  Restul=    1
Direct:       Catul=    5  Restul=    1
```

Fig. 7.24d

Fig. 7.24d evidențiază evoluția valorilor variabilelor **cat** respectiv **rest** în timpul execuției programului. Rezultă evident creşterea cu 1 a variabilei **cat** la fiecare pas (scădere efectuată). Instrucțiunile corespunzătoare acestor afişări nu sunt specificate în fig. 7.23.

7.4.12 Căutare numere "cubice"

E 7.12 Program TurboPascal pentru căutare numere "cubice"

Vom înțelege prin numere cubice, numerele întregi care satisfac proprietatea că sunt egale cu suma cuburilor cifrelor lor. Exemplu:

$$N = 153 = 1^3 + 5^3 + 3^3 = 1 + 125 + 27$$

Ne propunem căutarea numerelor cubice în intervalul 100 ÷ 499, aplicând următoarea logică: numerele **N** sunt compuse din următoarele cifre: **S** sute, **Z** unități şi **U** unități, ele fiind deci exprimabile prin relația: **N** = **S** x 100 + **Z** x 10 + **U**. Dar, din enunțul problemei, rezultă că cifra sutelor **S** poate varia între 1 şi maxim 4, iar cifra zecilor **Z** şi unităților **U** între 0 şi 9. Variind deci valorile cifrelor între aceste limite, vom genera succesiv numere, dar numai acelea care respectă regula de mai jos vor fi admise ca fiind numerele căutate, fig. 7.26:

$$N = S \times 100 + Z \times 10 + U = S^3 + Z^3 + U^3$$

Regula exprima necesitatea egalității numerice între numărul calculat în format zecimal cu numărul corespondent rezultat din suma cuburilor cifrelor primului număr.

```
PROGRAM veri_cub;
{Numere intregi cuprinse intre 100-499
egale cu suma cuburilor cifrelor. Exemplu:
153=1*1*1+5*5*5+3*3*3}
VAR S,Z,U,N: INTEGER;
BEGIN
FOR S:=1 TO 4 Do
  FOR Z:=0 TO 9 DO
    FOR U:=0 TO 9 DO
      BEGIN
        N:=S*100+Z*10+U;
        IF N=S*S*S+Z*Z*Z+U*U*U THEN WRITELN(N);
      END;
READLN
END.
```

IC

Fig. 7.25

```
153
370
371
407
```

Fig. 7.26

Programul are la bază o idee simplă: se generează toate numerele cuprinse între 100 ÷ 499, prin intermediul a 3 cicluri incluse unul în altul: primul ciclu generează valorile cifrei sutelor, al doilea al cifrei unităților, iar ultimul al zecilor, apoi se calculează numărul în format zecimal şi se compară (în sensul egalității) cu valoarea expresiei formate din suma cuburilor cifrelor generate prin contorii de ciclu. La îndeplinirea condiției asociate instrucțiunii **IF** se afişează numărul, deoarece respectă condiția de număr "*cubic*".

Programul este un exemplu de aplicare a ciclurilor îmbricate (incluse unul în altul) (& 7.3), la care regula de funcționare este următoarea: ciclurile interioare se execută mai repede decât cele exterioare. Deci, pentru acest exemplu, cel mai repede se execută ciclul unităților, cel mai greu se execută ciclul sutelor, între ele fiind poziționat, din punct de vedere al execuției, ciclul zecilor. Oricare din ciclurile de tip interior se parcurg complet, înainte de a lăsa controlul ciclului de nivel superior.

7.4.13 Integrare numerică prin metoda dreptunghiurilor

E 7.13 Program TurboPascal pentru integrare numerică prin metoda dreptunghiurilor

Se considera funcția parabolă: $\mathbf{Y=X^2}$. Aria de sub curba $\mathbf{Y=X^2}$, cuprinsă între limitele **X=a** respectiv **X=b**, fig. 7.27, se calculează in mod exact prin integrala:

$$Aria = \int_a^b y \cdot dx = \int_a^b X^2 \cdot dx = \frac{X^3}{3} \Big|_b^a = \frac{b^3 - a^3}{3}$$

Calculul aproximativ al integralei se poate face prin integrare numerica prin metoda dreptunghiurilor, aproximând aria cu suma ariilor dreptunghiurilor $\mathbf{AD_i=l\ x\ Y_i}$, de lățime $\mathbf{l=(b-a)/N}$ şi înălțime $\mathbf{Y_i=X_i^2}$, unde **N** – este numărul impus de dreptunghiuri. Abscisele $\mathbf{X_i}$ din expresia înălțimii $\mathbf{Y_i}$ se calculează astfel:

$$\mathbf{X_1=a+l/2 \ ; \ X_2=a+l+l/2 \ ; \ X_3=a+2*l+l/2 \ ; \ X_4=a+3*l+l/2 \ ,}$$

deci pentru cazul general "**i**" : $\mathbf{X_i=a+(i-1)*l+l/2}$.

Datele de intrare sunt deci: numarul de intervale (dreptunghiuri de aproximare) **N** şi limitele de integrare **a**, **b**, iar rezultatul acestui program este aria calculată prin aproximare, comparată cu cea obținută prin integrare exactă (pentru funcția **parabolă**).

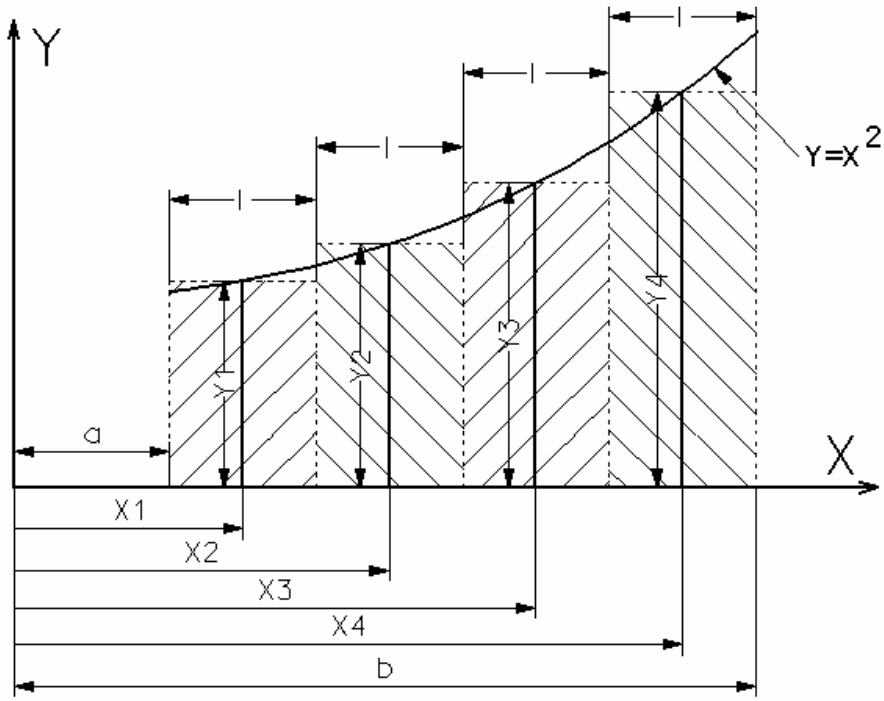


Fig. 7.27

Intervalul **[a, b]** este divizat în **N** "fâşii" de lățime constantă "**l**" şi înălțime variabilă "$\mathbf{y_i}$", calculată în mijlocul lățimii fâşiei.

Aria obținută prin însumarea fâşiilor elementare reprezintă o aproximare a ariei reale de sub curbă; este de aşteptat ca precizia aproximării să crească odată cu creşterea numărului de fâşii elementare. Procedeul poate fi aplicat pentru funcții a căror integrare analitică nu poate fi aplicată din cauză complexității lor.

```
PROGRAM integral;
{Calcul arie prin aproximare cu dreptunghiuri}
VAR aria,a,b,l,xi,yi,adi: REAL;
   N,i: INTEGER;
BEGIN
WRITELN('Numarul de intervale N');
READLN(N);
WRITELN('Limitele de integrare a,b');
READLN(a,b);
l:=(b-a)/N;
aria:=0; {Initializare variabila suma}
FOR i:=1 TO N DO
   BEGIN
      xi:=a+(i-1)*l+l/2; {Calcul abscisa}
      yi:=xi*xi; {Calcul ordonata}
      adi:=l*yi; {Arie dreptunghi elementar}
      aria:=aria+adi; {Sumare arii elementare}
   END;
WRITELN('Aria aprox. =',aria:12:4);
WRITELN('Aria exacta =',(b*b*b-a*a*a)/3:12:4);
READLN
END.
```

Fig. 7.28

IC

```
Numarul de intervale N
50
Limitele de integrare a,b
1 3
Aria aprox. =      8.6664
Aria exacta =      8.6667
```

Fig. 7.29

Programul exemplifică în TurboPascal o aplicație matematică, reducând calculul integral la o sumare (& 2.5.9). Variante ale execuției programului, pentru diferite valori ale limitelor de integrare **a**, **b** respectiv numere de intervale de divizare **N** sunt date în tabelul 7.1, din care rezultă creşterea preciziei odată cu creşterea lui **N**.

Tabel 7.1

A	b	Arie exacta	N	Arie aprox.
1	3	8,6667	3	8,5926
			10	8,66
			50	8,6664
			100	8,6666
1	50	41.666,3333	3	40.576,907
			10	41.568,295
			50	41.662,4117
			100	41.665,3529

Variabila **aria** este variabila care colectează suma ariilor elementare **adi** într-un ciclu de tip **FOR**, care are ca instrucțiune executabilă o instrucțiune de compusă (& 5.3), în interiorul căreia sunt calculate succesiv parametrii fiecărei fâşii elementare. În final se afişează aria calculată prin sumare comparativ cu cea exactă, calculată prin formula rezultată din integrarea analitică.

Desigur că programul este particularizat pentru funcția parabolă $\mathbf{Y=X^2}$, dar prin introducerea expresiei unei noi funcții în linia de program:

yi:=xi*xi; {Calcul ordonata}

programul poate fi aplicat şi altor funcții.

Există şi alte metode de integrare numerică, a căror eficiență este mai bună decât a acesteia.

7.4.14 Media aritmetică unui şir de numere

E 7.14 Program TurboPascal pentru calcul medie aritmetică unui şir de numere

De la tastatură se introduce o secvență de numere, secvență finalizată prin introducerea numărului 999. Se cere să se calculeze media aritmetică a secvenței de numere, media care să nu includă numărul final 999.

Media aritmetică a unui şir de numere se calculează prin raportul între suma lor şi numărul de numere al secvenței. Numărul de numere nu este fixat apriori, ci este determinat prin introducerea de la tastatură a numărului 999, motiv pentru care se necesită contorizarea fiecărui număr introdus, mai puțin a ultimului număr din secvență.

```
PROGRAM medie_999;
{Media numerelor citite pana la 999}
VAR x,suma,media: REAL;
   contor: INTEGER;
BEGIN
  contor:=0; {Initializare contor}
  suma:=0; {Initializare suma}
  WHILE x<>999 DO
    BEGIN
      WRITELN('Introduceti numar-Sfarsit=999');
      READLN(x);
      IF x<>999 THEN
        BEGIN
          suma:=suma+x; {Sumare numar}
          contor:=contor+1; {Contorizare numar}
        END;
    END;
  IF contor > 0 THEN
    BEGIN
      media:=suma/contor; {Calcul medie}
      WRITELN('Media=',media:12:3); {Afisare medie}
    END
  ELSE
    WRITELN('Nu s-a introdus nici un numar');
  READLN
END.
```

Fig. 7.30

```
Introduceti numar-Sfarsit=999
10
Introduceti numar-Sfarsit=999
20
Introduceti numar-Sfarsit=999
30
Introduceti numar-Sfarsit=999
40
Introduceti numar-Sfarsit=999
999
Media=      25.000
```

Fig. 7.31a

Fig. 7.31b

```
Introduceti numar-Sfarsit=999
999
Nu s-a introdus nici un numar
```

Variabilele **suma** şi **contor** au rolul funcțional de sumare (& 2.5.9) respectiv contorizare (& 2.5.12) a numerelor introduse, mai puțin a ultimului, motiv pentru care aceste operații s-au efectuat numai dacă **x<>999**, conform condiției din instrucțiunea **IF**.

Introducerea numerelor, sumarea şi contorizarea se execută într-un ciclu **WHILE** (& 7.1). Calculul şi afişarea mediei se execută numai dacă s-au introdus numere (instrucțiunea **IF** contor > 0 **THEN**), în caz contrar (ramura **ELSE**) se afişează numai un mesaj de avertizare.

7.5 PROCEDURI ASOCIATE CICLURILOR

Procedurile **BREAK** şi **CONTINUE** au fost introduse în versiunea 7.0 a limbajului şi pot apare în interiorul ciclurilor de tip **FOR**, **WHILE**, **REPEAT**, fiind necesare uneori pentru a simplifica codul programului. Se pot utiliza numai în interiorul ciclurilor, utilizarea în afara lor fiind semnalizată de către compilator cu eroare sintactică, fig. 7.32. Se recomandă evitarea acestor instrucțiuni şi utilizarea lor numai în cazurile de strictă necesitate.

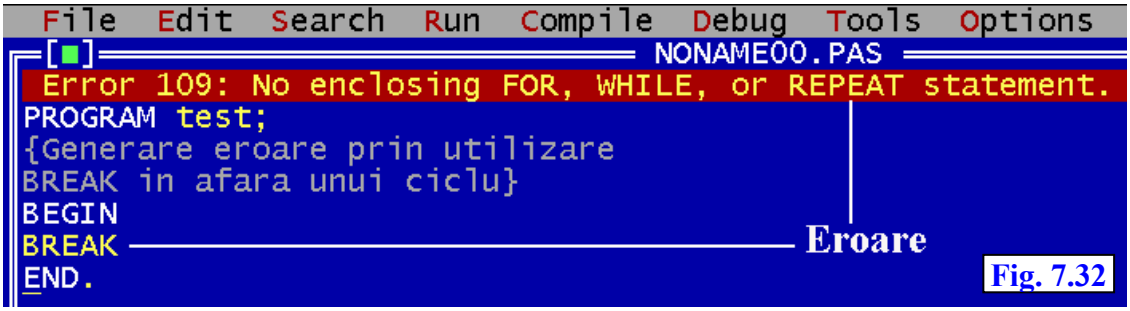


Fig. 7.32

7.5.1 Procedura BREAK

Procedura **BREAK** întrerupe necondiționat ciclurile de tip **FOR**, **WHILE**, **REPEAT** şi determină ieşirea din cel mai apropiat ciclu în care este inclusă, prin transferul execuției programului la prima instrucțiune de după ciclul în care este inclusă. Deoarece ieşirea din cicluri este stabilită prin condiția asociată ciclului, se recomandă utilizarea acestei instrucțiuni numai în situații speciale: execuția unei instrucțiuni care ar genera o eroare matematică (împărțire cu zero, radical din număr negativ, etc.).

E 7.15 Program TurboPascal pentru sumare numere cu ieşire forțată din ciclu prin BREAK

```
VAR x, suma:INTEGER;
BEGIN
suma:=0; {Initializare suma}
WRITELN('Introduceti numere-Exit=0');
WHILE TRUE DO
  BEGIN
  READLN(X);
  IF x=0 THEN BREAK;
  suma := suma+x;   {Sumare numar}
  END;
  WRITELN('Suma finala=',suma: 6);
  READLN
END.
```

IC

```
Introduceti numere-Exit=0
2
4
5
7
0
Suma finala=    18
```

Fig. 7.34

Fig. 7.33

Ciclul **WHILE** este infinit, deoarece nu există nici o posibilitate de a modifica condiția ciclului (valoarea **TRUE** nefiind modificabilă), iar ieşirea din ciclu se produce numai prin procedura **BREAK**, la introducerea valorii 0 de la tastatură. Desigur că exemplul este didactic, existând posibilitatea optimizării programului prin evitarea instrucțiunii **BREAK**.

7.5.2 Procedura CONTINUE

Procedura **CONTINUE** continuă ciclurile de tip **FOR**, **WHILE**, **REPEAT** şi determină trecerea la următoarea repetiție a ciclului în care este inclusă, instrucțiunile care urmează după procedura **CONTINUE** nemaifiind executate la repetiția curentă.

E 7.16 Program TurboPascal pentru afişare numere impare intre 1 şi 10

```
PROGRAM impare;
{Numere impare intre 1 si 10}
VAR i:INTEGER;
BEGIN
WRITELN ('Numere impare intre 1-10');
FOR i:=1 TO 10 DO
  IF ODD(i) THEN
    WRITELN(i)
  ELSE
    BEGIN
      CONTINUE;
      WRITELN('Numere pare')
    END;
READLN
END.
```

IC

Fig. 7.35

```
Numere impare intre 1-10
1
3
5
7
9
```

Fig. 7.36

?

Programul afişează numerele impare cuprinse între 1 şi 10 prin generarea tuturor numerelor (pare şi impare) din acest domeniu prin intermediul valorilor contorului de ciclu "**i**"; funcția **ODD**(i) asociată instrucțiunii **IF** testează dacă numărul este impar, situație în care acesta este afişat, iar în cazul contrar, ciclul se continuă prin procedura **CONTINUE** fără a se executa altceva; se observă inexistența mesajului **'Numere pare'**, care nu este afişat deoarece instrucțiunile următoare instrucțiunii **CONTINUE** nu mai sunt executate.

Exemplul este didactic, execuția programului oferă aceleaşi rezultate şi dacă se înlătură ramura **ELSE** a instrucțiunii **IF**.

Deşi limbajul TurboPascal oferă instrucțiunile **BREAK** respectiv **CONTINUE**, se recomandă utilizarea acestora numai în situațiile în care nu este posibilă elaborarea unui cod mai simplu în absența lor. În general există posibilități de optimizare a programului care permit evitarea utilizării acestor instrucțiuni.

8. INSTRUCȚIUNEA "GOTO". GENERAREA DE NUMERE ALEATOARE

8.1 INSTRUCȚIUNEA DE SALT GOTO

Instrucțiunea **GOTO** transferă execuția programului în mod necondiționat la o instrucțiune căreia i se atașează o *eticheta*. Deci instrucțiunea **GOTO** lucrează întotdeauna "*în tandem*" cu o etichetă, ambele fiind plasate în același bloc. Sintaxa acestei instrucțiuni este prezentată în fig. 8.1a, iar instrucțiunea la care se transferă controlul execuției programului este marcată prin prefixare cu o *etichetă* despărțită de instrucțiune prin simbolul "**:**", sub forma din fig. 8.1b:

GOTO *etichetă*

Fig. 8.1a

etichetă: instrucțiune

Fig. 8.1b

Eticheta trebuie declarată printr-o declarație de etichetă (declarație **LABEL**), secțiunea etichetelor fiind inclusă în secțiunea declarativă a programului (& 4.4). Etichetele pot fi de tip numeric cu maxim 4 cifre (ex. 10, 999, 1045) sau de tip identificator (A, ETI1, INTRARE2).

Restricții referitoare la etichete:

- orice etichetă trebuie declarată în secțiunea **LABEL**, anterior apariției ei în bloc căreia îi aparține;
- orice etichetă trebuie să prefixeze o singură instrucțiune;
- în cadrul aceluiași bloc etichetele trebuie să fie distincte.

Restricții privind plasarea instrucțiunilor **GOTO**:

- instrucțiunea **GOTO** nu trebuie să forțeze saltul în interiorul unei instrucțiuni structurate: compusă, **IF**, **CASE**, **WHILE**, **REPEAT**, **FOR**, **WITH**;
- instrucțiunea **GOTO** nu trebuie să forțeze saltul din exteriorul unui bloc în interiorul unui bloc;
- într-o instrucțiune condițională se interzice saltul dintr-o ramură în cealaltă.

Utilizarea instrucțiunii **GOTO** reprezintă o încălcare a principiului programării structurate (& 2.4) de aceea se recomandă evitarea utilizării ei. În practică pot exista însă situații care pot reclama utilizarea acestei instrucțiuni, pentru a genera un cod mai favorabil.

E 8.1 Program TurboPascal contorizare numere pozitive, negative şi nule din N numere - varianta programării cu instrucțiunea GOTO

Pentru comparație cu aplicația & 7.4.8, exemplul 7.8 va fi reluat exemplul contorizării numerelor pozitive, negative şi nule dintr-un şir de **N** numere, utilizând în programare instrucțiunea **GOTO**. Codul programului este prezentat în fig. 8.2.

În secțiunea declarațiilor programului (& 4.4), în secțiunea **LABEL** se declară trei etichete, dintre care două sunt de tip identificatori: **lpoz**, respectiv **lneg**, iar una de tip numeric: **100**.

```
PROGRAM cont_sir;
{Contorizare numere intregi
pozitive/negative/nule din sir
Utilizare instructiunea GOTO}
LABEL lpoz, lneg, 100;
VAR i,N,x,poz,nule,neg:INTEGER;
BEGIN
   WRITE('Cate numere ? '); READLN(N);
   poz:=0; {Initializare contor poz}
   nule:=0; {Initializare contor nule}
   neg:=0; {Initializare contor neg}
   FOR i:=1 TO N DO
     BEGIN
       WRITE('X',i,'=');READLN(x);
       IF X>0 THEN GOTO lpoz;
       IF X<0 THEN GOTO lneg;
       nule:=nule+1;
       GOTO 100;
lpoz:  poz:=poz+1;
       GOTO 100;
lneg:  neg:=neg+1;
100:   CONTINUE
     END;
   WRITELN('Numere pozitive=',poz);
   WRITELN('Numere nule=',nule);
   WRITELN('Numere negative=',neg);
   READLN
END.
```

IC

Fig. 8.2

În interiorul ciclului **FOR** se execută instrucțiunea compusă (& 5.3), evidențiată prin încadrare între cuvintele cheie **BEGIN – END**, în care se citesc şi se analizează numerele introduse de la tastatură, memorate succesiv în variabila **X**.

```
Cate numere ? 5
X1=1
X2=-2
X3=0
X4=3
X5=-4
Numere pozitive=2
Numere nule=1
Numere negative=2
```

Fig. 8.3

Programul se ramifică în 3 direcții:

- dacă numărul **X** este pozitiv, se transferă execuția programului la instrucțiunea prefixată de eticheta **lpoz**, unde se măreşte cu **1** contorul numerelor pozitive **poz**, după care execuția se transferă la eticheta prefixată de eticheta **100**;
- dacă numărul **X** este negativ, se transferă execuția programului la instrucțiunea prefixată de eticheta **lneg**, unde se măreşte cu **1** contorul numerelor negative **neg**, după care execuția se transferă la eticheta prefixată de eticheta **100**;
- dacă nici una din cele două variante anterioare nu se parcurge, se măreşte cu **1** contorul numerelor nule **nule**, după care execuția se transferă la eticheta prefixată de eticheta **100**.

Desigur că aceste salturi îngreunează înțelegerea algoritmului asociat programului, mai ales pentru programe complexe care conțin multe etichete.

Instrucțiunea **CONTINUE** permite continuarea ciclului, prin execuția următoarei repetiții a ciclului (& 7.5.2).

În final programul afişează valorile celor trei contori.

Libertatea alegerii tipurilor de etichete (numeric sau sub formă de identificatori) aparține programatorului, nefiind impusă de limbajul TurboPascal.

Declararea suplimentară, în secțiunea **LABEL**, a unei etichete neutilizate în program nu este semnalizată de compilator ca mesaj de atenționare sau mesaj de eroare.

8.2 GENERAREA DE NUMERE ALEATOARE

Limbajul Turbo Pascal oferă posibilitatea de generare a numerelor aleatoare, prin intermediul funcției **RANDOM**:

- pentru a genera numere aleatoare cuprinse în intervalul [0,1], se apelează funcția **RANDOM** fără nici un argument
- pentru a genera numere aleatoare cuprinse în intervalul [0 , N-1], se apelează funcția sub forma **RANDOM(N)**;
- pentru a genera numere aleatoare cuprinse în intervalul [M , N], unde M<N, se apelează funcția sub forma **M+RANDOM(M-N+1).**

Argumentele funcției **RANDOM** trebuie să fie de tip **INTEGER-WORD** (cuprinse deci în intervalul 0..65535) (& 4.2.1), în caz contrar compilatorul semnalează eroare **Error 76: Constant out of range.**

Pentru a reinițializa numărul de start al secvenței de numere aleatoare generate la apelul funcției **RANDOM** se utilizează procedura **RANDOMIZE**. Inițializarea se face cu o valoare obținută din ceasul sistemului. Neapelarea procedurii **RANDOMIZE** va avea ca efect generarea aceloraşi numere la fiecare execuție a programului.

E 8.2 Program TurboPascal pentru contorizare apariție număr fețe la aruncarea unui zar

Ne propunem contorizarea aparițiilor fețelor unui zar, adică afişarea, pentru fiecare față, a numărului de apariții dintr-un total de N aruncări ale zarului. Pentru aceasta vom utiliza funcția **RANDOM**, care va simula aruncarea zarului prin generarea de numere aleatoare întregi între 1 şi 6, echivalând generarea aleatoare a unui număr cu aruncarea feței corespondente.

Programul sursă este prezentat în fig. 8.4, iar execuția sa în fig. 8.5.

Variabila **fata** este rezervată pentru memorarea numărului aleator generat prin functia **RANDOM**. Variabilele **cf1**, **cf2**, **cf3**, **cf4**, **cf5**, **cf6**, sunt contori cu rolul de a memora numărul de aruncări pentru fiecare față din cele 6 posibile. Astfel, la generarea aleatoare a unui număr în variabila **fata**, va creşte cu o unitate variabila contor alocată numărului feței, aplicând astfel algoritmul de contorizare (& 2.5.12).

Aruncările multiple sunt programate printr-un ciclu de tip **FOR** (& 7.3), iar instrucțiunile repetitive se încadrează într-o instrucțiune compusă (& 5.3).

Se remarcă utilizarea instrucțiunii de selecție multiplă **CASE** pentru a selecta contorul feței care trebuie mărit, în raport cu aruncarea curentă (& 6.2).

În final sunt afişate valorile contorilor fețelor, a căror sumă trebuie să coincidă cu numărul total de aruncări **N**.

```
PROGRAM zaruri;
{Contorizare fete zar din N aruncari}
VAR N, fata, cf1, cf2, cf3, cf4, cf5, cf6, i:INTEGER;
BEGIN
RANDOMIZE; {Initializare generator numere aleatoare}
WRITE('Numarul de aruncari ?'); READLN(N);
{Initializare contori fete zar}
cf1:=0;cf2:=0;cf3:=0;cf4:=0;cf5:=0;cf6:=0;
FOR i:=1 TO N DO
   BEGIN
      fata:=1+random(6);
      WRITELN('Aruncarea=',i,'  fata=',fata);
IC    CASE fata OF
         1: cf1:=cf1+1;
         2: cf2:=cf2+1;
         3: cf3:=cf3+1;
         4: cf4:=cf4+1;
         5: cf5:=cf5+1;
         6: cf6:=cf6+1;
      END;
   END;
WRITELN('Contor fata 1 =',cf1);
WRITELN('Contor fata 2 =',cf2);
WRITELN('Contor fata 3 =',cf3);
WRITELN('Contor fata 4 =',cf4);
WRITELN('Contor fata 5 =',cf5);
WRITELN('Contor fata 6 =',cf6);
READLN
END.
```

Fig. 8.4

```
Numarul de aruncari ?5
Aruncarea=1  fata=1
Aruncarea=2  fata=4
Aruncarea=3  fata=4
Aruncarea=4  fata=2
Aruncarea=5  fata=1
Contor fata 1 =2
Contor fata 2 =1
Contor fata 3 =0
Contor fata 4 =2
Contor fata 5 =0
Contor fata 6 =0
```

Fig. 8.5

9. TIPURI DE DATE DEFINITE DE UTILIZATOR

Limbajul TurboPascal are predefinite tipurile standard de date: **INTEGER, BOOLEAN**, **REAL**, **CHAR (**& 4.2**)**. Aceasta nu constituie însă o restricţie referitoare la tipurile de date utilizabile în TurboPascal, dimpotrivă limbajul permite şi crearea de către utilizator a noi tipuri de date, care pot fi identificate prin nume alese de către programator.

Definiţiile acestor tipuri de date trebuie plasate în zona **TYPE** a secţiunii de declaraţii a programului **(**& 4.4**)**, în blocul în care va fi utilizat.

Sintaxa generală a unei instrucţiuni de definire a unui tip de date utilizator este:

```
TYPE Nume_tip = definitie_tip
```

unde **nume_tip** este numele tipului de date, iar **definitie_tip** reprezintă definiţia asociată tipului de date, definiţia tipului de date poate fi diversă funcţie de necesităţile impuse de program.

Deoarece în definiţia tipurilor de date utilizator pot interveni constante, iar numele tipurilor de date utilizator intervin în definirea variabilelor programului, secţiunea declaraţiilor de tip **TYPE** trebuie plasată între secţiunea declaraţiilor constantelor **CONST** şi secţiunea declaraţiilor variabilelor **VAR (**& 4.4**)**.

9.1 TIPURI SCALARE

Tipurile scalare sunt formate din tipurile **enumerare** şi tipul **interval**.

9.1.1 Tipul enumerare

Tipul enumerare este un tip de date ordinal **(**& 4.2**)** care defineşte o mulţime ordonată de valori şi este descris prin intermediul unei liste finite de constante, listă încadrată de paranteze rotunde, conform următoarei sintaxe:

```
TYPE Nume_enumerare = (const0, const1, const2, ……)
```

Constantele enumerării **const0, const1, const2, ……** nu trebuie declarate în secţiunea **CONST**, deoarece aparţin enumerării.

În cadrul acestei liste, fiecare element are asociat un număr de ordine, prin care se defineşte poziţia constantei în cadrul şirului. Astfel, primul element are numărul de ordine 0, următorul are numărul de ordine 1, etc. În listă constantele se definesc în ordinea crescătoare a numărului de ordine. Prin aceasta se defineşte o relaţie de ordine, în care poziţia fiecărei constante din enumerare este precizată în raport cu constantele vecine. De aceea operatorii relaţionali = , <> , < , > , <= , >= **(**& 4.2.2, tabel 4.5**)** pot fi utilizaţi pentru compararea a două constante ce aparţin aceleiaşi enumerări.

Constantele dintr-o enumerare nu pot aparţine altei enumerări, compilatorul semnalizând eroare, fig. 9.1.

Fig. 9.1

Variabilele de tip enumerare sunt declarate în secțiunea **VAR**. Ele pot lua una din valorile înșirate în lista enumerării din secțiunea **TYPE**, fig. 9.2a. Exemple:

```
TYPE  comutator=(deschis, inchis);
      culori=(alb, rosu, verde);
VAR   com: comutator;
      cul: culori;
```

Fig. 9.2a

Variabila **com** poate lua una din valorile *deschis* sau *inchis*, variabila **cul** poate lua una din valorile *alb*, *rosu* sau *verde*.

Este corectă şi varianta definirii enumerării direct în secțiunea **VAR**, fără a fi predefinit tipul în secțiunea **TYPE**, caz în care se specifică atât numele variabilei enumerare cât şi lista de constante asociate, fig. 9.2b.

Fig. 9.2b

```
VAR enumerare_test : (alfa, beta, gama)
```

Prima variantă este utilă când se impune definirea mai multor variabile enumerare de acelaşi tip, iar a doua se poate utiliza atunci când variabila enumerare este unică.

În exemplele prezentate, s-au definit următoarele tipuri de date enumerare:

- **comutator** defineşte o enumerare compusă din două poziții, **deschis** cu numărul de ordine **0** respectiv **închis** cu numărul de ordine **1**.
- **culori** defineşte o enumerare compusă din trei poziții, **alb** cu numărul de ordine **0, roşu** cu numărul de ordine **1** respectiv **verde** cu numărul de ordine **2**;

precum şi următoarele variabile:

- **com** variabilă de tip **comutator**;
- **cul** variabilă de tip **culori**;
- **enumerare_test** variabilă care defineşte o enumerare compusă din trei poziții, **alfa** cu numărul de ordine **0, beta** cu numărul de ordine **1** respectiv **gama** cu numărul de ordine **2**.

Operațiile care se pot face cu valorile unui tip enumerare sunt:

- **atribuire**

  exemplu com := *deschis*;
  cul := *alb*;
  enumerare_test := *alfa*;

- **determinarea numărului de ordine** cu funcția **ORD :**

  exemplu **ORD**(*deschis*) returnează valoarea 0
  ORD(*rosu*) valoarea 1;

- **determinarea succesorului/predecesorului** cu funcţiile **SUCC**( ) / **PRED**();
  exemplu prin atribuirea **cul**:=**SUCC**(*rosu*), **cul** va primi valoarea *verde*;
  prin atribuirea **com**:=**PRED**(*inchis*), **com** va primi valoarea *deschis*;

OBS. încercarea determinării succesorului ultimului element sau predecesorului primului element din listă genereaza eroare **Error 76: Constant out of range.**

- **comparaţia** : două valori V1 şi V2 sunt într-o relaţie, dacă **ORD**(V1) şi **ORD**(V2) sunt în aceeaşi relaţie;

exemplu: comparaţia "*alb*<*verde*" este adevărată deoarece **ORD**(*alb*)=0 este mai mic decât **ORD**(*verde*)=2.

O variabilă enumerare nu poate fi folosită într-o operaţie de intrare – ieşire (**READ-WRITE**), rezultând eroarea: **Error 64: Cannot Read or Write variables of this type**.

9.1.2 Tipul interval (subdomeniu)

Tipul de date interval este un tip definit de utilizator, în baza unui tip scalar numit *scalar asociat*. Tipul **interval** (**subdomeniu**) are aceleaşi proprietăţi ca şi tipul scalar asociat şi este o submulţime de valori ale tipurilor de date ordinale **(**& 4.2**)**. Definiţia unui interval indică valorile constante cea mai mică respectiv mare din interval (în sensul numărului de ordine), despărţite prin separatorul "**..**" şi cuprinde toate valorile dintre ele. Sintaxa unui tip interval este:

TYPE nume_tip_interval = valoare minimă .. valoare maximă

Restricţii:

- Valoarea minimă trebuie să fie mai mică decât cea maximă;
- Nu este permisă definirea unui interval al tipului real, deoarece acesta nu este un tip ordinal **(**& 4.2**)**;
- Limitele domeniului pot fi specificate prin expresii constante; apare o problemă de sintaxă atunci când expresia începe cu paranteza "**(",** deoarece aceasta este specifică tipului enumerare; de exemplu declaraţia **TYPE subdom = (a-1)*2 .. 5** se poate înlocui cu declaraţia **TYPE subdom = 2*(a-1) .. 5**, ceea ce va elimina ambiguitatea;
- Rezultatele operaţiilor efectuate asupra datelor de tip interval nu trebuie să genereze valori în afara intervalului.

Exemple:

TYPE luni = (ian, feb, mar, apr, mai) {enumerare}
TYPE luni_iarna = (ian .. mar) {interval al enumerării}
TYPE cifre = (0 .. 9) {interval al tipului **INTEGER**}
TYPE litere_mici = 'a' .. 'z' {interval al tipului **CHAR**}

Tipurile interval sunt frecvent utilizate ca tipuri pentru indici de tablouri (**ARRAY**) sau ca tip de bază pentru tipurile mulţime (**SET**), & 9.2.

9.2 TIPURI STRUCTURATE DE DATE

Tipuri de date simple definesc o unică informaţie prin intermediul unui identificator. Tipurile de date studiate până în acest moment au fost tipuri de date **simple**, care includ următoarele categorii de tipuri de date:

- tipuri standard: **INTEGER, BOOLEAN**, **REAL**, **CHAR** (& 4.2);
- tipuri scalare: **enumerare**, **interval** (& 9.1).

Tipurile structurate de date se definesc cu ajutorul tipurilor de date simple şi grupează, sub acelaşi identificator un grup de componente, în care fiecare componentă poate fi accesată individual. Printr-un tip structurat se defineşte un tip de date, specificat prin următoarele caracteristici:

- tipul de date al componentelor;
- tehnica de structurare - metoda de organizarea a componentelor în cadrul structurii;
- metoda de accesare - mecanismul de regăsire a componentelor în cadrul structurii;

după acest criteriu se disting următoarele tipuri de date structurate:

- **array (tablou)**;
- **string** (**şir de caractere**);
- **set (mulţime)**;
- **record (înregistrare)**
- **file (fişier)**.

Necesarul de memorie pentru tipurile structurate de date rezultă din tipul şi numărul componentelor şi rămâne constant pe parcursul execuţiei programului, de unde rezultă şi denumirea de **structuri statice**.

9.2.1 Tipul tablou (array)

Tabloul furnizează un mijloc de a grupa sub acelaşi nume mai multe valori cu caracteristici identice.

Tabloul este o structură formată dintr-un număr fix de componente, toate de acelaşi tip, denumite *componente (elemente)*, identificate printr-un un unic identificator (numele variabilei tablou) şi memorate într-o zonă continuă de memorie.

Accesarea oricărui element al tabloului se face direct, prin intermediul numelui variabilei tablou, urmat de **indici** încadraţi între paranteze drepte, **indici** care precizează poziţia elementului în cadrul structurii. Componentele unei structuri tablou se mai numesc şi **variabile indexate**, deoarece selecţia fiecărei componente se face în mod obligatoriu prin intermediul *indici*-lor.

Funcţie de numărul de dimensiuni, tablourile pot clasificate după urmează:

- **tablouri lineare** (tablouri cu o singură dimensiune) – echivalente noţiunii de vector sau şir din matematică, motiv pentru care sunt apelate şi sub titulatura de **vectori**; tablourile lineare se definesc prin intermediul unui singur indice, care precizează poziţia fiecărei componente în cadrul tabloului şi permite accesarea acesteia;
- **tablouri bidimensionale** (tablouri cu două dimensiuni) – echivalente noţiunii de matrice din matematică, motiv pentru care sunt apelate şi sub titulatura de **matrici**; tablourile bidimensionale se definesc prin intermediul a doi indici, din care primul precizează linia, iar al doilea precizează coloana, poziţia fiecărei componente în cadrul tabloului fiind reperată la

intersecția liniei cu coloanei; structura este similară cu o grila rebus, unde accesarea fiecărei căsuțe se realizează la intersecția liniei cu a coloanei;

- **tablouri multidimensionale** (tablouri cu mai multe dimensiuni) – care nu sunt apelate sub o titulatură specială; tablourile multidimensionale se definesc prin intermediul unui număr de indici egal cu numărul de dimensiuni al tabloului; de exemplu pentru accesarea componentelor unui tablou **n**-dimensional se realizează prin intermediul a **n** indici; numărul de dimensiuni pentru un tablou este nelimitat de limbaj.

Ca **indice** se poate folosi orice expresie, care se va evalua în momentul accesării prin program a elementului, însă rezultatul expresiei trebuie să fie de acelaşi tip cu cel declarat în descrierea tabloului şi să se încadreze între valorile declarate ale indicelui, în caz contrar rezultă eroare la execuție.

Indicii se încadrează între paranteze drepte, parantezele fiind necesare din cauza imposibilității dispozitivelor de intrare de a lucra cu indici inferiori şi sunt asemănători în multe privințe cu indicii matematici (care se scriu în partea de jos a elementului fără paranteze).

Fig. 9.3

Şir matematic (vector) - X

X_1 , X_2 , X_3 , X_4 , X_5

Tablou linear (vector) - X

Indice →	1	2	3	4	5
Valoare →	V_1	V_2	V_3	V_4	V_5

Analogia dintre reprezentarea matematică şi informatică a unui vector este prezentată în fig. 9.3. Astfel X_i reprezintă valoarea elementului din poziția "**i**" a şirului **X**, care are **N=5** componente. În reprezentarea TurboPascal, tabloul linear echivalent **X** va conține 5 componente, memorate în 5 locații de memorii succesive, fiecărei componente fiindu-i alocat câte un indice şi o valoare. Accesarea valorii elementului din poziția "**i**" se face sub forma **X[i]**, unde **X** este identificatorul tabloului. Concret, accesarea valorii V_3 din poziția 3-a a tabloului se face sub forma **X[3]**.

Tablou bidimensional (matrice)-Y

Linia/Coloana	1	2	3
1	V_{11}	V_{12}	V_{13}
2	V_{21}	V_{22}	V_{23}
3	V_{31}	V_{32}	V_{33}
4	V_{41}	V_{42}	V_{43}

Indice de coloană

Indice de linie

Valori

Fig. 9.4

Reprezentarea TurboPascal a unui tabloul bidimensional cu 4 linii şi 3 coloane este prezentată în fig. 9.4. Accesarea valorii V_{32} din linia 3, coloana 2 se face sub forma **Y[3,2]**, unde **Y** este identificatorul tabloului. Numărul de componente a tabloului este dat de produsul dintre numărul total de linii şi numărul total de coloane, adică:

N = 4 x 3 = 12

componente. Deci pentru memorarea tabloului sunt necesare 12 locații de memorie.

Definirea unui tablou impune precizarea următoarelor caracteristici:

- **tipul componentelor**: poate fi admis oricare din tipurile de date ale limbajului TurboPascal (simple sau structurate, mai puțin tipul **fişier**);
- **tipul şi domeniile indicilor**: tipurile legale de indici sunt tipurile ordinale, exceptând tipurile **LONGINT** şi subintervale ale acestuia (& 4.2.1); domeniul indicilor este intervalul de valori ale acestora şi este specificat prin uzual tipul de date interval (& 9.1.2).

Sintaxa declarării tipurilor de date tablou este următoarea:

TYPE nume_tablou = **ARRAY [**tip_ind1, tip_ind2, tip_ind3, ….**] OF** tip_comp

unde:

- *nume_tablou* – reprezintă identificatorul asociat tabloului, prin care acesta va fi accesat;
- *tip_ind1, tip_ind2, tip_ind3, …*– sunt tipuri ordinale care definesc mulțimile valorilor pentru fiecare dimensiune a tabloului; numărul total de componente al tabloului este dat de produsul numărului de valori ale tipului fiecărui indice;
- *tip_comp* – reprezintă tipul componentelor tabloului.

Valorile unui tip de date sunt accesate prin intermediul variabilelor, ceea ce impune definirea variabilelor de tip tablou. Aceasta se poate realiza în două moduri:

a) declarația tipului tabloului în secțiunea **TYPE**, urmat de declarația variabilelor de tip tablou în secțiunea **VAR** a secțiunii declarative (& 4.4); această formă de declarare este utila la declararea mai multor variabile tablou de același tip;

b) declararea directă şi implicită a variabilei de tip tablou în secțiunea **VAR**, prin specificarea în totalitatea a caracteristicilor acestuia, fără a mai declara anterior tipul acestuia în secțiunea **TYPE**.

Desigur că numărul de tablouri este impus de necesitățile de programare a algoritmului.

Exemple de declarații de forma **a**:

Declararea tipului tabloului din fig. 9.3, presupunând ca valorile de memorat în tablou vor fi de tip real, se face prin următoarea declarație:

TYPE VECT_X=ARRAY [1 .. 5] OF REAL

urmând ca în secțiunea de declarații **VAR** să fie declarate două variabile tablou **X** respectiv **X1** de tipul **VECT_X**, sub forma:

VAR X, X1: VECT_X

care va avea ca efect crearea în memorie a două variabile de tip tablou cu aceeaşi structură, definită prin declarația **TYPE**, denumite **X** respectiv **X1**, în fiecare vor fi memorate câte 5 componente cu valori de tip real.

Pentru declararea tipului tabloului din fig. 9.4, presupunând ca valorile de memorat în tablou vor fi de tip întreg, se utilizează următoarea declarație:

TYPE MATR_Y=ARRAY [1 .. 4, 1 .. 3] OF INTEGER.

urmând ca în secțiunea de declarații **VAR** să fie declarate trei variabile tablou **Y, Y1** respectiv **Y2** de tipul **MATR_Y**, sub forma:

VAR Y, Y1, Y2 : MATR_Y;

care va avea ca efect crearea în memorie a trei variabile de tip tablou cu aceeaşi structură, definită prin declarația **TYPE**, denumite **Y, Y2** respectiv **Y3**, în fiecare vor fi memorate câte 12 componente cu valori de tip întreg.

Declarația

TYPE TABL = ARRAY [‘a’ .. ‘i’] OF REAL

defineşte un tip de vector de numere reale *numerotate* de la ‘a’ la ‘z’, iar declarația:

VAR SIR, SIR1: TABL;

va avea ca efect crearea în memorie a două variabile de tip tablou cu aceeaşi structură, definită prin declarația **TYPE**, denumite **SIR** respectiv **SIR1**, în fiecare vor fi memorate câte 9 componente cu valori de tip real; valorile din. fig. 9.5 sunt exemplificative.

SIR	Indice	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**
	Valoare	1.2	1.4	1.5	12.0	0.8	1.2	0.9	2.4	0.0

SIR1	Indice	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**
	Valoare	12.0	-1.6	22.1	-0.9	1.0	2.2	3.1	-1.1	89.0

Fig. 9.5

Accesarea valorilor se poate face conform sintaxei următoare:

- **SIR[‘b’] => 1.4**, reprezentând poziția **‘b’** (a doua) din vectorul **SIR**;
- **SIR1[‘d’] => -0.9**, reprezentând poziția **‘d’** (a patra) din vectorul **SIR1**.

Exemple de declarații de forma **b**:

Vom relua acelaşi exemple, declarând aceleaşi variabile direct în secțiunea **VAR**, fără predefinirea tipului tablourilor în secțiunea **TYPE**.

```
VAR   X, X1: ARRAY [1 .. 5] OF REAL;
      Y, Y1, Y2 : ARRAY [1 .. 4, 1 .. 3] OF INTEGER;
      SIR, SIR1: ARRAY [‘a’ .. ‘i’] OF REAL;
```

În cazul în care pentru indici se foloseşte tipul interval, este avantajos ca limitele să fie declarate în secțiunea **CONST** a secțiunii declarative (& 4.4), pentru a mări caracteristica de generalitate a programului.

Exemple

Vom relua acelaşi exemple, declarând anterior limitele în secțiunea **CONST**:

```
CONST       cx=5;  cl=4;  cc=3;
            lmin=’a’ ; lmax=’i’;
VAR   X, X1: ARRAY [1 .. CX] OF REAL;
      Y, Y1, Y2 : ARRAY [1 .. CL, 1 .. CC] OF INTEGER;
      SIR, SIR1: ARRAY [lmin .. lmax] OF REAL;
```

În toate instrucțiunile care lucrează cu valorile maximale ale tablourilor, în locul utilizării valorilor numerice **5**, **4**, **3**, **’a’, ’i’** se va lucra cu constantele asociate, adică **cx**, **cl**, **cc**, **lmin**, **lmax**, astfel încât modificarea numerică a valorilor limitelor va afecta în codul programului numai instrucțiunea **CONST**, restul instrucțiunilor rămânând nemodificate.

Se pot defini şi constante de tip tablou, care trebuie să includă în declarație valorile componentelor sale incluse între paranteze rotunde şi despărțite prin separatorul "**,**", conform sintaxei:

CONST nume_tablou **:** tip_tablou = **CONST (**const1, const2, const3, ….)

unde:

- *nume_tablou* – reprezintă identificatorul asociat tabloului de tip *tip_tablou*;
- *const1, const2, const3, …* **-** sunt valori sau expresii constante.

Exemplu

CONST vcx : ARRAY[1..5] OF REAL=(1.2, 7*8, 1, 1/3, 1.4);

Valorile componentelor tablourilor declarate prin secțiunea **CONST** pot fi modificate la execuția programului, declararea constituind o metoda simpla de inițializare a valorilor tabloului.

Alte metode de generare a valorilor componentelor unui tablou pot fi:

- citirea individuală a valorilor componentelor sale: instrucțiunea **READLN** (& 5.5.1);
- atribuirea explicită de valori individuale sau rezultate din evaluarea unor expresii prin intermediul instrucțiunii de atribuire (& 5.2);
- generarea de valori aleatoare prin funcții specifice limbajului TurboPascal.

O variabilă de tip tablou nu poate fi citită sau scrisă în întregime, ci în general se lucrează pe componente, aplicându-se aceleaşi operații care se aplică variabilelor simple.

Dacă însă sunt definite două tablouri prin acelaşi tip, **a** respectiv **b**, numele variabilelor de tip tablou pot apare într-o instrucțiune de atribuire (& 5.2), sub forma: **a := b,** care provoacă copierea tuturor valorilor componentelor tabloului din membrul drept în pozițiile corespondente din tabloul aflat în membrul stâng.

9.3 PRELUCRĂRI ASUPRA TABLOURILOR

În programare, tablourile reprezintă una din cele mai des utilizate structuri de date, datorită următoarelor avantaje:

- oferă posibilitatea manipulării unor date de acelaşi tip sub aceeaşi denumire;
- aplicarea unitară a operațiilor asupra unui grup de date şi nu asupra unor date individuale;
- posibilitatea utilizării structurilor repetitive (& 7).

În general un program care lucrează cu tablouri poate avea trei părți: citirea sau generarea valorilor componentelor, prelucrări efectuate asupra tabloului, afişarea componentelor tabloului, aceste etape nu se pot constitui însă ca o metodă unică şi generală.

Prelucrările efectuate asupra unui tablou se referă în general la prelucrări efectuate asupra valorii componentelor sale, fiecare componentă fiind accesată prin intermediul indicilor. În general, prelucrările se aplică tuturor componentelor sale, începând cu prima şi terminând cu ultima, ceea ce înseamnă că indicii tabloului trebuie să varieze în acest sens. Pentru a putea utiliza structura repetitivă, modalitatea tehnică utilizată în prelucrarea tablourilor este de a asocia indicii de tablouri cu indicii ciclurilor, prin utilizarea aceleiaşi variabile de memorie, în care să se genereze repetitiv valorile dorite ale indicilor tabloului.

Astfel, pentru dublarea valorilor componentelor vectorului **X** din fig. 9.3, se poate aplica operația de dublare prin intermediul a 5 instrucțiuni de atribuire.

```
X[1] := 2 * X[1]
X[2] := 2 * X[2]
X[3] := 2 * X[3]
X[4] := 2 * X[4]
X[5] := 2 * X[5]
```

Se observă caracterul repetitiv al acestor instrucțiuni, singurul parametru care variază fiind indicele tabloului. Din acest motiv, o modalitate mult mai practică de prelucrare este includerea relației de dublare într-un ciclu cu un contor, care, prin valorile parcurse, să descrie valorile indicelui de tablou. Prelucrarea ar putea fi descrisă sintetic prin operația: **X[i] := 2 * X[i],** unde "**i**" variază de la 1 la 5.

În acest exemplu se observă deci că variabila "**i**" are o funcție dublă:

- funcția de *contor de ciclu*, aşa cum a fost definit în paragraful (& 4.2) şi beneficiind de toate proprietățile ciclurilor;
- funcția de *indice de tablou*, prin intermediul căruia sunt accesate succesiv componentele sale în vederea aplicării operației dorite.

In limbaj TurboPascal prelucrarea poate fi descrisă prin instrucțiunea:

FOR i := 1 TO 5 DO X[i] := 2 * X[i]

Se observă diferența programării operației prin 5 instrucțiuni comparativ cu execuția în ciclu a operației, care consumă o singură instrucțiune; cu cât creşte numărul de componente a tabloului cu atât creşte şi eficiența programării prin această a doua tehnică în raport cu prima.

Prelucrările asupra tablourilor vor fi incluse în structuri repetitive, impunând accesarea componentele tablourilor prin indici a căror valori să fie generate prin intermediul contorilor de ciclu.

Dacă prelucrarea unui tablou linear (vector) presupune parcurgerea succesivă a componentelor sale, prelucrarea componentelor unui tablou bidimensional (matrice) se poate realiza în două variante:

- **pe linii**: se fixează prima linie şi se aplică prelucrările pentru toate coloanele primei linii, apoi se fixează a doua linie şi se aplică prelucrările pentru toate coloanele celei de-a doua linii, procedeul continuând până la epuizarea tuturor liniilor; cu alte cuvinte, indicele de coloană se consumă mai repede decât cel de linie, ceea ce, conform caracteristicilor ciclurilor (pct. 8, & 7.3) ne conduce la concluzia că indicele de coloană trebuie alocat ciclului interior, iar cel de linie ciclului exterior;
- **pe coloane**: se fixează prima coloană şi se aplică prelucrările pentru toate liniile primei coloane, apoi se fixează a doua coloană şi se aplică prelucrările pentru toate liniile celei de-a doua coloane, procedeul continuând până la epuizarea tuturor coloanelor; cu alte cuvinte, indicele de linie se consumă mai repede decât cel de coloană, ceea ce, conform caracteristicilor ciclurilor (pct. 8, & 7.3) ne conduce la concluzia că indicele de linie trebuie alocat ciclului interior, iar cel de coloană ciclului exterior.

Similar se aplică prelucrări asupra unui tablou cu mai multe dimensiuni.

Recomandările din acest paragraf au un caracter de generalitate, dar aplicarea lor depinde şi de algoritmul concret, care însă nu de puține ori impune şi prelucrarea individuală a componentelor tablourilor sau aplicarea prelucrărilor asupra unei părți a tabloului. Din acest motiv un algoritm general de prelucrare a tablourilor nu poate fi dezvoltat.

În cele ce urmează vor fi exemplificate prelucrări des utilizate asupra tablourilor.

9.3.1 Citirea componentelor unui vector şi afişarea valorilor sale

E 9.1 Program TurboPascal pentru citirea componentelor unui vector şi afişarea valorilor sale

Citirea de la tastatură a valorilor componentelor unui vector se face prin citirea fiecărei componente în parte, într-un ciclu **FOR** (& 7.3), al cărui număr de repetiţii este egal cu numărul de componente al vectorului.

```
PROGRAM vector1;
{Citire si afisare elemente vector}
CONST n=5;
TYPE vect=ARRAY[1..N] OF INTEGER;
VAR a:vect;
   i :INTEGER;
BEGIN
  {Citire elemente vector}
  FOR i:=1 TO N DO
    BEGIN
      WRITE('Elementul ',i:3,'=');
      READLN(a[i])
    END;
  {Afisare elemente vector pe acelasi rand,
   despartite de spatii}
  WRITELN('Afisare vector pe linie');
  FOR i:=1 TO N DO WRITE(a[i],' ');
  WRITELN; {Trecere cursor pe linia urmatoare }
  {Afisare fiecare element vector pe randuri diferite}
  WRITELN('Afisare vector pe coloana');
   FOR i:=1 TO N DO WRITELN(a[i]);
  READLN
END.
```

IC

Fig. 9.6

```
Elementul   1=12
Elementul   2=16
Elementul   3=0
Elementul   4=-19
Elementul   5=4
Afisare vector pe linie
12 16 0 -19 4
Afisare vector pe coloana
12
16
0
-19
4
```

Fig. 9.7

Se observă citirea de la tastatură a valorii fiecărei componente a vectorului, de la prima la ultima, prin încadrarea în ciclu **FOR** a instrucţiunii de citire **READLN**, precedată de o instrucţiune de afişare **WRITELN** cu rol de atenţionare a introducerii datei.

Primul mod de afişare a valorii componentelor (pe aceeaşi linie) impune utilizarea formei **WRITE** a instrucţiunii de afişare (& 5.5.2), pentru a conserva poziţia cursorului pe acelaşi rând după scrierea valorii fiecărei componente. Deoarece şi după afişarea ultimei componente cursorul ecran rămâne poziţionat pe acelaşi rând, se impune o instrucţiune **WRITELN** pentru a provoca trecerea cursorului ecran pe rândul următor. În sintaxa comenzii este inclusă afişarea caracterului spaţiu, prin încadrarea unui spaţiu între apostroafe, pentru a separa cifrele alocate valorilor componentelor.

Al doilea mod de afişare a valorii componentelor (pe linii diferite) impune utilizarea instrucţiunii **WRITELN**, care provoacă trecerea cursorului pe rândul următor la sfârşitul comenzilor de scriere incluse între parantezele asociate comenzii.

9.3.2 Afişarea în ordine inversă a valorilor unui vector generate aleator

E 9.2 Program TurboPascal pentru afişarea în ordine inversă a valorilor unui vector generate aleator

Programul afişează în ordine inversă valorile componentelor unui vector, generate aleator prin funcţia **RANDOM** (& 8.2), fără a realiza inversarea componentelor sale.

Fig. 9.8

```
PROGRAM vector2;
{Generare elemente vector prin numere
aleatoare cuprinse intre 5 si 15 si
afisare in ordine normala si inversa}
CONST n=5;
VAR a:ARRAY[1..N] OF INTEGER;
   i:INTEGER;
BEGIN
   {Initializare generator numere aleatoare}
   RANDOMIZE;
   {Generare valori aleatoare}
   FOR i:=1 TO N DO a[i]:=5+RANDOM(11);
   {Afisare formatata elemente vector pe acelasi
   rand in ordinea normala a elementelor }
   WRITELN('Vector in ordine normala');
   FOR i:=1 TO N DO WRITE(a[i]:4);
   WRITELN; {Trecere cursor pe randul urmator}
   {Afisare formatata elemente vector pe acelasi
   rand in ordinea inversa a elementelor }
   WRITELN('Vector in ordine inversa');
   FOR i:=N DOWNTO 1 DO WRITE(a[i]:4);
   WRITELN; {Trecere cursor pe randul urmator}
   READLN
END.
```

Fig. 9.9 a şi b prezintă rezultatele a două execuţii ale programului

Fig. 9.9a

```
Vector in ordine normala
   5  12  14  11  10
Vector in ordine inversa
  10  11  14  12   5
```

```
Vector in ordine normala
  10   5  11  13  10
Vector in ordine inversa
  10  13  11   5  10
```

Fig. 9.9b

Vectorul **a** este definit direct în secţiunea **VAR**. Se observă utilizarea funcţiei **RANDOM** pentru generarea de valori aleatoare pentru componentele vectorului **a** şi plasarea prioritară a instrucţiunii **RANDOMIZE** (& 8.2), absenţa acestei instrucţiuni ar provoca generarea unei secvenţe de numere aleatoare, dar cu aceleaşi valori la diverse execuţii ale programului.

Afişare valorii componentelor în ordine normală impune utilizarea instrucţiunii de afişare **WRITE** (& 5.5.2), pentru a conserva poziţia cursorului pe acelaşi rând după scrierea valorii fiecărei componente, încadrată în ciclu **FOR**. Deoarece şi după afişarea ultimei componente cursorul ecran rămâne poziţionat pe acelaşi rând, se impune o instrucţiune **WRITELN** pentru a provoca trecerea cursorului ecran pe rândul următor. În sintaxa comenzii este inclusă formatarea valorilor pe 4 poziţii (& 5.5.2).

Afişare valorii componentelor în ordine inversă impune utilizarea instrucţiunii de afişare **WRITE** a (& 5.5.2), încadrată într-un ciclu **FOR** descendent, în care indicele variază de la **N** spre 1.

9.3.3 Inversarea valorilor generate aleator ale unui vector în alt vector

E 9.3 Program TurboPascal pentru inversarea valorilor generate aleator ale unui vector în alt vector

Vector **a**	Vector **b**	Indici vectori	Suma indici
a[1]	b[n]	1=>n	1+n
a[2]	b[n-1]	2=>n-1	1+n
a[3]	b[n-2]	3=>n-2	1+n
.....	.....	.....	1+n
a[i]	b[n-i+1]	i=>n-i+1	1+n
.....	.....	.....	1+n
a[n]	b[1]	n=>1	1+n

Ne propunem inversarea valorilor vectorului **a** in vectorul **b**. Inversarea se va face prin plasarea in componenta **b[1]** a componentei **a[n]**, in **b[2]** pe **a[n-1]**,... , in **b[n]** pe **a[1]**. Se observă că suma indicilor componentelor de inversat este totdeauna **n+1**; deci, pentru cazul general, în componenta **b[i]** se va pune componenta **a[n+1-i]**.

```
PROGRAM vector3;
{Generare elemente vector prin numere aleatoare
cuprinse intre 5 si 15 si inversare in alt vector}
CONST n=5;
TYPE v=ARRAY[1..N] OF INTEGER;
VAR a,b:v;
   i:INTEGER;
   BEGIN
   {Generare/afisare elemente vector "a"}
   RANDOMIZE;
   WRITELN('Vector a');
   FOR i:=1 TO N DO
     BEGIN
IC   a[i]:=5+RANDOM(11); WRITE(a[i]:4);
     END;
   WRITELN;
   {Inversarea si afisare valori vector b}
   WRITELN('Vector b');
   FOR i:=1 TO n DO
     BEGIN
IC   b[i]:=a[n+1-i]; WRITE(b[i]:4);
     END;
      READLN
END.
```

Fig. 9.10

```
Vector a
   7   9  15   8  13
Vector b
  13   8  15   9   7
```

Fig. 9.11

Vectorii **a** şi **b** sunt declaraţi de acelaşi tip - tablou **v**, adică vor memora **N** componente de tip întreg. Generarea aleatoare a valorilor în vectorul **a** se face, pentru fiecare componentă, în primul ciclu **FOR** prin apel la funcţia **RANDOM** (& 8.2). Inversarea se produce de asemenea pentru fiecare componentă, în al doilea ciclu **FOR**, care conţine o instrucţiune compusă (& 5.3), în care sunt incluse două instrucţiuni: inversarea propriu-zisă conform relaţiei deduse şi afişarea formatată a valorilor

9.3.4 Inversarea valorilor generate aleator ale unui vector în acelaşi vector

E 9.4 Program TurboPascal pentru inversarea valorilor generate aleator ale unui vector în acelaşi vector

Vector **a**	Vector **b**	Indici vectori	Suma indici
a[1]	a[n]	1=>n	1+n
a[2]	a[n-1]	2=>n-1	1+n
a[3]	a[n-2]	3=>n-2	1+n
.....	.....	.....	1+n
a[i]	a[n-i+1]	i=>n-i+1	1+n
.....	.....	.....	1+n
a[n]	a[1]	n=>1	1+n

Inversarea se va face, *utilizând o variabila intermediară* (& 2.5.2), prin inversarea primei componente cu ultima, a doua cu penultima, până la mijloc, definit prin **n div 2** (& 4.2.1) indiferent daca **n** este par sau impar. Deci vectorul se parcurge într-un ciclu **FOR** de la **1** până la **n div 2**, schimbând pe **a[i]** cu **a[n+1-i]**.

```
PROGRAM vector4;
{Inversarea valorilor unui vector in acelasi vector}
CONST n=5;
VAR a:ARRAY[1..N] OF INTEGER;
   i, memo:INTEGER;
BEGIN
   {Generare /afisare componente vector "a"}
   WRITELN('Vector initial');
   RANDOMIZE;
   FOR i:=1 TO N DO
      BEGIN
IC       a[i]:=5+RANDOM(11); WRITE(a[i]:4);
      END;
      WRITELN;
   {Inversare si afisare valori vector}
   FOR i:=1 TO n DIV 2 DO
      BEGIN
         memo:=a[i];
IC       a[i]:=a[n+1-i];
         a[n+1-i]:=memo;
      END;
   WRITELN('Vector inversat');
   FOR i:=1 TO n DO WRITE(a[i]:4);
   READLN
END.
```

Fig. 9.12

```
Vector initial
  11  12   6  15   9
Vector inversat
   9  15   6  12  11
```

Fig. 9.13

Generarea aleatoare a valorilor în vectorul **a** se face în primul ciclu **FOR** prin apel la funcţia **RANDOM** (& 8.2). Inversarea se produce pentru fiecare componentă, în al doilea ciclu **FOR**, care conţine o instrucţiune compusă (& 5.3), în care sunt incluse cele trei instrucţiuni necesare inversării celor două componente ale aceluiaşi vector, prin intermediul variabilei intermediare **memo**. În ultimul ciclu **FOR** are loc afişarea formatată a valorilor inversate ale vectorului

9.3.5 Minimul şi maximul unui vector

E 9.5 Program TurboPascal pentru determinarea minimului şi maximul unui vector

Se realizează prin parcurgerea tuturor componentelor vectorului într-un ciclu **FOR** si actualizarea variabilei **minim** şi **maxim** la realizarea condiției de valoare a componentei vectorului mai mică respectiv mai mare decât variabila corespunzătoare. Se aplică algoritmul de calcul a minimului /maximului **(**& 2.5.5**)**, pentru fiecare componentă a vectorului.

```
PROGRAM vector5;
{Generare elemente vector prin numere aleatoare
si determinarea simultana a
minimului /maximului}
CONST n=5;
VAR a:ARRAY[1..N] OF INTEGER;
   i, minim, maxim:INTEGER;
BEGIN
   {Generare elemente ale vector "a" prin
   valori aleatoare si afisare componente}
   RANDOMIZE;
   WRITELN('Vector initial');
   FOR i:=1 TO N DO
     BEGIN
IC     a[i]:=5+RANDOM(10); WRITE(a[i]:3);
     END;
   WRITELN;
   {Determinare minim si maxim}
   minim:=a[1]; maxim:=a[1];
   FOR i:=2 TO n DO
     BEGIN
       IF a[i]<minim THEN minim:=a[i];
IC     IF a[i]>maxim THEN maxim:=a[i];
     END;
   WRITELN('Minim=', minim:3);
   WRITELN('Maxim=', maxim:3);
   READLN
END.
```

Fig. 9.14

```
Vector initial
 11 13  9 14  8
Minim=  8
Maxim= 14
```

Fig. 9.15

Variabilele **minim** şi **maxim** vor memora valorile minime respectiv maxime din vectorul **a**, ele fiind inițializate cu valorile generate în prima poziție a vectorului. Printr-un ciclu **FOR**, se parcurg pozițiile vectorului, începând de la a doua până la ultima, comparând succesiv valoarea componentei curente **a[i]** a vectorului cu valoarea variabilelor **minim** şi **maxim**; la îndeplinirea condiției din instrucțiunile **IF**, variabila preia valoarea curentă a componentei. În final se afişează valorile extreme calculate. Algoritmul extinde, la un vector cu **N** componente, problema determinării extremului aplicat asupra a 4 numere **(**& 2.5.5**)**.

9.3.6 Sumarea componentelor unui vector

E 9.6 Program TurboPascal pentru sumarea componentelor unui vector

Se realizează prin parcurgerea tuturor componentelor vectorului vectorului într-un ciclu **FOR** şi sumarea lor, aplicând algoritmul de sumare (& 2.5.9). Singura diferenţă este dată de faptul că valorile vectorului, nu sunt citite de la tastatură, ci sunt precizate iniţial în vector. Presupunem că dorim să sumăm consumuri lunare pe 12 luni.

Fig. 9.16

```
PROGRAM vector6; {Sumare elemente vector}
TYPE enum_luni=(IAN,FEB,MAR,APR,MAI,IUN,IUL,AUG,SEP,OCT,NOI,DEC);
CONST cl:ARRAY[IAN..DEC] OF INTEGER=(10,20,30,40,50,60,70,80,90,100,110,120);
VAR luna:enum_luni;
   total_consum:INTEGER;
BEGIN
   total_consum:=0; {Initializare variabila suma}
   FOR luna:=IAN TO DEC DO
     BEGIN
       total_consum:=total_consum+cl[luna];
IC     WRITELN (cl[luna]:5, total_consum:5)
     END;
   WRITELN('Suma=', total_consum:7);
   READLN
END.
```

```
   10   10
   20   30
   30   60
   40  100
   50  150
   60  210
   70  280
   80  360
   90  450
  100  550
  110  660
  120  780
Suma=    780
```

Fig. 9.17

În secţiunea **TYPE** se defineşte o enumerarea **enum_luni** (& 9.1.1), care memorează numele lunilor. Prin secţiunea **CONST** se defineşte vectorul **cl**, ai cărui indici, nu sunt de tip numeric, ci pot parcurge valorile enumerării; deci poziţia în vector se va identifica prin numele lunii, în intervalul (& 9.1.2) specificat; vectorul va memora consumurile lunare (valori întregi), specificate valoric în aceeaşi secţiune **CONST**. Secţiunea **VAR** conţine două variabile, **luna** fiind declarată de acelaşi tip ca şi enumerarea **enum_luni**, iar **total_consum** de tip numeric întreg, pentru memorarea sumei finale a consumurilor.

Ciclul **FOR** utilizează ca şi contor variabila **luna**, prin aceasta se parcurge domeniul declarat al lunilor, concomitent cu sumarea consumurilor lunare (valorile din vectorul **cl**) în variabila de sumare **total_consum**. Instrucţiunea de afişare din interiorul ciclului nu este necesară din punct de vedere al algoritmului, ci este plasată pentru a oferi posibilitatea evidenţierii evoluţiei calculelor numerice în timpul sumării.

Pentru sumarea altor valori lunare singura linie din program care trebuie modificată este declaraţia **CONST**, în care sunt precizate valoric consumurile lunare.

Se remarcă claritatea programului, obţinută ca o consecinţă a alegerii unor nume sugestive pentru variabile, precum şi a utilizării adecvate a tipurilor de date oferite de TurboPascal.

9.3.7 Contorizarea componentelor unui vector. Procedurile "INC" şi "DEC"

E 9.7 Program TurboPascal pentru contorizarea componentelor unui vector

Presupunem că dorim să contorizăm, separat pentru valorile pozitive şi negative, temperaturile medii lunare pe 12 luni. Operația se realizează prin parcurgerea tuturor componentelor vectorului vectorului într-un ciclu **FOR** şi contorizarea separată a temperaturilor pozitive şi negative, aplicând algoritmul de contorizare (& 2.5.12). Singura diferență este dată de faptul că valorile vectorului, nu sunt citite de la tastatură, ci sunt precizate initial în vector.

Fig. 9.18

```
PROGRAM vector7; {Contorizare componente vector}
TYPE enum_luni=(IAN,FEB,MAR,APR,MAI,IUN,IUL,AUG,SEP,OCT,NOI,DEC);
CONST temp:ARRAY[IAN..DEC] OF INTEGER=(-4,-3,2,5,12,20,22,23,18,16,5,-5);
VAR luna:enum_luni;
   temp_poz, temp_neg:INTEGER;
BEGIN
   temp_poz:=0; temp_neg:=0; {Initializare contori}
   FOR luna:=IAN TO DEC DO
     BEGIN
       IF temp[luna]>0 THEN
         INC(temp_poz)
       ELSE
         INC(temp_neg);
       WRITELN (temp[luna]:5, temp_poz:5, temp_neg:5)
     END;
   WRITELN('Total luni pozitive', temp_poz:3);
   WRITELN('Total luni negative', temp_neg:3);
   READLN
END.
```

IC

TurboPascal dispune de procedura **INC**(N,R) de incrementare a numărului N cu rația R, echivalentă relației N:=N+R. Pentru R=1, procedura devine **INC**(N). Există şi procedura similară **DEC**(N,R), echivalentă relatiei N:=N-R.

```
   -4    0    1
   -3    0    2
    2    1    2
    5    2    2
   12    3    2
   20    4    2
   22    5    2
   23    6    2
   18    7    2
   16    8    2
    5    9    2
   -5    9    3
Total luni pozitive  9
Total luni negative  3
```

temp_neg

temp_poz

Fig. 9.19

Prin secțiunea **CONST** se defineşte vectorul **temp**, ai cărui indici, nu sunt de tip numeric, ci pot parcurge valorile enumerării; vectorul va memora mediile lunare ale temperaturilor zilnice / lună (valori întregi pozitive sau negative), specificate valoric în aceeaşi secțiune **CONST**. Secțiunea **VAR** conține trei variabile, **luna** fiind declarată de acelaşi tip ca şi enumerarea **enum_luni**, iar **temp_poz** şi **temp_neg** de tip numeric întreg, pentru memorarea numărului de apariții a temperaturilor pozitive respectiv negative. Contorizarea se efectuează în ciclul **FOR** prin intermediul variabilelor contor **temp_poz** şi **temp_neg**, care cresc cu 1 funcție de pozitivitatea sau negativitatea temperaturii. Instrucțiunea de afişare din interiorul ciclului nu este necesară din punct de vedere al algoritmului, ci este plasată pentru a oferi posibilitatea evidențierii evoluției calculelor numerice în timpul execuției.

9.3.8 Căutarea secvențială într-un vector

E 9.8 Program TurboPascal pentru căutarea secvențială într-un vector

Fig. 9.20

```
PROGRAM vector8;
{Generare elemente vector prin numere aleatoare
cuprinse intre 1 si 10 si cautare secventiala
a unei valori printre elementele vectorului }
CONST n=5;
VAR a:ARRAY[1..N] OF INTEGER;
   i, numar, pozitia :INTEGER;
   gasit:BOOLEAN;
BEGIN
  {Generare si afisare componente vector}
  RANDOMIZE;
  WRITELN('Vector generat aleator');
  FOR i:=1 TO N DO
    BEGIN
      a[i]:=1+RANDOM(10);
      WRITE(a[i]:3);
    END;
  WRITELN;
  WRITELN('Numarul cautat-intre 1 si 10');
  READLN(numar);
  {Cautare numar}
  gasit:=FALSE; pozitia:=0; i:=1;
  WHILE (i<=N) AND (gasit=False) DO
    IF  a[i]=numar THEN
      BEGIN
        pozitia:=i;
        gasit:=TRUE;
      END
    ELSE
        INC(i);
  IF pozitia = 0 THEN
      WRITELN('Numar inexistent in vector')
  ELSE
      WRITELN('Numar cautat in pozitia=', pozitia:3);
  READLN
END.
```

IC

IC

Se realizează prin parcurgerea tuturor componentelor vectorului într-un ciclu **WHILE (**& 7.1**)** din care se iese la regăsirea primei apariții a numărului căutat sau în urma parcurgerii tuturor componentelor (număr inexistent).

```
Vector generat aleator
  8  2  6 10  7
Numarul cautat-intre 1 si 10
10
Numar cautat in pozitia=  4
```

```
Vector generat aleator
  7  5  2  7  9
Numarul cautat-intre 1 si 10
8
Numar inexistent in vector
```

Fig. 9.21

Numărul căutat se citește în variabila **număr**. Variabila **poziția** cu valoarea inițială **0,** este rezervată pentru a memora poziția în vector, în cazul în care numărul va fi găsit. Variabila logică **gasit** şi contorul "**i**" controlează finalizarea ciclului **WHILE**, inițial având valorile **FALSE** respectiv **1,** ciclu prin care se parcurg componentele vectorului (contorul "**i**" crește cu 1 la fiecare repetiție generând astfel indicii vectorului). Dacă numărul este găsit, se memorează poziția din vector iar **gasit** primeste valoarea **TRUE**, ceea ce va provoca iesirea din ciclu **WHILE**. Din ciclu se iese şi după parcurgerea completă a vectorului. ceea ce înseamnă că numărul nu a fost găsit. Funcție de situația generată, prin instrucțiunea finală **IF** se afișează poziția din vector unde s-a găsit prima apariție a numărului sau, în cazul inexistenței numărului căutat, se va afişa un mesaj de atenționare.

9.3.9 Ordonarea crescătoare a unui vector prin metoda selecţiei minimului

E 9.9 Program TurboPascal pentru ordonarea crescătoare a unui vector prin metoda selecţiei minimului

```
PROGRAM vector9;
{Ordonare crescatoare vector prin selectie minim}
CONST n=5;
VAR a:ARRAY[1..n] OF INTEGER;
   i, j, k, memo :INTEGER;
   BEGIN
   {Generare elemente si afisare componente}
   RANDOMIZE;
   WRITELN('Vector initial - neordonat');                    α
   FOR i:=1 TO N DO
      BEGIN
IC    a[i]:=RANDOM(10); WRITE(a[i]:3);                       β
      END;
   WRITELN; {Trecere cursor pe randul urmator}
   {Ordonare}
   FOR i:=1 TO n-1 DO
      BEGIN
         • FOR j:=i+1 TO n DO IF a[j]<a[i] THEN
            BEGIN
IC          IC memo:=a[i]; a[i]:=a[j]; a[j]:=memo;
            END;
IC       • WRITE('Vector intermediar');
         • FOR k:=1 TO n DO WRITE(a[k]:3);                   δ
         • WRITELN {Trecere cursor pe randul urmator}
      END;
   WRITELN('Vector ordonat');
   FOR I:=1 TO n DO WRITE(a[i]:3);                           ε
READLN
END.
```

Fig. 9.22

```
Vector initial - neordonat                                   α
  7  2  1  9  3
Vector intermediar    1  7  2  9  3                          β
Vector intermediar    1  2  7  9  3
Vector intermediar    1  2  3  9  7                          δ
Vector intermediar    1  2  3  7  9
Vector ordonat
  1  2  3  7  9                                              ε
```

Fig. 9.23

Ne propunem ordonarea crescătoare a unui vector. Metoda selecţiei minimului constă în următorul raţionament:

Se fixează primul element al vectorului şi se compară pe rând cu următoarele (de la al doilea la ultimul). Dacă elementul fixat este mai mare decât cele comparate, se impune inversarea valorilor lor, astfel că, la sfârşitul acestei prime treceri prin vector, pe prima poziţie se află cel mai mic element din vector, fără însă ca procesul să fie finalizat.

În continuare se mai execută o nouă trecere, fixându-se al doilea element, care se compară cu următoarele (de la al treilea la ultimul), efectuându-se inversarea lor, dacă elementul fixat este mai mare decât cele comparate. La sfârşitul acestei a doua treceri, pe a doua poziţie în vector se află înregistrată a doua valoarea minimă din vector.

Procedeul se continuă identic, fiind necesare de efectuat **n-1** treceri, la finalul cărora vectorul va deveni ordonat crescător.

Programarea trecerilor repetate prin vector se realizează prin intermediul unui ciclu **FOR** cu contor "**i**", al cărui domeniu de valori este de la **1** la **n-1**, deoarece numărul de treceri necesar este **n-1**.Valoarea fixată este valoare **a[i],** aflată în vector pe poziția fixată "**i**". Acest ciclu conține o instrucțiune compusă, formată din următoarele structuri (marcate cu • în fig. 9.22):

- programarea parcurgerii următoarelor componente în vederea comparării cu componenta fixată **a[i]** se realizează printr-un ciclu **FOR** cu contor "**j**", al cărui domeniu de valori este de la "**i+1**" la **n**, adică de la următoarea poziție față de cea curent fixată ("**i**") până la ultima poziție din vector **n**; valorile vectorului sunt parcurse pe rând prin valorile **a[j]**; în acest ciclu se execută o instrucțiune de decizie **IF** în care se compară dacă valorile din pozițiile parcurse succesiv **a[j]** sunt mai mici poziția fixată **a[i]**, situație în care se inversează valorilor între ele, folosind algoritmul de inversare a două variabile - utilizând variabila intermediară **memo** (& 2.5.2); cele 3 instrucțiuni de inversare sunt încadrate într-o instrucțiune compusă, deoarece toate aparțin instrucțiunii de decizie **IF**, care la rândul ei aparține ciclului de contor "**j**".
- instrucțiunile marcate prin δ în fig. 9.22 nu sunt necesare din punct de vedere al algoritmului, ci sunt plasate pentru a oferi posibilitatea evidențierii evoluției calculelor numerice în timpul execuției. Aceste instrucțiuni au rolul de a afișa informativ starea momentană a valorilor din vector la sfârșitul fiecărei treceri (la fiecare repetiție a ciclul de contor "**i**"). Această afișare se produce într-un ciclu de contor "**k**", care parcurge toate pozițiile din vector (de la **1** la **n**).

Pentru a realiza ordonarea descrescătoare a elementelor vectorului, singura modificare care trebuie efectuată este modificarea semnului "**<**" în semnul "**>**", în condiția asociată instrucțiunii **IF**.

9.3.10 Ordonarea crescătoare a unui vector prin metoda BUBBLE-SORT

E 9.10 Program TurboPascal pentru ordonarea crescătoare a unui vector prin metoda BUBBLE-SORT

Metoda presupune realizarea mai multor treceri prin vector, fiecare trecere fiind programată prin ciclu **FOR** şi constă în parcurgerea succesivă a elementelor vectorului şi compararea componentelor vecine. Dacă poziția curentă este "**i**", poziția următoare este "**i+1**", deci domeniul contorului ciclului **FOR** "**i**" nu poate merge până la ultima poziție "**n**", deoarece poziția "**n+1**" nu există în vector. Din acest motiv domeniul contorului "**i**" este de la **1** la "**n-1**".

În ciclul **FOR** de contor i:=1 TO n-1, se compară elementul **a[i]** de pe poziția curentă "**i**" cu elementul următor **a[i+1]** de pe poziția "**i+1**" şi se inversează valorile între ele, dacă **a[i+1]<a[i]**, concomitent cu atribuirea valorii 1 variabilei **schimb** la fiecare inversare. Rolul acestei variabile este de a evidenția efectuarea inversiunii, prin valoarea **1** pe care o primeşte în cazul efectuării fiecărei inversiuni. Variabila **schimb** este inițializată cu **0** înainte de fiecare trecere (execuție a ciclului **FOR**). Inversarea foloseşte algoritmul de inversare a două variabile - utilizând variabila intermediară **memo** (& 2.5.2).

Fig. 9.24

```
PROGRAM vector10;
{Ordonare crescatoare vector -metoda BUBBLE-SORT}
CONST n=5;
VAR a:ARRAY[1..N] OF INTEGER;
   i, k, schimb, memo :INTEGER;
BEGIN
   {Generare si afisare componente vector}
   WRITELN('Vector neordonat');                          α
   RANDOMIZE;
   FOR i:=1 TO N DO
      BEGIN
IC    a[i]:=RANDOM(10); WRITE(a[i]:3);                   β
      END;
   WRITELN; {Trecere cursor pe randul urmator}
   {Ordonare}
REPEAT
      schimb:=0; {Initializare variabila}
      FOR i:=1 TO n-1 DO IF a[i+1]<a[i] THEN
      BEGIN
         memo:=a[i];
         a[i]:=a[i+1];
IC       a[i+1]:=memo;
         schimb:=1;
         WRITE('Vector intermediar ');                   δ
         FOR k:=1 TO n DO WRITE(a[k]:3);
         WRITELN {Trecere cursor pe randul urmator}
      END;
UNTIL schimb=0;
   WRITELN('Vector ordonat');
   FOR I:=1 TO N DO WRITE(a[i]:3);                       ε
   READLN
END.
```

```
Vector neordonat                                         α
  7  4  3  2  1                                         β
Vector intermediar    4  7  3  2  1
Vector intermediar    4  3  7  2  1
Vector intermediar    4  3  2  7  1
Vector intermediar    4  3  2  1  7                      δ
Vector intermediar    3  4  2  1  7
Vector intermediar    3  2  4  1  7
Vector intermediar    3  2  1  4  7
Vector intermediar    2  3  1  4  7
Vector intermediar    2  1  3  4  7
Vector intermediar    1  2  3  4  7
Vector ordonat                                           ε
  1  2  3  4  7_
```

Fig. 9.25

Acest algoritm se repetă, prin ciclul **REPEAT**, până când variabila **schimb** rămâne la valoarea **0**, deci nu se mai produce nici o inversare, vectorul fiind ordonat, situație din care se iese şi din ciclul **REPEAT**. Ciclul **REPEAT-UNTIL** conține următoarele instrucțiuni:

- inițializarea variabilei schimb cu valoarea **0**;
- trecerea prin vector prin ciclu **FOR**, execută repetitiv instrucțiunea de decizie **IF**; numai la îndeplinirea condiției din instrucțiunea **IF** se execută instrucțiunea compusă, care include: efectuarea inversiunii elementelor vecine, atribuirea valorii **1** variabilei **schimb** şi grupul de instrucțiuni marcate cu δ în fig. 9.22, care nu sunt necesare din punct de vedere al algoritmului, ci au un rol informativ prin afişarea stării momentane a valorilor din vector la sfârşitul fiecărei treceri. Ciclul **REPEAT-UNTIL** se opreşte atunci când variabila **schimb** rămâne la valoarea inițializată **0**, deci nu s-a mai efectuat nici o inversiune în vector, ceea ce înseamnă că vectorul este ordonat.

Pentru ordonarea descrescătoare a elementelor vectorului, singura modificare care trebuie efectuată este modificarea semnului "**<**" în semnul "**>**", în condiția asociată instrucțiunii **IF**.

9.3.11 Contorizare în vector a aparițiilor fețelor la aruncarea unui zar

E 9.11 Program TurboPascal pentru contorizare în vector a aparițiilor fețelor la aruncarea unui zar

Ne propunem reluarea problemei contorizării numărului de apariții a fețelor unui zar, adică afișarea, pentru fiecare față, a numărului de apariții dintr-un total de N aruncări ale zarului, utilizând tablourile ca mijloc de simplificare a rezolvării problemei. Problema a fost rezolvată în & 8.2, utilizând câte un contor pentru fiecare față, fig. 8.4.

```
PROGRAM zaruri;
{Contorizare fete zar din N aruncari}
VAR  cf:ARRAY[1 .. 6] OF INTEGER;
   N, fata, i:INTEGER;
BEGIN
   RANDOMIZE; {Initializare generator numere aleatoare}
   WRITE('Numarul de aruncari ?'); READLN(N);
   FOR i:=1 TO N DO
     BEGIN
       fata:=1+random(6);
IC     WRITELN('Aruncarea=',i,'  fata=',fata);
       cf[fata]:=cf[fata]+1;
     END;
   FOR i:=1 TO 6 DO WRITELN('Contor fata ',i,' =',cf[i]);
   READLN
END.
```

Fig. 9.26

Fig. 9.27

```
Numarul de aruncari ?5
Aruncarea=1  fata=2
Aruncarea=2  fata=2
Aruncarea=3  fata=5
Aruncarea=4  fata=6
Aruncarea=5  fata=5
Contor fata 1 =0
Contor fata 2 =2
Contor fata 3 =0
Contor fata 4 =0
Contor fata 5 =2
Contor fata 6 =1
```

Simplificarea se bazează pe următoarea observație: vom echivala zarul cu 6 fețe cu un vector cu 6 poziții, corespunzătoare fețelor zarului; valorile din vector vor memora numărul de aruncări corespunzător feței şi deci această valoare trebuie mărită cu 1 de fiecare dată când a ieşit prin aruncare fața corespunzătoare. Deci indicele vectorului corespunde feței zarului, iar valoarea asociată corespunde numărului de aruncări. Exemplificând, dacă prin aruncarea zarului a ieşit fața a 3-a, valoarea din vector care trebuie mărită cu 1 este valoarea al cărei indice este 3, indice egal chiar cu valoarea feței. Prin aceasta se elimină necesitatea testării din versiunea anterioară realizată prin instrucțiunea **CASE**, al cărei scop era de a selecta contorul care trebuie mărit funcție de numărul feței.

Din fig. 9.26 comparativ cu fig. 8.4, se remarcă faptul că în loc de 6 instrucțiuni de contorizare incluse în instrucțiunea de selecție multiplă **CASE** se utilizează o singură instrucțiune de contorizare. Aceeaşi reducere se aplică şi la afişarea valorilor contorilor, deoarece în acest caz se poate lucra într-un ciclu **FOR** care conține o singură instrucțiune de afişare. Acest exemplu evidențiază facilitățile de programare care pot fi obținute prin utilizarea tablourilor, în locul utilizării de variabile individualizate, care trebuie manevrate individual, mărind programul prin instrucțiuni suplimentare.

9.3.12 Sumare şi produse de elemente într-o matrice

E 9.12 Program TurboPascal pentru sumare şi produse elemente matrice

```
PROGRAM matrice1;                                          Fig. 9.28
{Pentru o matrice se cere: suma tuturor elementelor,
suma elementelor de pe ultima linie / coloana,
produsul elementelor de pe diagonala principala / secundara}
CONST nl=3;   nc=4;
TYPE matrice = ARRAY [1..nl, 1..nc] OF INTEGER;
VAR m:matrice;
   i,j,ste,sul,suc,minlc,pdp,cc,pds:integer;
BEGIN
   {Generare si afisare elemente matrice}
   RANDOMIZE;
   FOR i:=1 TO nl DO
     BEGIN
       • FOR j:=1 TO nc DO
          BEGIN
IC        IC  m[i,j]:=1+RANDOM(6); WRITE(m[i,j]:3)
          END;
       • WRITELN; {Trecere cursor pe randul urmator}
     END;
ste:=0; {Suma tuturor elementelor}
FOR i:=1 TO nl DO
     FOR j:=1 TO nc DO ste:=ste+m[i,j];
sul:=0;  {Suma elementelor de pe ultima linie}
FOR j:=1 TO nc DO sul:=sul+m[nl,j];
suc:=0; {Suma elementelor de pe ultima coloana}
FOR i:=1 TO nl DO suc:=suc+m[i,nc];
{Produs elemente diagonala principala}
pdp:=1; minlc:=nl;
IF nc<minlc THEN minlc:=nc;
FOR i:=1 TO minlc DO pdp:=pdp*m[i,i];
{Produs elemente diagonala secundara}
pds:=1; cc:=nc;
FOR i:=1 TO minlc DO
     BEGIN
IC     pds:=pds*m[i,cc]; dec(cc);
     END;
WRITELN('Suma elemente   =',ste);
WRITELN('Suma ultima linie=',sul);
WRITELN('Suma ultima col. =',suc);
WRITELN('Produs diag. pr. =',pdp);
WRITELN('Produs diag. sec.=',pds);
READLN
END.
```

Pentru o matrice cu **nl** linii şi **nc** coloane să se calculeze următoarelor sume: suma tuturor elementelor **ste**, suma elementelor de pe ultima linie **sul** respectiv coloana **suc**, produsul elementelor de pe diagonala principala **pdp** respectiv secundară **pds**.

Fig. 9.29

```
 1  3  2  4
 1  2  2  2
 4  6  4  5
Suma elemente    =36
Suma ultima linie=19
Suma ultima col. =11
Produs diag. pr. =8
Produs diag. sec.=48
```

Accesarea elementelor unei matrici se face prin intermediul a doi indici, dintre care primul specifică linia iar al doilea coloana. Din acest motiv generarea elementelor matricii impune parcurgerea a două cicluri **FOR**: primul generează indicii de linii (ciclu de contor "**i**" de la **1** la **nl**) care execută repetat o instrucţiune compusă, formată din următoarele instrucţiuni (marcate cu simbolul •): **1)** al doilea ciclu **FOR** care generează indicii de coloană (ciclu de contor "**j**" de la **1** la **nc**) şi **2)** instrucţiunea de trecere a cursorului ecran pe rândul următor.

Ciclul de contor "**j**" include execuţia repetată a două instrucţiuni: generarea valorilor aleatoare în poziţia **i**,**j** a matricii **m** prin funcţia **RANDOM**, afişarea succesivă şi pe acelaşi rând a valorilor unei linii, prin instrucţiunea **WRITE** (& 5.3.2). Pentru ca toate liniile matricii să nu apară pe aceeaşi linie ecran, după generarea şi afişarea unei linii a matricii se trece cursorul ecran pe rândul următor prin instrucţiunea **WRITELN** (& 5.3.2).

Calculul sumei tuturor elementelor necesită parcurgerea matricii, care se face pe linii (& 9.3), fig. 9.30, iar operaţia de prelucrare aplicată este sumarea componentelor, conform algoritmului de sumare (& 2.5.9). Ciclul de contor "**j**" este interior ciclului de contor "**i**", prin aceste două cicluri îmbricate, matricea se parcurge astfel: se fixează câte o linie a matricii (descrisă prin indicele "**i**") şi se parcurg toate coloanele acestei linii fixate (descrise prin indicele "**j**"), apoi se trece la linia următoare şi din nou se parcurg toate coloanele acestei linii fixate, procedeul continuând până la epuizarea tuturor liniilor matricii. Deci ciclul de contor "**j**" se execută mai repede (& 7.3), fiind interior ciclului de contor "**i**".

Fig. 9.30

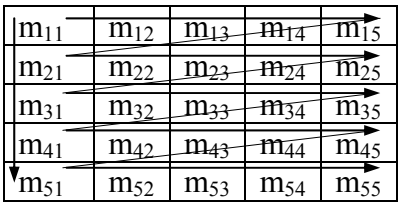
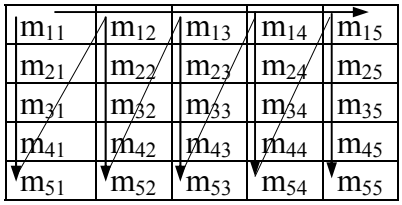


Fig. 9.31



Desigur că matricea poate fi parcursă şi pe coloane, ceea ce ar corespunde la schema din fig. 9.31.

Calculul sumei elementelor de pe ultima linie înseamnă sumarea a **nc** elemente, toate având fixate indicele ultimei linii la valoarea **nl**, indicele de coloană variind între **1** şi **nc**, prin intermediul ciclului **FOR** de contor "**j**".

Calculul sumei elementelor de pe ultima coloană înseamnă sumarea a **nl** elemente, toate având fixate indicele ultimei coloane la valoarea **nc**, indicele de linie variind între **1** şi **nl**, prin intermediul ciclului **FOR** de contor "**i**".

Pentru calculul produsului elementelor diagonala principala se impune determinarea numărului de elemente de pe această diagonală, memorat în variabila **minlc**, ca fiind minimul între numărul de linii **nl** şi numărul de coloane **nc,** calculat conform algoritmului din & 2.5.3. Calculul produsului elementelor de pe diagonala principală înseamnă produsul a **minlc** elemente, toate având proprietatea că indicele de linie este egal cu cel de coloană şi variind între **1** şi **minlc**, prin intermediul ciclului **FOR** de contor "**i**". Se aplică algoritmul produsului (& 2.5.11).

Numărul de elemente de pe diagonala secundară este tot **minlc**. Elementele de pe diagonala secundară au proprietatea că indicele de linie variază crescător de la **1** spre **minlc**, în timp ce indicele de coloană variază descrescător pornind de la valoarea iniţială **nc**. Calculul produsului elementelor de pe diagonala secundară înseamnă produsul a **minlc** elemente, toate având proprietatea că indicele de linie variază între **1** şi **minlc**, prin intermediul ciclului **FOR** de contor "**i**", indicele de coloană asociat porneşte de la valoarea iniţială **cc=nc** şi scade cu **1** la fiecare repetiţie a ciclului **FOR**, prin intermediul procedurii **DEC** (& 9.3.7). Se aplică algoritmul produsului (& 2.5.11). Variabila **cc** joacă rolul unui contor descrescător.

În general prelucrarea elementelor unei matrici necesită manipularea indicilor prin intermediul contorilor de ciclu, astfel ca valorile acestora să genereze valorile dorite ale indicilor matricii, accesând astfel poziţiile dorite din matrice. Numele contorului de ciclu sub care se face prelucrarea nu este important, ci valorile generate ale acestuia.

9.3.13 Inversare prima şi ultima linie şi coloană într-o matrice

E 9.13 Program TurboPascal pentru inversare linie/coloană într-o matrice

```
PROGRAM matrice2;
{Inversare prima si ultima linie respectiv
prima si ultima coloana intr-o matrice}
CONST nl=3; nc=4;
VAR m:ARRAY [1..nl, 1..nc] OF INTEGER;
   l,c,memo:integer;
BEGIN
   {Generare si afisare elemente matrice}
   RANDOMIZE;
   WRITELN('Matricea initiala');
   FOR l:=1 TO nl DO
     BEGIN
       FOR c:=1 TO nc DO
         BEGIN
           m[l,c]:=1+RANDOM(10); WRITE(m[l,c]:3)
         END;
       WRITELN; {Trecere cursor pe randul urmator}
     END;
   {Inversare prima cu ultima linie}
   FOR c:=1 TO nc DO
     BEGIN
       memo:=m[1,c];m[1,c]:=m[nl,c]; m[nl,c]:=memo
     END;
   WRITELN('Dupa inversare linii');
   FOR l:=1 TO nl DO
     BEGIN
       FOR c:=1 TO nc DO WRITE(m[l,c]:3);
       WRITELN; {Trecere cursor pe randul urmator}
     END;
   {Inversare prima cu ultima coloana}
   FOR l:=1 TO nl DO
     BEGIN
       memo:=m[l,1];m[l,1]:=m[l,nc]; m[l,nc]:=memo
     END;
   WRITELN('Dupa inversare coloane');
   FOR l:=1 TO nl DO
     BEGIN
       FOR c:=1 TO nc DO WRITE(m[l,c]:3);
       WRITELN; {Trecere cursor pe randul urmator}
     END;
   READLN
END.
```

Fig. 9.32

Pentru o matrice cu **nl** linii şi **nc** coloane să se inverseze elementele primei cu a ultimei linii, respectiv coloane.

```
Matricea initiala
  9  5  3  2
  3  6  7  2
  8  5  6  5
Dupa inversare linii
  8  5  6  5
  3  6  7  2
  9  5  3  2
Dupa inversare coloane
  5  5  6  8
  2  6  7  3
  2  5  3  9
```

Fig. 9.33

Inversare primei şi ultimei linii înseamnă parcurgerea coloanelor asociate prin intermediul ciclului **FOR**, de contor "**c**" care variază între **1** şi **nc** şi inversare elementului din linia **1** cu cel din linia **nl**, folosind variabila suplimentară **memo** (& 2.5.2).

Inversare primei şi ultimei coloane înseamnă parcurgerea liniilor asociate prin intermediul ciclului **FOR**, de contor "**l**" care variază între **1** şi **nl** şi inversare elementului din coloana **1** cu cel din coloana **nc**, folosind variabila suplimentară **memo** (& 2.5.2).

9.3.14 Maxime şi sume pe coloană într-o matrice

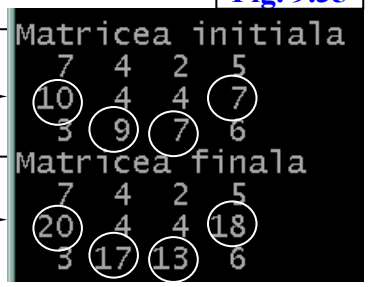
E 9.14 Program TurboPascal pentru maxime şi sume pe coloană în matrice

Fig. 9.34

```
PROGRAM matrice3;
{Inlocuire element maxim de pe
fiecare coloana a unei matrici
cu suma elementelor sale}
CONST nl=3; nc=4;
VAR m:ARRAY [1..nl, 1..nc] OF INTEGER;
   l,c,sumac,poz_lin,maxim:integer;
BEGIN
   {Generare/afisare elemente matrice}
   RANDOMIZE;
   WRITELN('Matricea initiala');
   FOR l:=1 TO nl DO
     BEGIN
       FOR c:=1 TO nc DO
         BEGIN
           m[l,c]:=1+RANDOM(10);
           WRITE(m[l,c]:3)
         END;
       WRITELN;
     END;
FOR c:=1 TO nc DO
  BEGIN
    • sumac:=0;
    • maxim := m[1,c]; poz_lin:=1;
    • FOR l:=1 TO nl DO
      BEGIN
        sumac := sumac + m[l,c];
        IF m[l,c] > maxim THEN
          BEGIN
            maxim :=m[l,c];
            poz_lin:=l;
          END
      END;
    • m[poz_lin,c] := sumac;
  END;
   WRITELN('Matricea finala');
   FOR l:=1 TO nl DO
     BEGIN
       FOR c:=1 TO nc DO WRITE(m[l,c]:3);
       WRITELN;
     END;
READLN
END.
```

Pentru o matrice cu **nl** linii şi **nc** coloane să înlocuiască elementul maxim de pe fiecare coloană cu suma elementelor de pe acea coloană.

Fig. 9.35



Variabila **sumac** va memora suma elementelor coloanei, iar **poz_lin** numărul liniei unde a fost detectat maximul coloanei.
Se vor parcurge succesiv coloanele prin ciclul de contor "**j**" în domeniul de la **1** până **nc**, şi pentru coloana "**c**" fixată se vor executa următoarele instrucţiuni, incluse în instrucţiunea compusă şi marcate cu simbolul • :

- iniţializarea cu 0 a variabilei de sumare **sumac**;
- iniţializarea cu **m[1,c]** a variabilei **maxim,** adică cu valoarea din prima linie a coloanei "**c**" şi a variabilei **poz_lin** cu **1**;
- parcurgerea liniilor coloanei fixate "**c**", prin ciclul **FOR** de contor "**l**", în care se execută însumarea propriu-zisă şi instrucţiunea **IF** pentru determinarea maximului şi a

poziției acestuia, adică numărul liniei unde s-a detectat maximul, memorat în variabila **poz_lin**;

- după finalizarea ciclului **FOR** de contor "**l**", deci după ce s-a calculat suma elementelor coloanei "**c**" şi s-a determinat maximul şi poziția acestuia, se efectuează înlocuirea sumei calculate **sumac** pe poziția din matrice corespunzătoare coloanei fixate "**c**" şi liniei unde s-a detectat anterior maximul, adică **poz_lin**, prin instrucțiunea de atribuire.

Pentru acest algoritm este foarte importantă includerea inițializării variabilei **maxim**, a poziției acestuia **poz_lin** precum şi a sumei **sumac** în ciclul de contor "**c**", deoarece aceste mărimi se referă la fiecare coloană în parte şi nu la matricea în ansamblul ei. Deci inițializările trebuie efectuate la fiecare repetiție a ciclului **FOR** de contor "**c**", în caz contrar rezultatele nu ar fi cele corecte.

9.3.15 Desfăşurarea unei matrici într-un vector

E 9.15 Program TurboPascal pentru desfăşurarea elementelor unei matrici într-un vector

Ne propunem depunerea elementelor unei matrici cu **nl** linii şi **nc** coloane într-un vector, în care numărul final de componente va rezulta din relația: **nv** = **nl** x **nc**. Schema de depunere este prezentată în fig. 9.36.

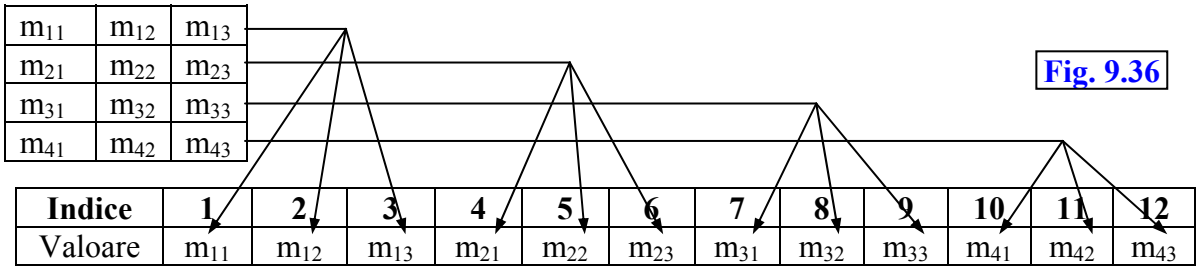


Fig. 9.36

Din schema din fig. 9.36, se observă următoarele:

- depunerea se realizează prin parcurgerea matricii pe linii, deci pentru o linie fixată "**l**" se vor parcurge coloanele asociate acesteia, ceea ce înseamnă că ciclul de coloană de contor "**c**" se execută mai repede, deci va fi ciclu interior în raport cu cele de linii (& 7.3);
- depunerea începe de poziția **1** în vector, poziție care creşte cu **1** pentru fiecare depunere efectuată; această observație permite utilizarea algoritmului contorizării & 2.5.12, astfel încât poziția depunerii în vector să fie specificată prin intermediul unei variabile contor.

Programul este prezentat în fig. 9.37, iar rezultatele acestuia în fig. 9.38.

Variabila **contor** va genera prin valorile sale pozițiile de depunere în vectorul **v**. Procesul de generare aleatoare a valorilor în matricea **m** este combinat cu procesul de depunere succesivă în vector a valorilor generate ale matricii.

Ciclul **FOR** exterior de contor "**l**" generează indicii liniilor matricii, în domeniul de la **1** la **nl**. În acest ciclu se execută repetitiv instrucțiunea compusă, care include următoarele instrucțiuni (marcate prin simbolul •):

Fig. 9.37

```
PROGRAM matrice4;
{Desfasurare matrice in vector}
CONST nl=3; nc=4; nv=nl*nc;
TYPE  nr_lin=1..nl; nr_col=1..nc;
VAR   m:ARRAY[nr_lin, nr_col] OF INTEGER;
      v:ARRAY[1..nv] OF INTEGER;
      l,c,contor:integer;
BEGIN
   {Generare/afisare elemente matrice}
   RANDOMIZE;
   WRITELN('Matricea initiala');
   contor:=0;
   FOR l:=1 TO nl DO
     BEGIN
       FOR c:=1 TO nc DO
         BEGIN
           m[l,c]:=1+RANDOM(10);
           WRITE(m[l,c]:3);
           INC(contor);
           v[contor]:=m[l,c];
         END;
       WRITELN
     END;
   WRITELN('Vector final');
   FOR l:=1 TO nv DO WRITE(v[l]:3);
   READLN
END.
```

IC (FOR c:=1 TO nc DO)

IC (BEGIN … END;)

- Ciclul **FOR** interior de contor "**c**" generează indicii liniilor matricii, în domeniul de la **1** la **nc**;
- instrucțiunea de trecere a cursorului pe rândul următor.

Ciclul interior de contor "**c**" execută repetitiv o instrucțiune compusă, care conține următoarele instrucțiuni:

Fig. 9.38

```
Matricea initiala
  7  9  1  9
  9  8  3  3
  2  9  7  1
Vector final
  7  9  1  9  9  8  3  3  2  9  7  1
```

- generarea prin funcția **RANDOM** a valorilor în matricea **m**, în linia "**l**" şi coloana "**c**";
- afişarea valorii generate a matricii;
- incrementarea contorului cu **1**, prin procedura **INC**;
- depunerea valorii generate a matricii **m[l,c]** în vectorul **v**, în poziția specificată de valoarea curentă a variabilei **contor**.

Secțiunea de declarații a programului:

- secțiunea **CONST** - declară valorile constantelor **nl** şi **nc** şi calculează valorea constantei **nv**;
- secțiunea **TYPE** – declară valorile variabilelor interval **nr_lin** şi **nr_col**, care se vor utiliza la definirea tipului matricii;
- secțiunea **VAR** – declară matricea **m**, vectorul **v**, şi variabile simple **l**,**c**,**contor** toate de tip întreg.

Instrucțiunea **INC** se va executa de **nl** x **nc** ori, fiind inclusă în ciclul interior de contor "**c**", care la rândul său este inclus în ciclul exterior de contor "**l**", crescând cu **1** valoarea variabilei **contor** la fiecare repetiție şi parcurgând prin aceasta valorile de la **1** la **nv** = **nl** x **nc**, generând astfel succesiv valorile indicilor din vector unde trebuie depozitate valorile din matrice.

Orice prelucrare într-un tablou accesează elementele tabloului prin intermediul indicilor, iar scopul care trebuie urmărit atunci când se lucrează cu tablouri, este de a găsi acel algoritm prin care să fie generate valorile dorite ale indicilor prin intermediul contorilor de ciclu, pentru a putea include prelucrarea într-o structură repetitivă.

9.3.16 Suma elementelor marginale ale unei matrici

E 9.16 Program TurboPascal pentru sumarea elementelor marginale ale unei matrici

```
PROGRAM matrice5;
{Suma elementelor marginale ale unei matrici}
CONST nl=3; nc=4;
VAR   m:ARRAY[1..nl, 1..nc] OF INTEGER;
      l,c,suma:integer;
BEGIN
   {Generare/afisare elemente matrice}
   RANDOMIZE;
   WRITELN('Matricea initiala');
   FOR l:=1 TO nl DO
      BEGIN
        FOR c:=1 TO nc DO
          BEGIN
            m[l,c]:=1+RANDOM(10);
            WRITE(m[l,c]:3);
          END;
        WRITELN
      END;
   suma:=0;
   {Sumare prima si ultima linie}
   FOR c:=1 TO nc DO suma:=suma+m[1,c]+m[nl,c];
   {Sumare prima si ultima linie coloana}
   FOR l:=2 TO nl-1 DO suma:=suma+m[l,1]+m[l,nc];
   WRITELN ('Suma =',suma:4);
   READLN
END.
```

Fig. 9.40

Ne propunem sumarea elementelor marginale ale unei matrici cu **nl** linii şi **nc** coloane, conform schemei prezentate în fig. 9.39.

m_{11}	m_{12}	m_{13}	m_{14}	m_{15}
m_{21}	m_{22}	m_{23}	m_{24}	m_{25}
m_{31}	m_{32}	m_{33}	m_{34}	m_{35}
m_{41}	m_{42}	m_{43}	m_{44}	m_{45}
m_{51}	m_{52}	m_{53}	m_{54}	m_{55}

Fig. 9.39

```
Matricea initiala
  7  2  5 10
  6  4  8  7
  8  1 10  8
Suma =  64
```

Fig. 9.41

După cum se observă din fig. 9.39, sumarea se efectuează astfel:

- se adună elementele de pe prima şi ultima linie a matricii;
- se adună elementele de pe prima respectiv ultima coloană, mai puțin elementele extreme ale coloanelor, care s-au adunat deja prin sumarea anterioară.

Prima sumare este programată prin ciclul **FOR** de contor "**c**" în domeniul de la **1** la **nc**, în care se însumează elementele matricii din linia **1** respectiv linia **nl**, pentru toate coloanele asociate acestora.

A doua sumare este programată prin ciclul **FOR** de contor "**l**" în domeniul de la **2** la **nl**, în care se însumează elementele matricii din coloana **1** respectiv coloana **nc**, pentru toate liniile asociate acestora, mai puțin linia **1** şi **nl**, pentru a evita dublarea sumării elementelor extreme.

10. TIPURI DE DATE STRING (ŞIR DE CARACTERE)

10.1 DEFINIREA TIPURILOR ŞIR DE CARACTERE

Prelucrarea datelor utilizează în mod frecvent date sub forma unor şiruri de caractere. Limbajul TurboPascal permite manipularea acestora în următoarele forme: definirea de tipuri de date şir, definirea constante respectiv variabile şir de caractere. Pentru manipularea şirurilor de caractere limbajul TurboPascal operează cu un tip de date specializat, denumit, **STRING** (**şir de caractere**).

Vom înțelege prin **STRING** un tip de date descris printr-o structură de tip vector (& 9.2.1), cu elemente de tip **CHAR** (& 4.2.4), care memorează, pe lângă succesiunea de caractere, şi lungimea şirului, adică numărul de caractere existente efectiv în şir, limitat la maxim 255 caractere.

Numărul de caractere este limitat superior la un număr maximal, denumit *dimensiunea* şirului, care însă nu poate depăşi valoarea 255. Valoarea unui şir este constituită din succesiunea caracterelor, numărul lor poate fi cuprins între 0 şi dimensiunea maximă declarată a sa. Definiția unui tip **STRING** se poate realiza prin una din următoare sintaxe:

TYPE Nume_string = STRING

TYPE Nume_string = STRING[dimensiune]

În prima formă a sintaxei, dimensiunea nu este declarată explicit, fiind considerată la valoarea de 255. În a două formă a sintaxei, dimensiunea este limitată maximal la valoarea din declarație, declarația nu poate impune însă o valoare mai mare decât 255.

Limitarea de 255 decurge din modul particular de reprezentare a acestor tipuri de date în calculator: şirurile sunt memorate ca succesiuni de locații de memorie, pe lungime de **dimensiune+1** octeți, unde primul octet memorează lungimea curentă a şirului, prin intermediul unui caracter al cărui cod este egal cu lungimea şirului. Dar codificarea caracterelor este limitată superior la valoarea 255, ceea ce provoacă şi limitarea lungimii şirurilor, prin inexistența unui caracter cu cod mai mare, care să poată exprima lungimea mai mare de 255.

Dacă lungimea şirului este 0 se obține şirul vid, adică şirul care nu conține nici un caracter, reprezentat prin constanta **‘’**.

Valorile unui tip de **STRING** sunt accesate prin intermediul variabilelor, ceea ce impune definirea variabilelor de tip **STRING**. Aceasta se poate realiza în două moduri:

a) definirea tipului şirului în secțiunea **TYPE**, urmat de declarația variabilelor de tip şir în secțiunea **VAR** a secțiunii declarative (& 4.4); această formă de declarare este utila la declararea mai multor variabile şir de acelaşi lungime;

b) declararea directă şi implicită a variabilei de tip şir în secțiunea **VAR**, prin specificarea caracteristicilor, fără a mai declara anterior tipul acestuia în secțiunea **TYPE**.

Exemplul 1 oferă exemple de definire a tipurilor respectiv variabilelor de tip şir. Astfel, în secțiunea **TYPE** se defineşte tipul a două date de tip şir: **sir1** fiind declarat ca şir fără a impune explicit o dimensiune, deci va fi limitat la

Exemplu 1

```
TYPE sir1 = STRING;
     sir2 = STRING[35];
VAR  a : sir1;
     b : sir2;
     c : STRING[48];
```

255 caractere, a doua **sir2**, fiind limitată explicit la 35 caractere. În secţiunea **VAR** se definesc două variabile, **a** de tipul **sir1**, **b** de tip **sir2**, care preiau caracteristicile declarate în secţiunea **TYPE**, suplimentar fiind definită variabila **c**, cu lungime declarată de 48 de caractere, declaraţia fiind efectuată direct în secţiunea **VAR**, fără predefinirea în secţiunea **TYPE**.

Variabilă de tip şir poate fi accesată în două moduri posibile:

- prin intermediul identificatorului asociat – care operează cu valoarea variabilei şir în totalitatea ei;
- prin accesarea unui caracter din şir - deoarece şirurile sunt structuri de tip vectorial, maniera de operare caracteristică vectorilor poate fi aplicată şi şirurilor; astfel, accesarea unui singur caracter se poate realiza prin construcţia **id_sir [expresie]**, unde **id_sir** este identificatorul şirului, iar expresie trebuie încadrată între paranteze drepte ca la vectori (& 9.2.1), iar valoarea calculată a acesteia trebuie să se încadreze între 0 şi dimensiunea şirului.

Exemplu 2 Pentru variabilele declarate în exemplul 1 şi valorile atribuite mai jos

```
a:='12345678901234567890123456789012345678901234567890abcde';
b:='TEST';
c:='ALFABET';
```

afişarea valorii primului octet, prin intermediul funcţiei **ORD** (& 4.2.4), oferă codul acestuia, echivalent numeric cu lungimea curentă a şirului:

```
WRITELN(ord(a[0]));   =>  65
WRITELN(ord(b[0]));   =>  4
WRITELN(ord(c[0]));   =>  7
```

iar accesarea individuală a caracterelor şirurilor se obţine prin specificarea între paranteze drepte a unui indice sau expresie, a cărui valoare specifică poziţia ocupată de caracter în cadrul succesiunii de caractere:

```
WRITELN(a[25]);  =>  5
WRITELN(b[3]);   =>  S
WRITELN(c[2]);   =>  L
```

Modificarea lungimii şirului (deci a valorii memorate în primul octet) se produce automat la modificarea numărului de caractere din şir printr-o operaţie specifică, dar poate fi efectuată şi prin program prin atribuirea:

sir[0] := #nr sau **sir[0] := CHR(ORD(nr))**

unde **nr** este un număr cuprins între 0 şi dimensiunea şirului (lungimea maximă admisă). Dacă **nr** are valoarea 0 se obţine şirul vid.

Depăşirea dimensiunii maxime a unui şir este semnalizată cu eroare **Error 76: Constant out of range.**

10.2 OPERAŢII CU ŞIRURI DE CARACTERE

Tipurile **STRING** sunt compatibile între ele, atât ca operanzi în expresii, cât şi în cadrul instrucţiunii de atribuire (& 5.2). Compatibilitatea între tipurile **STRING** cu cele de tip **CHAR** (& 4.2.4) este asigurată numai unidirecţional, în sensul echivalenţei tipului **CHAR** cu a tipului **STRING** în orice context, reciproca nefiind însă valabilă.

Un avantaj a datelor de tip **STRING** este că acestea pot fi utilizate în instrucţiuni de citire (& 5.5.1) respectiv scriere (& 5.5.2).

Ordonarea şirurilor de caractere se bazează pe codurile numerice ale caracterelor (& 4.2.4). Din acest punct de vedere operatorii relaţionali < , > , <= , >= , = pot fi aplicaţi asupra datelor de tip **STRING**, rezultatul comparaţiei fiind determinat funcţie de codurile asociate caracterelor din şir. La compararea unei variabile **CHAR** cu o variabilă de tip **STRING** se consideră lungimea variabilei **CHAR** la valoarea 1. Compararea decurge după următorul algoritm: se compară primul caracter din cele două şiruri:

- dacă codul primului caracter din primului şir este mai mic decât codul primului caracter din al doilea şir, atunci primul şir este mai mic decât al doilea, indiferent de restul caracterelor din cele două şiruri;
- dacă codul primului caracter din primului şir este mai mare decât codul primului caracter din al doilea şir, atunci primul şir este mai mare decât al doilea, indiferent de restul caracterelor din cele două şiruri;
- în caz de egalitate, se algoritmul se aplică succesiv asupra următoarelor caractere.

Exemplu 3 ‘A’ < ‘ALFABET’ ; ‘AX’ > ‘ALFABET’ ; ‘ALX’ > ‘ALFABET’

O altă operaţie utilizată în prelucrarea şirurilor este concatenarea (alipirea) a două sau mai multe şiruri, care se realizează prin operatorul + sau funcţia **concat**. Operanzii pot fi de tip variabile, constante **STRING** sau **CHAR**, iar rezultatul va fi de tip **STRING**. Dacă în urma operaţiei, dimensiunea şirului obţinut depăşeşte 255, acesta va fi trunchiat la primele 255 caractere, pentru a respecta limitarea dimensiunii admise pentru acest tip de date.

Se pot defini şi tablouri de şiruri caractere, prin declaraţia:

```
TYPE nume =  ARRAY [1..N] OF STRING[NS]
```

unde **nume** reprezintă identificatorul tabloului, **N** numărul de elemente al tabloului, iar NS dimensiunea maximală a elementelor de tip şir. Accesarea tabloului se poate face în două feluri:

- la nivel de element (şir), printr-un singur indice, corespunzător poziţiei şirului, similar cu accesarea unui vector (& 9.2.1);
- la nivel de caracter, prin doi indici, primul indicând poziţia şirului, al doilea poziţia caracterului în cadrul şirului, similar cu accesarea unei matrici (& 9.2.1).

Astfel, programul din fig. 10.1 afişează, prin primul ciclu **FOR**, succesiunea de şiruri accesată printr-un singur indice:

ALFA BETA GAMA DELTA EPSILON

iar prin al doilea ciclu **FOR** se afişează succesiunea de caractere, accesată prin doi indici:

A E M T L

```
CONST NS=10; N=5;
TYPE VECT=ARRAY[1..N] of STRING[NS];
VAR   V: VECT;
      I:INTEGER;
BEGIN
V[1] := 'ALFA';
V[2] := 'BETA';
V[3] := 'GAMA';
V[4] := 'DELTA';
V[5] := 'EPSILON';
FOR I:=1 TO N DO WRITE(V[I],’ ‘ );
WRITELN;
FOR i:=1 TO N DO WRITE(V[I,I],' ');
WRITELN;
END.
```

Fig. 10.1

10.3 FUNCŢII ŞI PROCEDURI ASOCIATE ŞIRURILOR DE CARACTERE

Concatenarea şirurilor se realizează prin funcţia **CONCAT** sau operatorul +, argumentul acestora fiind format din două sau mai multe şiruri, iar rezultatul, de tip **STRING**, obţinut prin alipirea şirurilor argument. Sintaxa operaţiei, exemplificată pentru trei argumente, este:

şir4 = CONCAT (şir1, şir2, şir3) sau **şir4 := şir1 + şir2 + şir3**

unde: **şir1, şir2, şir3** pot fi date de tip **STRING** sau **CHAR**, iar rezultatul **şir4** de tip **STRING**. Depăşirea lungimii maxime admise pentru datele de tip **STRING** (255) generată ca rezultat al concatenării, provoacă trunchierea rezultatului la primele 255 caractere.

Exemplu 4 Concatenarea a trei şiruri generează rezultatul afişat:
sir1:='ALFA' ; sir2:='BETA' ; sir3:='GAMA'
sir1+sir2+ sir3 => ALFABETAGAMA

Exemplu 5 **CONCAT**(‘abc’,’def’,’ghi’) <=> ‘abc’+’def’+’ghi’ => ‘abcdefghi’

Lungimea curentă a unui şir se determină prin funcţia **LENGTH(şir)**; valoarea se poate obţine şi din expresia **ORD(şir[0])**, deoarece în primul octet este memorată lungimea şirului. Argumentul funcţiei este un **şir**, iar rezultatul acesteia este un **număr** întreg, care reprezintă numărul de caractere curent memorat în şir, nu cel declarat în secţiunea declarativă a programului. Sintaxa funcţiei este:

LENGTH (şir)

Exemplu 6 **LENGTH**(‘exemplu’) => 7

Detectarea poziţiei primei apariţii a unui subşir **sir1** *în interiorul unui şir* **şir2** se realizează prin funcţia **POS**, care furnizează ca rezultat un număr întreg, cuprins între 0 şi lungimea şirului: dacă subşirul **sir1** este găsit în şirul **sir2** (identic ca şi caractere, lungime şi dispunere a caracterelor) atunci funcţia oferă ca rezultat un **număr** ce reprezintă poziţia în care se află prima apariţie a şirului căutat, în caz contrar se generează valoarea 0. Argumentele funcţiei sunt două şiruri: primul este subşirul care se caută, iar al doilea este şirul în care se execută căutarea. Sintaxa funcţiei este:

POS (şir1, sir2)

Exemplu 7 **POS**(‘abc’,’defabc’) => 4 ; **POS**(‘abc’,’defabg’) => 0

Deoarece funcţia **POS** returnează prima apariţia a subşirului în interiorul unui şir, următoarele apariţii trebuie căutate într-o copie a şirului iniţial, din care s-a exclus succesiunea de caractere care a inclus prima apariţie, în caz contrar căutarea furnizează acelaşi rezultat.

Extragerea unui subşir dintr-un şir sursă prin funcția **COPY** realizează copierea părții din şirul **şir**, care începe de la poziția **p** pe o lungime de **n** caractere într-o altă variabilă de tip **STRING**. Argumentele funcției sunt: şirul sursă **şir**, poziția (indicele) **p** a caracterului din **şir** de la care începe copierea, numărul de caractere copiate **n**, iar rezultatul acesteia este un şir. Sintaxa funcției este:

COPY (şir, p, n)

Exemplu 8 **COPY**(‘abcdef’,3,2) => ‘cd’ ; **COPY**(‘abcdef’,3,4) => ‘cdef’

Ştergerea unei părți dintr-un şir se realizează prin procedura **DELETE**; efectul acesteia este de ştergere a părții din şirul sursă **şir**, începînd de la poziția **p** pe o lungime de **n** caractere. Deci argumentele acestei funcții sunt şirul sursă **şir**, poziția (indicele) **p** a caracterului din **şir** de la care începe ştergerea, numărul de caractere şterse **n (**lungimea şirului de şters**)**, iar rezultatul acesteia este un şir. Sintaxa procedurii este:

DELETE (şir, p, n)

În urma ştergerii lungimea şirului inițial se reduce cu numărul de caractere şterse **n**. Nu pot fi şterse mai multe caractere decât cele existente între poziția specificată pentru ştergere şi sfârşitul şirului sursă.

Exemplu 9 **DELETE**(‘abcdef’,3,2) => ‘abef’

Inserarea unui subşir într-un şir se realizează prin procedura **INSERT**; efectul acesteia este de inserarea subşirului **s** în şirul **şir** înaintea caracterului de pe poziția **p**. Argumentele procedurii sunt: subşirul ce trebuie inserat **s**, şirul destinație în care se face inserarea **şir**, poziția în şirul **sir** de la care începe inserarea; primii doi parametrii sunt de tip **STRING**, iar ultimul este de tip de tip **INTEGER**; primul şi ultimul parametru pot fi de tip expresii, constante sau variabile. Rezultatul procedurii este un şir.

Prin înserare lungimea şirului creşte, dar fără a putea depăşi lungimea declarată a şirului, în caz contrar ultimele elemente se pierd. Dacă se încearcă inserarea în afara şirului, operația se transformă în concatenare. Sintaxa procedurii este:

INSERT(s, şir, p)

Exemplu 10 **INSERT**(‘12’,‘abcdef’,3) => ‘ab12cdefef’
INSERT(‘12’,‘abcdef’,25) => ‘abcdefef12’

Convertirea unui şir în valoarea numerică întreagă sau reală se realizează prin procedura **VAL,** argumentele acesteia fiind: şirul de caractere **şir** ce trebuie convertit, variabila de tip numeric **număr** care va memora valoarea numerică de tip întreg sau real rezultată în urma conversiei şi un indicator de **eroare**, care primeşte valoarea 0 dacă

conversia s-a executat corect, în caz contrar, valoarea mai mare ca 0, oferă poziţia din şir unde s-a detectat eroarea generată de un caracter nepermis din punct de vedere numeric. Sintaxa procedurii este:

VAL(şir, număr, eroare)

Exemplu 11 Dacă V şi ER sunt de tip întreg:

VAL('12',V,ER) => V=12 şi ER=0 (conversie corectă)

VAL('1g2',V,ER) => V=0 şi ER=2 (conversie incorectă din cauza caracterului 'g' din poziţia a 2-a)

VAL('12.4',V,ER) => V=0 şi ER=3 (conversie incorectă din cauza punctului zecimal din poziţia a 3-a, incompatibil cu tipul **INTEGER**)

Dacă ER este de tip întreg şi V este de tip real:

VAL('12',V,ER) => V=12 şi ER=0 (conversie corectă)

VAL('12.4',V,ER) => V=12.4 şi ER=0 (conversie corectă)

VAL('12.f4',V,ER) => V=0 şi ER=4 (conversie incorectă din cauza caracterului 'f' din poziţia a 4-a).

Conversia unui număr în şirul echivalent se realizează prin procedura **STR;** efectul acesteia este de convertire după formatul **f** a numărului real sau întreg **număr** în şirul de caractere **şir**; formatul **f** este opţional. Argumentele acestei proceduri sunt deci: valoarea numerică **număr** ce poate fi de tip constantă, variabilă sau expresie de tip întreg sau real, urmată opţional de un format de conversie **f** similar cu cel utilizat în cadrul procedurilor **WRITE** sau **WRITELN** (& 5.5.2) şi variabila **şir** în care se memorează numărul convertit în şirul echivalent. Procedura îşi dovedeşte utilitatea la alinierea rezultatelor numerice în tabele.

Sintaxa procedurii este:

STR(număr : f, şir)

Exemplu 12 **STR**(123.4 : 7 : 2, s) => '123.70'

STR(4+9, s) => '13'

10.4 ŞIRURI DE CARACTERE CU TERMINAŢIE NULĂ

Ultimele versiuni de TurboPascal au introdus un nou tip de şir de caractere, denumite **şiruri de caractere cu terminaţie nulă**, care permit generarea de şiruri de caractere ce depăşesc limita de 255 de caractere impuse tipurilor STRING. Pentru aceste noi tipuri limita maximă este de 65535 de caractere. Denumirea acestor şiruri provine din faptul că semnalizarea sfârşitului şirului nu se realizează printr-un număr memorat în primul octet, ci prin caracterul #0 plasat la sfârşitul şirului. Acest tip de date se utilizează sub forma variabilelor de tip pointer, pentru utilizarea lor fiind disponibile proceduri şi funcţii specializate, similare ca denumire şi funcţionalitate celor pentru tipul de date STRING, funcţii depozitate într-o bibliotecă externă, denumită STRING.TPU, care trebuie încărcată explicit înaintea utilizării, prin clauza **USES** plasată imediat după antetul programului.

10.5 APLICAŢII

10.5.1 Căutare numere "cubice"

E 10.1 Program TurboPascal pentru căutare numere "cubice"

Vom relua problema numerelor cubice & 7.4.12, exemplul 7.12 şi vom rezolva aceeaşi problemă utilizând altă tehnică de programare. Vom înţelege prin numere cubice, numerele întregi care satisfac proprietatea că sunt egale cu suma cuburilor cifrelor lor. Exemplu:

$$N = 153 = 1^3 + 5^3 + 3^3 = 1 + 125 + 27$$

Ne propunem căutarea numerelor cubice, aplicând următoarea logică: se parcurg toate numerele **N** în intervalul 100 ÷ 499 şi, prin funcţii de tip şir de caractere, numerele **N** sunt descompuse în cifrele componente: **S** sute, **Z** unităţi şi **U** unităţi; vor fi selectate numai acelea care verifică relaţia de definiţie:

$$N = S^3 + Z^3 + U^3$$

Regula exprima necesitatea egalităţii numerice între numărul calculat în format zecimal cu numărul corespondent rezultat din suma cuburilor cifrelor primului număr.

```
PROGRAM veri_cub;
{Numere intregi cuprinse intre 100-499
egale cu suma cuburilor cifrelor.
Exemplu: 153=1*1*1+5*5*5+3*3*3}
VAR S,Z,U,N,ER: INTEGER;
   S1,Z1,U1:STRING[1];
   NSIR: STRING[3];
BEGIN
FOR N:=100 TO 499 DO
   BEGIN
      STR(N,NSIR); {Conversie numar in sir}
      {Extragere cifre sute, zeci, unitati ca sir}
      S1 := COPY(NSIR, 1, 1); {Sir-Cifra sute}
      VAL(Z1, Z, ER); {Cifra zeci}
IC    Z1 := COPY(NSIR, 2, 1); {Sir-Cifra zeci}
      U1 := COPY(NSIR, 3, 1); {Sir-Cifra unitati}
      {Conversie cifre caracter in cifre numerice }
      VAL(S1, S, ER); {Cifra sute}
      VAL(U1, U, ER); {Cifra unitati}
      IF N=S*S*S+Z*Z*Z+U*U*U THEN WRITELN(N);
   END;
READLN
END.
```

Fig. 10.2

Prin intermediul ciclului **FOR**, în variabila contor **N** sunt generate numerele între 100 şi 499, pentru fiecare număr se execută:

- numărul **N** se converteşte în şirul **NSIR** prin funcţia **STR**;
- din şirul **NSIR** sunt extrase, prin funcţia **COPY,** cifrele sutelor, zecilor şi unităţilor, sub formă de caractere individuale, în variabilele **S1**, **Z1**, **U1**;
- caracterele echivalente sutelor, zecilor şi unităţilor **S1**, **Z1**, **U1** sunt transformate prin funcţia **VAL** în cifrele numerice echivalente, în variabilele **S**, **Z**, **U**;
- condiţia din instrucţiunea **IF** verifică relaţia de definire a numerelor cubice, la îndeplinirea acesteia numărul **N** fiind afişat.

```
153
370
371
407
```

Fig. 10.3

10.5.2 Căutare numere "automorfice"

E 10.2 Program TurboPascal pentru căutare numere "automorfice"

Fie "**i**" un număr natural compus din "**k**" cifre. Se defineşte "**i**" ca fiind automorfic, dacă ultimele "**k**" cifre ale pătratului lui "**i**" formează un număr egal cu "**i**". Exemplu: **i=625 ; k=3 ; i*i=390625** Ne propunem căutarea numerelor automorfice, în domeniul 1 – 1000, aplicând logica exprimată de însăşi definiţia lor.

Fig. 10.4

```
PROGRAM automorfic;
{Fie numere "i" de "k" cifre, intre 1-1000 ;
numerele automorfice sunt acelea pentru care
ultimele "k"  ale patratului lui "i" formeaza
un numar egal cu "i". Exemplu:
   i=625   k=3    i*i=390625}
VAR k, kp: INTEGER;
   i, patrat_i: LONGINT;
   isir, patrat_sir, ultime_k:STRING[20];
BEGIN
WRITELN('Numere automorfice sunt:');
FOR i:=1 TO 1000 DO
  BEGIN
    STR(i,isir); {Conversie numar in sir}
    k := LENGTH(isir); {Lungime isir}
    patrat_i := i*i; {Calcul patrat numar}
    STR(patrat_i, patrat_sir); {Conversie patrat numar in sir}
    kp := LENGTH(patrat_sir); {Lungime patrat_sir}
    ultime_k := COPY(patrat_sir, kp-k+1,kp); {Ultime k caractere}
    IF isir =  ultime_k THEN
      WRITELN('i=',i:4,'  k=',k:2,  '  kp=',kp:2,'  i*i= ', patrat_sir);
  END;
READLN
END.
```

IC

Variabilele "**i**", "**patrat_i**" sunt declarate de tip **LONGINT**, deoarece valorile rezultate din pătratul numărului "**i**" depăşesc domeniul admis pentru variabilele de tip **INTEGER**, & 4.2.1, tabel 4.3.

Variabilele "**k**" şi "**kp**" vor memora lungimile şirurilor de caractere echivalente numărului "**i**" respectiv pătratului acestuia.

```
Numere automorfice sunt:
i=    1  k= 1  kp= 1  i*i= 1
i=    5  k= 1  kp= 2  i*i= 25
i=    6  k= 1  kp= 2  i*i= 36
i=   25  k= 2  kp= 3  i*i= 625
i=   76  k= 2  kp= 4  i*i= 5776
i=  376  k= 3  kp= 6  i*i= 141376
i=  625  k= 3  kp= 6  i*i= 390625
```

Fig. 10.5

Prin intermediul ciclului **FOR**, în variabila contor "**i**" sunt generate numerele între 1 şi 1000; pentru fiecare număr se execută:

- conversia număr "**i**" în şirul echivalent "**isir**", prin funcţia **STR**;
- memorarea în variabila "**k**" a lungimii şirului "**isir**", prin funcţia **LENGTH**;
- calculul pătratului lui "**i**" în variabila "**patrat_i**" şi conversia în şirul echivalent "**patrat_sir**", prin funcţia **STR**;
- memorarea în variabila "**kp**" a lungimii şirului "**patrat_sir**", prin funcţia **LENGTH**;
- extragerea ultimelor "**k**" caractere din şirul "**patrat_sir**" în şirul **"ultime_k**" prin funcţia **COPY**;
- testarea (ca şir) a ultimelor "**k**" caractere ale pătratului cu şirul echivalent numărului "**i**" şi afişarea **WRITELN** în caz de egalitate.

10.5.3 Construire şir echivalent unui şir dat

10.3 Program TurboPascal pentru construire şir echivalent unui şir dat

Se dă un şir de caractere. Să se construiască un şir echivalent care, pentru fiecare caracter, să fie format din caracterul '**1**', dacă - în şirul iniţial - caracterul echivalent este o cifră şi caracterul '**0**' în caz contrar. Exemplu: **abc3def4fg5hj6 => 00010001001001.**

Fig. 10.6

```
PROGRAM sir_echivalent;
VAR i, cifra: INTEGER;
   sir_a, sir_b, cifra_sir :STRING[50];
   gasit:BOOLEAN;
BEGIN
WRITELN('Dati un sir [max. 50 caractere ]');
READLN(sir_a);
FOR i:=1 TO LENGTH(sir_a) DO
   BEGIN
      • gasit:=FALSE;
      • FOR cifra := 0 TO 9 DO
         BEGIN
            STR(cifra,cifra_sir);
            IF sir_a[i]=cifra_sir THEN
               BEGIN
                  gasit:=TRUE;
                  BREAK
               END;
         END;
      • IF gasit=TRUE THEN
            sir_b := sir_b+'1'
         ELSE
            sir_b := sir_b+'0';
   END;
WRITELN(sir_b);
READLN
END.
```

IC, IC, IC

```
Dati un sir [max. 50 caractere ]
abc3def4fg5hj6
00010001001001
```

Fig. 10.7

Şirul de caractere se introduce de la tastatură în variabila "**sir_a**". Fiecare caracter din şirul iniţial "**sir_a**" (ciclul exterior de contor "**i**") este comparat cu toate cifrele caracter de la 0 la 9 (ciclul interior de contor "**cifra"**), iar, în caz de coincidenţă, la şirul "**sir_b**" se concatenează caracterul '**1**', în caz contrar caracterul '**0**'. Ciclu **FOR**, de contor "**i**" are numărul de repetiţii egal cu numărul de caractere al şirul "**sir_a**", executându-se instrucţiunile marcate cu simbolul • :

- iniţializarea variabilei logice "**gasit**" cu valoarea **FALSE**;
- execuţia ciclului interior de contor "**cifra**" (generând cifrele între 0-9) ce include în instrucţiunea compusă două instrucţiuni:

1. conversia cifrei din număr în caracterul echivalent "**cifra_sir**" , prin funcţia **STR**;
2. testarea identităţii caracterului din poziţia "**i**" a şirului iniţial **sir_a[i]** cu "**cifra şir**"; pentru identitate variabila "**găsit**" devine **TRUE** şi se iese forţat din ciclul interior prin **BREAK** (& 7.5.1) la ultima instructiune **IF**.

- logica instrucţiunii **IF** finale este: dacă coincidenţa nu a fost găsită pentru nici o cifră din domeniu de la 0 la 9, variabila "**gasit**" ramâne la valoarea **FALSE** şi deci la şirul "**sir_b**" se concatenează caracterul '**0**'; în caz contrar variabila semafor "**găsit**" primeşte valoarea **TRUE**, se va executa ramura **THEN**, deci la şirul "**sir_b**" se concatenează caracterul '**1**';
- în final se afişează şirul rezultat "**sir_b**", fig. 10_7.

10.5.4 Despărţire propoziţie în cuvinte

E 10.4 Program TurboPascal pentru despărţire propoziţie în cuvinte

Se dă o propoziţie. Să se descompună propoziţia în cuvintele componente, utilizând caracterul spaţiu ca şi separator al cuvintelor.

```
PROGRAM despartire_cuvinte;
{Despartire propozitie in cuvinte}
VAR s:STRING;
   cuvinte:ARRAY[1..50] OF STRING;
   i, p, nrcuv:INTEGER;
BEGIN
   WRITELN('Introduceti sirul');
   READLN(s);
   s:=s+' ';
   nrcuv:=0;
   WHILE s<>'' DO
      BEGIN
         p:=POS(' ',s);
         INC(nrcuv);
         cuvinte[nrcuv]:=COPY(s,1,p-1);
         DELETE(s,1,p);
      END;
   {Afisare cuvinte}
   FOR i:=1 TO nrcuv DO WRITELN(cuvinte[i]);
   READLN
END.
```

Fig. 10.8

```
Introduceti sirul
Acesta este un test.
Acesta
este
un
test.
```

Fig. 10.9

IC

Fie o propoziţie formată din cuvinte (şiruri de caractere) despărţite prin spaţii. Se determină în şirul **S** poziţia **P** a primei apariţii a caracterului *spaţiu*, se incrementează variabila **NRCUV**, iar prin funcţia **COPY** se preia cuvântul din şirul **S** de la poziţia **1** la **P-1** şi se introduce în poziţia **NRCUV** a tabloului **CUVINTE**. Apoi se şterge partea din sirul **S** de la poziţia **1** la **P**. Algoritmul se repetă atâta timp cât şirul **S** este diferit de şirul nul.

Concatenarea şirului "**s**" cu caracterul final *spaţiu*, prin instrucţiunea de atribuire **s := s + ' '** asigură finalizarea ciclului **WHILE**; şirul "**s**" este permanent micşorat prin ştergerea caracterelor cuprinse între poziţia 1-a şi până la primul caracter spaţiu; în lipsa instrucţiunii de atribuire şi în lipsa unui spaţiu plasat la sfârşitul şirului, ultimul cuvânt nu va fi şters, şirul "**s**" nu va deveni niciodată vid, iar ciclul devine infinit.

Programul este limitat la memorarea a 50 de cuvinte, datorită dimensionării tabloului "**cuvinte**" la această valoare.

11. SUBPROGRAME

11.1 INTRODUCERE

De multe ori este necesar ca o secvență de instrucțiuni să se execute de mai multe ori în cadrul aceluiaşi program, eventual chiar şi pentru alte date de intrare. Aceste instrucțiuni se pot grupa într-un modul unitar, denumit **subprogram**, cu nume definit.

Programele TurboPascal sunt constituite din secțiunea declarațiilor şi secțiunea instrucțiunilor, & 4.4, care este o instrucțiune compusă încadrată de cuvintele cheie **BEGIN-END**. Aceste două părți formează un **bloc**.

Şi subprogramele formează tot un bloc, deoarece sunt constituite pe aceeaşi structură: secțiunea de declarații şi o instrucțiune compusă. Un bloc poate conține la rândul său alte blocuri (module), adică acestea pot fi îmbricate (incluse unele în altele).

Un program constituit din mai multe blocuri se numeşte **modularizat**.

În limbajul TurboPascal există două tipuri de proceduri:

- **proceduri** - sunt subprograme care produc nici unul, unul sau mai multe rezultate;
- **funcțiile** – sunt subprograme similare structural ca procedurile, dar care se deosebesc de acestea prin modul de apelare şi prin faptul că generează ca rezultat o singură valoare, similar funcțiilor standard ale limbajului.

Programul de bază al aplicației se numeşte **program principal**. Programul sau subprogramul care apelează un alt subprogram se numeşte **program apelant**, iar subprogramul activat se numeşte **program apelat**. În construcția programelor, un subprogram poate fi de tip apelant, dacă apelează la rândul său alte proceduri, sau poate fi program apelat, dacă este apelat de către un alt program. Astfel se creează o structură ierarhică de apeluri.

La apelarea unei proceduri dintr-un program apelant, execuția programului continuă cu prima instrucțiune din procedură, iar la finalizarea acesteia se revine în programul apelant şi execuția continuă cu prima instrucțiune de după apel, fig. 11.1. Procedurile şi funcțiile se pot apela între ele, singura restricție fiind ca, în momentul apelului, procedura apelată să fi fost definită. Un subprogram se defineşte în întregime în partea declarativă a programului apelant, deci înainte de secțiunea instrucțiunilor acestuia, sub formă de bloc, fig. 11.2.

De asemenea un subprogram se poate autoapela, tehnica fiind denumită *recursivitate*.

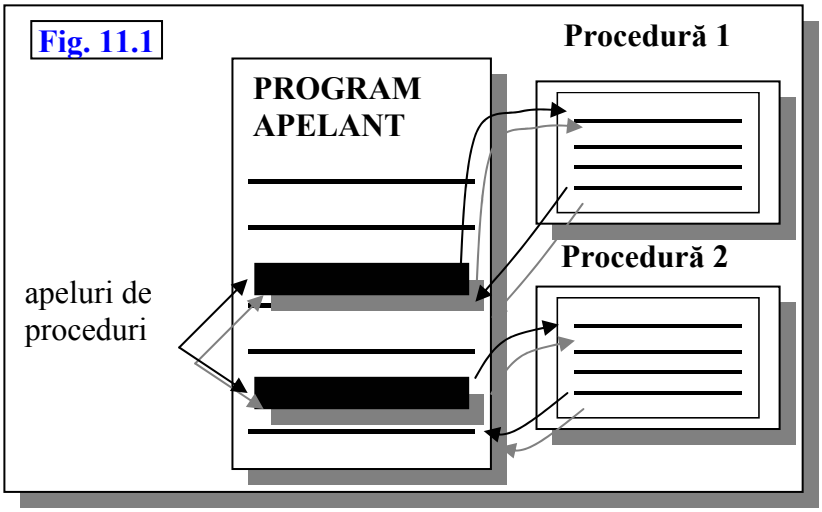


Fig. 11.1

```
antet de procedură;
declarații proprii procedură;
BEGIN
      instrucțiuni;
END;
```

Fig. 11.2

11.2 DEZVOLTAREA ASCENDENTĂ ŞI DESCENDENTĂ A PROGRAMELOR

Scrierea modularizată a programelor oferă două posibilităţi de dezvoltare a acestora, inclusiv posibilitatea combinării acestora: dezvoltarea ascendentă şi descendentă.

Dezvoltare **ASCENDENTĂ** – procedurile se declară succesiv în partea declarativă a programului, unul după altul; caracteristicile acestei dezvoltări sunt următoarele, fig. 11.3:

- din blocul principal se pot apela toate procedurile;
- din fiecare procedură "**i**" pot fi apelate toate procedurile anterior definite (de la "**1**" la "**i-1**"), inclusiv pe ea însăşi (recursivitate).

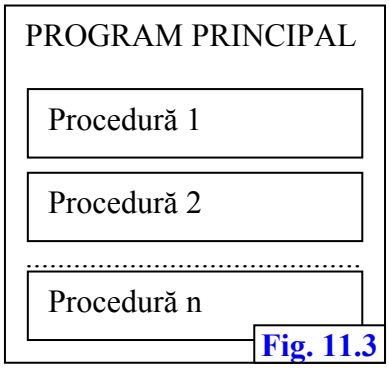


Fig. 11.3

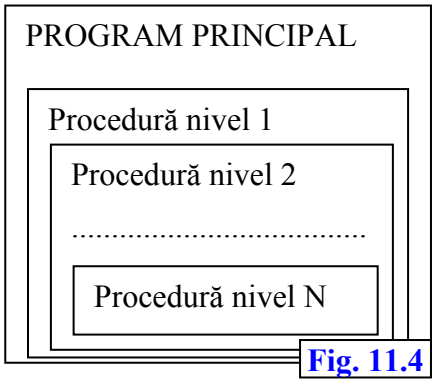


Fig. 11.4

Dezvoltare **DESCENDENTĂ** – procedurile se declară prin includere unele în altele; caracteristicile acestei dezvoltări sunt următoarele, fig. 11.4:

- din blocul principal se pot apela numai procedurile de pe nivelul imediat următor;
- din fiecare procedură pot fi apelate numai procedurile nivelului imediat următor (nu există posibilitate de recursivitate).

11.3 DOMENIUL DE VIZIBILITATE AL IDENTIFICATORILOR

Procedurile TurboPascal sunt bazate pe conceptul de bloc, format din secţiunea declaraţiilor şi instrucţiunea compusă (încadrată de cuvintele cheie **BEGIN-END**). Din punct de vedere al identificatorilor, blocul posedă proprietatea localizării identificatorilor; prin aceasta, toţi identificatorii definiţi în bloc sunt **locali** blocului şi sunt cunoscuţi datorită declaraţiilor din interiorul blocului. Aceşti identificatori apar la intrarea în bloc datorită declaraţiilor şi dispar la ieşirea din bloc, procesul de apariţie şi dispariţie fiind raportat la memorie. Dar un bloc poate include alte blocuri, situaţie în care se pune problema definirii domeniului de vizibilitate al unui identificator (care poate fi tip de date, variabilă, constantă, funcţie sau procedură) declarat într-un bloc în raport cu alte blocuri.

Prin *domeniul de vizibilitate al unui identificator* se înţelege totalitatea blocurilor în care este recunoscut.

În TurboPascal, această problemă se rezolvă astfel: *identificatorul va fi recunoscut atât în blocul în care a fost declarat, cât şi în toate blocurile conţinute în acesta, cu condiţia ca respectivele blocuri să fi fost definite după declararea identificatorului*. Pentru toate aceste blocuri, identificatorul este **global**, în rest fiind **necunoscut**.

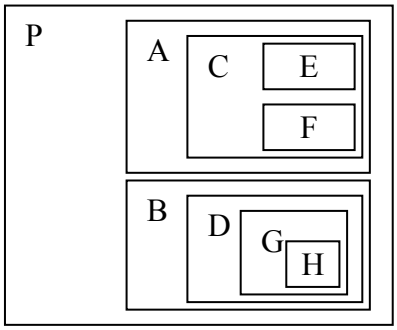
În situaţia redeclarării în interiorul unui bloc a unui identificator global, atunci această definiţie va avea prioritate faţă de cea anterioară.

Se pot enunța următoarele *reguli de vizibilitate a identificatorilor*:

- în cadrul aceluiaşi bloc, un identificator poate fi definit o singură dată;
- un acelaşi identificator poate fi definit în blocuri diferite, situație în care natura acestuia este definită în cadrul celui mai interior bloc ce conține atât instrucțiunea în care este utilizat cât şi declarația lui;
- un identificator nu poate fi folosit în exteriorul blocului în care a fost declarat.

Pentru programul modularizat din fig. 11.5, tabelul 1 prezintă ierarhizarea blocurilor pe nivele, iar tabelul 2 domeniul de valabilitate al identificatorilor. Variabila declarată în blocurile B şi G şi apelată în blocul H va avea tipul precizat în declarația din blocul G.

Fig. 11.5



Tabel 1

Bloc	Nivel
P	0
A, B	1
C, D	2
E, F, G	3
H	4

Tabel 2

Identificatorii declarați în blocul	sunt accesibili în blocul
P	P, A, B, C,D,E, F, G, H
A	A, C, E, F
B	B, G, D, H
C	C, E, F
D	D, G, H
E	E
F	F
G	G, H
H	H

O procedură poate avea propria sa parte declarativă, în care se pot defini constante, tipuri, variabile şi chiar alte funcții, toate acestea fiind văzute numai în interiorul acesteia. Identificatorii definiți în interiorul blocurilor sunt **identificatori locali**. Variabilele globale şi cele locale pot avea acelaşi nume, fără a avea influență asupra valorilor, însă semnificațiile vor fi diferite în cele două blocuri. În interiorul procedurilor se recomandă utilizarea cu predilecție a variabilelor locale şi nu a celor globale, deoarece astfel scade gradul de interacțiune cu programul principal şi se minimizează riscul apariției erorilor.

11.4 PROCEDURI

Procedurile sunt subprograme care produc nici unul, unul sau mai multe rezultate. Descrierea unei proceduri trebuie să conțină definirea identificatorilor interiori procedurii precum şi grupul de instrucțiuni executive asociate acesteia, toate fiind grupate sub un nume unic, nume prin care ea va fi apelată.

Definirea unei proceduri se face în următoarea succesiune, fig. 11.6:

- **Antetul** procedurii cuprinde:
  - cuvântul cheie **PROCEDURE** urmat de *identificatorul* procedurii **id_proc** (denumirea acesteia);
  - *lista de parametrii (argumente) formali*, compusă din identificatori **V1, V2, ..., Vn, W2, ..., Wn, Z1, Z2, ..., Zn, U1, U2, ..., Un**, pentru fiecare grup de identificatori fiind precizat tipul de date asociat, respectiv **T1, T2, T3, T4**; parametrii listei se despart prin separatorul “**,**”, grupul de identificatori se despart de declarația tipului de date prin

separatorul "**:**", iar separatorul "**;**" desparte grupuri de parametrii de acelaşi tip; lista de parametrii formali conține două categorii:

- ✓ *parametrii constantă (valoare)* - parametrii ai căror valori se pot modifica în interiorul procedurii, dar rămân nemodificați în exteriorul ei; aceştia se folosesc pentru transmiterea datelor de către programul apelant spre procedură;
- ✓ *parametrii variabilă (referință)* - parametrii care sunt precedați de cuvântul cheie **VAR** şi a căror modificare de valoare este transmisă şi programului apelant; aceştia se pot folosi atât pentru transmiterea datelor de către programul apelant spre procedură, cât şi pentru returnarea de valori programului apelant;

- **Partea declarațiilor subprogramului** poate conține, ca şi blocul programului principal, orice declarație de tip, constante, variabile sau alte subprograme de nivel superior. Declarațiile din interiorul unei proceduri au un caracter local. Nu pot fi declarate variabile care apar în lista de parametrii formali. Se pot redeclara însă, cu acelaşi nume, variabile care apar în blocul apelant, situație în care, conform regulilor de vizibilitate a identificatorilor nu mai putem accesa variabilele respective din blocul apelant.
- **Corpul procedurii** este o instrucțiune compusă, ce începe cu **BEGIN** se termină cu END, urmat de separatorul ";" spre deosebire de programul principal, ce se termină cu ".".

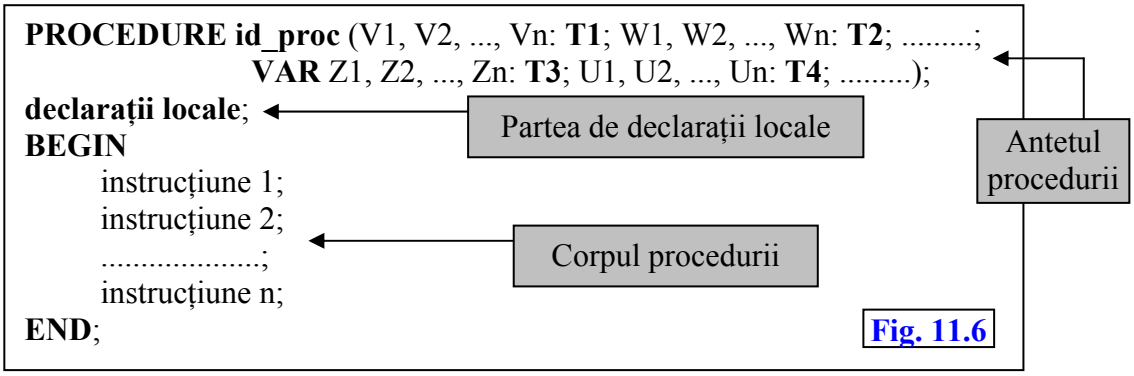


Fig. 11.6

Apelul procedurii se va realiza prin numele acesteia urmat de lista parametrilor actuali:

id_proc(lista de parametrii actuali).

Comunicarea între un bloc apelant şi un bloc apelat se poate realiza:

- în ambele sensuri, prin parametrii variabili (precedați de cuvântul cheie **VAR**);
- doar de la blocul apelant spre cel apelat, prin intermediul celorlalți parametrii.

Observații:

1. O procedură are structura asemănătoare cu a unui program (& 4.4), singura diferență fiind antetul;
2. Într-o declarație de procedură sunt obligatorii minimal numai antetul de procedură şi instrucțiunea compusă **BEGIN-END**;
3. Orice identificatori utilizat în interiorul procedurii trebuie declarat, fie în zona de declarații a procedurii (când are o valoare locală valabilă numai în interiorul acesteia), fie în blocul în care este inclusă procedura (caz în care are o valoare globală, valabilă în blocul respectiv, dar numai dacă nu este redeclarat în interiorul procedurii);
4. Dacă un identificator este declarat sub acelaşi nume şi în procedură şi în blocul în care este inclusă procedura, atunci semnificațiile lui vor fi diferite în cele două module.

11.5 FUNCŢII

Cu toate că limbajul pune la dispoziţia utilizatorului un set de funcţii standard, există situaţii în care sunt necesare funcţii inexistente în acest set. Funcţiile sunt subprograme care generează ca rezultat o singură valoare, similar funcţiilor standard ale limbajului.

Definirea unei funcţii se face în următoarea succesiune, fig. 11.7:

- **Antetul** funcţiei cuprinde, pe lângă caracteristicile enunţate la proceduri, cuvântul cheie **FUNCTION** şi caracteristica **tip_funcţie**, prin care se defineşte tipul rezultatului returnat de funcţie. Aceasta poate fi de tip simplu, string sau referinţă.
- **Partea declaraţiilor funcţiei** poate conţine, ca şi la proceduri, orice declaraţie de tip, constante, variabile sau alte subprograme de nivel superior.
- **Corpul funcţiei** este o instrucţiune compusă, ce începe cu **BEGIN** se termină cu **END**, urmat de separatorul ";" spre deosebire de programul principal, ce se termină cu ".". Suplimentar faţă de proceduri, corpul funcţiei trebuie să conţină, uzual la sfârşit, o instrucţiune de atribuire de forma **id_fun := expresie**, unde expresia trebuie să furnizeze un rezultat de tip **tip_funcţie**. În urma execuţiei funcţiei, rezultatul calculat al acesteia este atribuit numelui funcţiei, prin intermediul căruia se va realiza returnarea rezultatului către programul apelant. Acesta este motivul pentru care, în secţiunea de instrucţiuni a funcţiei, numele acesteia trebuie să figureze cel puţin odată în stânga unei instrucţiuni de atribuire.

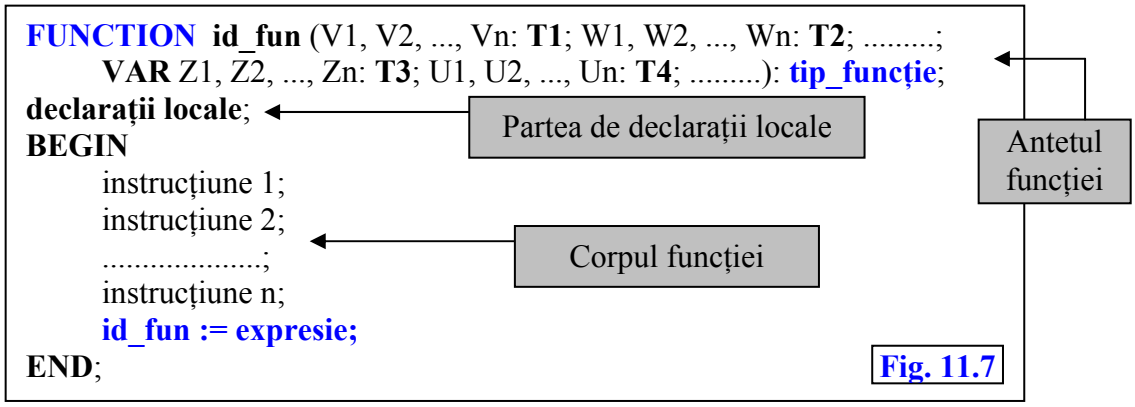


Fig. 11.7

Observaţii:

1. Lista parametrilor formali are sintaxa şi semnificaţia identică cu cea a procedurilor, cu diferenţa că toţi parametrii sunt numai de intrare, deci primesc valori de la programul apelant, dar nu returnează valori către acesta. Teoretic, este posibil ca funcţia să nu aibe parametrii formali, situaţie în care variabile globale pot fi utilizate pentru comunicarea de date cu programul apelant.
2. Apelarea unei funcţii nu este o instrucţiune de sine stătătoare, ea trebuie inclusă ca operand în cadrul expresiilor din programul apelant.
3. Rezultatul funcţiei este reprezentat de o singură valoare (deci nu poate fi de tip structurat, de exemplu tablou).
4. Numele funcţiei nu poate fi utilizat - în interiorul blocului care defineşte funcţia – în membrul drept al unei instrucţiuni de atribuire, deoarece acesta, nefiind o variabilă, serveşte numai la returnarea valorii funcţiei.

11.6 COMUNICAREA ÎNTRE PROGRAMUL APELANT ŞI SUBPROGRAME

În acest capitol, termenul de subprogram se va referi atât la proceduri cât şi la funcţii. Subprogramele sunt secvenţe de instrucţiuni scrise o singură dată şi apelate de mai multe ori, de către diverse programe apelante şi cu diferite date de intrare. De asemenea subprogramele trebuie să returneze valorile calculate către programele apelante. Această tehnică este mult mai eficientă decât rescrierea aceloraşi instrucţiuni de mai multe ori, iar avantajele sunt evidente: reducerea numărului de instrucţiuni ale programului, evitarea erorilor care pot apare la rescrierea şi modificarea aceloraşi instrucţiuni lor la execuţia pentru alte date de intrare, creşterea clarităţii programului.

Execuţia unui subprogram se face prin apelul acestuia dintr-un program apelant, după următoarea succesiune:

- programul apelant conţine o instrucţiune de apel de subprogram, la întâlnirea căreia işi suspendă execuţia şi activează execuţia subprogramului;
- în continuare, subprogramul execută instrucţiunile proprii, până la finalizarea completă a acestora (întâlnirea instrucţiunii **END;**);
- controlul este returnat programului apelant, la instrucţiunea imediat următoare apelului de subprogram.

Desigur însă că programarea prin utilizarea subprogramelor impune existenţa de mecanisme de comunicare între programul apelant şi subprogram. Sunt posibile două modalităţi de comunicare:

- comunicarea prin variabile globale;
- comunicarea prin mecanismul corespondenţei parametrilor actuali cu parametrii formali.

În cele ce urmează vor fi analizate cele două variante.

11.6.1 Comunicarea prin variabile globale

Declaraţia identificatorilor şi etichetelor este valabilă în interiorul blocului în care este plasată declaraţia, inclusiv în toate blocurile subordonate. În particular, identificatorii şi etichetele din programul principal pot fi utilizaţi în toate blocurile asociate acestuia.

Deci, un **identificator global** în raport cu un subprogram de referinţă trebuie definit în mod necesar în exteriorul acestuia, într-un alt program sau subprogram, care îl include, dar fără a fi redeclarat în interiorul subprogramului de referinţă. Identificatorul este deci cunoscut în subprogramul de referinţă, cu definiţia şi semnificaţia declarate în exteriorul acestuia, constituind astfel mijlocul de comunicare între programul apelant şi subprogram. De asemenea, este posibilă şi comunicarea în sens invers: modificarea valorilor unui identificator în interiorul subprogramului este cunoscută şi în exteriorul acestuia (în domeniul de vizibilitate al acestuia, & 11.3), constituind astfel mijlocul de comunicare dintre subprogram spre programul apelant.

Pentru această variantă de comunicare antetul subprogramului poate să nu conţină parametrii, deoarece comunicarea cu programul apelant nu se realizează prin parametrii din antet, ci prin variabile globale.

Deşi această variantă de comunicare este disponibilă în TurboPascal, nu se recomandă utilizarea ei, mecanismul corespondenţei parametrilor actuali cu parametrii formali fiind o variantă mult mai eficientă.

E 11.1 Program TurboPascal pentru calculul unei suprafețe dreptunghiulare prin însumarea ariilor dreptunghiurilor elementare

Vom exemplifica tehnica de comunicare prin variabile globale, în baza unui exemplu didactic, care-şi propune calculul suprafeței dreptunghiulare 120 x 150, prin sumarea ariilor dreptunghiurilor **1** (40x70), **2** (80 x70) şi **3** (120 x 80), fig. 11.8.

```
PROGRAM arie_dreptunghi;
{Calcul arie dreptunghi ca suma a
ariilor mai multor dreptunghiuri
elementare, utilizand o procedura ce
comunica cu programul apelant prin
variabile globale}

{Declaratii de variabile globale}
VAR L,H,s1,s2,s3,s,stotal: INTEGER;

{Definire procedura}
PROCEDURE aria;
BEGIN
s:= H * L
END;

{Program principal}
BEGIN
{Calcul arie triunghi 1}
H := 40 ; L := 70;
aria; s1 := s;
WRITELN('Arie 1=',s1:6)
{Calcul arie triunghi 2};
H := 80 ; L := 70;
aria; s2 := s;
WRITELN('Arie 2=',s2:6);
{Calcul arie triunghi 3}
H := 120 ; L := 80;
aria; s3 := s;
WRITELN('Arie 3=',s3:6);
{Calcul si afisare arie totala}
stotal := s1 + s2 + s3;
WRITELN('Aria totala =',stotal:6);
READLN
END.
```

Fig. 11.9

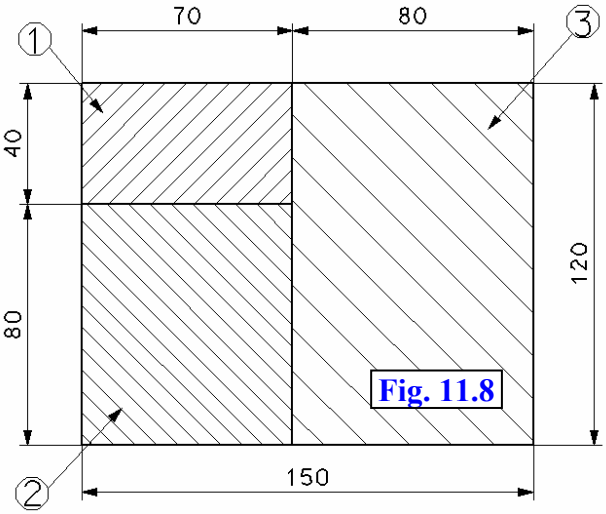


Fig. 11.8

```
Arie 1=  2800
Arie 2=  5600
Arie 3=  9600
Aria totala = 18000
```

Fig. 11.10

Programul principal **arie_dreptunghi** conține în secțiunea declarațiilor, declarațiile variabilelor **L**, **H**, **s1**, **s2**, **s3**, **s**, **stotal** ca fiind de tip întreg, precum şi procedura **aria**. În raport cu procedura **aria**, toate variabilele sunt *globale*, fiind definite în exteriorul acesteia. Procedura calculează aria unui dreptunghi **s** (dată de ieşire din procedură), prin produsul dintre variabilele **H** şi **L** (date de intrare în procedură). Se observă lipsa parametrilor formali din antetul procedurii, comunicarea cu programul apelant se realizează deci prin variabilele globale **H**, **L** şi **s**. Apelul procedurii se realizează prin intermediul numelui ei, adică **aria**. Se observă deci că, procedura este apelată în

programul principal de 3 ori, anterior fiecărui apel valorile variabilelor **H** şi **L** fiind actualizate prin atribuire la valorile corespunzătoare dreptunghiului a cărui arie se doreşte a se calcula, De asemenea, după apelul de procedură, valoarea calculată de aceasta în variabila **s** este preluată succesiv, prin atribuire, în variabilele **s1**, **s2**, **s3**. Aria finală **stotal** rezultă prin suma ariilor celor 3 dreptunghiuri **s1**, **s2** şi **s3**. Vom detalia procesul execuţiei programului principal:

- valorile înălţimii şi lăţimii primului dreptunghi 40 respectiv 70 sunt memorate în variabilele globale **H** respectiv **L**;
- prin intermediul numelui procedurii, adică **aria**, controlul execuţiei programului principal este temporar suspendat şi execuţia este transferată procedurii **aria**;
- cu valorile curente ale lui **H** şi **L**, procedura calculează aria şi depune rezultatul în variabila **s**, după care transferă execuţia din nou către programul principal;
- în programul principal execuţia continuă cu prima instrucţiune după primul apel al procedurii, adică atribuirea **s1 := s**, prin care valoarea ariei **s** este memorată în variabila **s1**, deoarece variabila **s** va fi refolosită la următoarele apeluri ale procedurii **aria**;
- procedeul se repetă încă de 2 ori, cu deosebirea că valorile variabilelor globale (înălţimea **H** şi lăţimea **L** ale dreptunghiurilor) se modifică de fiecare dată anterior apelului procedurii; de asemenea după fiecare apel de procedură, valoarea calculată a ariei este depusă în variabila corespunzătoare, adică **s2** respectiv **s3**.

11.6.2 Comunicarea prin mecanismul corespondenţei parametrilor actuali cu parametrii formali

Parametrii specificaţi în listele din instrucţiunile de apel a subprogramelor se numesc **parametrii actuali** şi pot fi variabile, expresii sau valori.

Parametrii specificaţi în listele de parametrii din antetul subprogramelor se numesc **parametrii formali** şi pot fi identificatori de variabile locale care pot fi accesate în interiorul subprogramului.

Mecanismul corespondenţei dintre parametrii actuali cu parametrii formali impune o corespondenţă directă şi biunivocă (în sensul transmiterii de valori) între *parametrii actuali* (identificatori şi expresii cu care se apelează procedura din cadrul blocului apelant) cu *parametrii formali* ai subprogramului (identificatori din antetul procedurii), în *ordinea definirii* lor de la stânga la dreapta, astfel încât *numărul, tipul de date* şi *ordinea* parametrilor actuali trebuie să corespundă identic cu cel al parametrilor formali. Corespondenţa se realizează deci între parametrii actuali şi formali din aceeaşi poziţie în cele două liste.

Parametrii formali pot fi de 2 categorii:

- *parametrii constantă (valoare)* - parametrii ai căror valori se pot modifica în interiorul procedurii, dar rămân nemodificaţi în exteriorul ei; aceştia se folosesc pentru transmiterea datelor din programul apelant spre procedură, dar nu şi invers, şi se comportă ca o variabilă locală subprogramului; la apelul procedurii li se alocă spaţiu de memorie, primesc valoarea parametrului actual corespondent, apoi procedura poate modifica valoarea lui fără ca această modificare să altereze şi valoarea parametrului actual corespondent.
- *parametrii variabilă (referinţă)* - parametrii care sunt precedaţi de cuvântul cheie **VAR** şi a căror modificare de valoare este transmisă programului apelant; aceştia se pot

folosi atât pentru transmiterea datelor de către programul apelant spre procedură, cât şi pentru returnarea de valori programului apelant; din acest motiv parametrul actual corespunzător trebuie să fie în mod necesar o variabilă, neputând fi o expresie.

Regula generală este dată de caracteristica de intrare / ieşire a parametrului în raport cu subprogramul: în general datele de intrare (care nu trebuie să sufere modificări în interiorul subprogramului) se vor declara ca parametrii formali constantă, iar datele de ieşire (rezultatele subprogramului) se vor declara ca parametrii formali variabilă.

Corespondenţa dintre parametrii actuali şi parametrii formali constantă se execută astfel: la apelul subprogramului se alocă spaţiu de memorie pentru parametrul formal, se evaluează expresia parametrului actual, iar valoarea calculată a acestuia este depozitată în spaţiul creat ca şi valoare curentă a parametrului formal. Acest parametru formal primeşte statut de variabilă locală, deci orice modificare de valoare efectuată în interiorul subprogramului nu va avea efecte în exteriorul său. Ca o consecinţă, valoarea dobândită de parametrul formal se va pierde la ieşirea din subprogram.

Corespondenţa dintre parametrii actuali şi parametrii formali variabilă constă în asocierea locaţiei de memorie a parametrului actual cu a parametrului formal variabilă, astfel încât, pe durata execuţiei subprogramului, orice referire la parametrul formal acţionează de fapt în mod direct asupra parametrului actual corespondent. Prin aceasta modificarea de valoare se transmite şi în afara subprogramului.

OBSERVAŢII:

- Apelul prin valoare necesită mai multă memorie, prezintă însă avantajul că se pot transmite procedurii şi valori ale unor expresii.
- Dacă parametrii actuali sunt tipuri structurate de mari dimensiuni, un spaţiu de memorie de aceleaşi dimensiuni este consumat la transmiterea către parametrii formali de tip valoare, iar copierea valorilor consumă de asemenea timp de execuţie. În aceste situaţii este mai preferabilă utilizarea parametrilor formali variabilă, dar numai dacă nu se fac atribuiri asupra parametrilor formali, care pot modifica nedorit valorile acestora;
- Lista parametrilor formali poate lipsi, ceea ce presupune şi absenţa parantezelor; în această situaţie procedura va fi apelată fără parametrii actuali, ex. **READLN**; **WRITELN**; în această situaţie schimbul de informaţii cu programul apelant nu există sau se realizează prin variabile globale.
- Parametrii formali pot fi de orice tip, dar dacă sunt de tip structurat, aceştia trebuie definiţi anterior. Exemplu, nu putem avea definiţia **PROCEDURE** test (X : **ARRAY** ....), dar dacă se defineşte anterior **TYPE** vector=**ARRAY** ...., se poate defini **PROCEDURE** test (X : vector).

E 11.2 Program TurboPascal pentru calculul unei suprafeţe dreptunghiulare prin însumarea ariilor dreptunghiurilor elementare

Vom relua exemplul din fig. fig. 11.8 pentru a exemplifica tehnica de comunicare dintre programul apelant şi subprogram prin mecanismul corespondenţei parametrilor actuali cu parametrii formali, comparativ cu tehnica de comunicare prin intermediul variabilelor globale.

```
PROGRAM arie_dreptunghi;
{Calcul arie dreptunghi ca suma a
ariilor mai multor dreptunghiuri
elementare, utilizand o procedura ce
comunica cu programul apelant prin
liste de parametrii}

{Declaratii de variabile globale}
VAR s1,s2,s3,stotal: INTEGER;

{Definire procedura}
PROCEDURE aria(H,L:INTEGER; VAR s:INTEGER);
BEGIN
s:= H * L
END;

{Program principal}
BEGIN
{Calcul arie triunghi 1}
aria(40,70,s1);
WRITELN('Arie 1=',s1:6);
{Calcul arie triunghi 2};
aria(80,70,s2);
WRITELN('Arie 2=',s2:6);
{Calcul arie triunghi 3}
aria(120,80,s3);
WRITELN('Arie 3=',s3:6);
{Calcul si afisare arie totala}
stotal := s1 + s2 + s3;
WRITELN('Aria totala =',stotal:6);
READLN
END.
```

Fig. 11.11

```
Arie 1=   2800
Arie 2=   5600
Arie 3=   9600
Aria totala = 18000
```

Fig. 11.12

Din fig. 11.11 comparativ cu fig. 11.9 se observă limitarea declarațiilor globale la numai patru variabile: **s1**, **s2**, **s3**, **stotal**. De asemenea antetul procedurii **aria** este completat cu lista de parametrii formali, dintre care:

- datele de intrare înălțimea **H** respectiv lățimea **L** sunt parametrii formali constantă;
- data de ieşire **s** (valoarea calculată a ariei) este parametru formal variabilă, deoarece valoarea calculată a ariei din interiorul procedurii trebuie transmisă spre programul apelant.

În programul principal, apelul de subrutină conține, pe primele două poziții în lista de parametrii actuali, valorile numerice ale înălțimii respectiv lățimii, iar pe ultima poziție variabila care va primi valoarea ariei, calculată în interiorul procedurii (în variabila **s**).

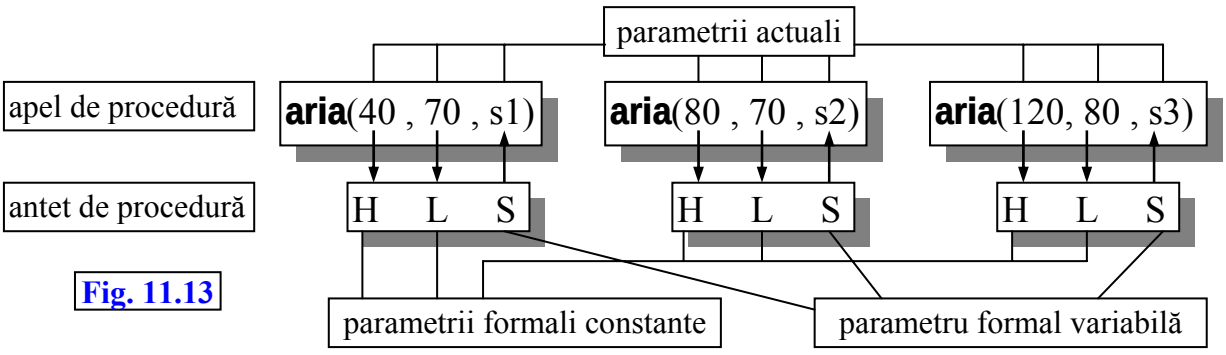


Fig. 11.13

Fig. 11.13 ilustrează grafic corespondența dintre parametrii actuali din apelul de procedură cu parametrii formali din antetul de procedură, evidențiind sursa şi sensul transmiterii valorilor.

11.7 APLICAȚII

11.7.1 Ordonarea crescătoare a trei numere

11.3 Program TurboPascal pentru ordonarea crescătoare a trei numere

Ne propunem ordonarea crescătoare a trei numere folosind următorul algoritm: dacă primul număr este mai mare decât al doilea, se inversează valorile celor două numere; dacă primul număr este mai mare decât al treilea, se inversează valorile celor două numere; dacă al doilea număr este mai mare decât al treilea, se inversează valorile celor două numere; pentru inversarea valorilor a două variabile se va utiliza algoritmul din & 5.6.3, exemplul 5.7, încadrat într-o procedură.

Fig. 11.14

```
PROGRAM ordonare_3_numere;
{Ordonarea crescatoare a trei numere}

VAR n1, n2, n3: INTEGER;

PROCEDURE invers(VAR a,b: INTEGER);
{Inversarea valorilor a doua variabile}
VAR x:INTEGER;
BEGIN
x := a;
a := b;
b := x;
END;

{Program principal}
BEGIN
WRITELN('Introduceti 3 numere intregi');
READLN(n1,n2,n3);
IF n1 > n2 THEN invers(n1,n2);
IF n1 > n3 THEN invers(n1,n3);
IF n2 > n3 THEN invers(n2,n3);
WRITELN('Numerele inversate sunt');
WRITELN(n1:4,n2:4,n3:4);
READLN
END.
```

Datele de intrare în subprogramul **invers** sunt două variabile nominalizate prin identificatorii **a** şi **b** în lista de parametrii formali. Datele de ieşire sunt returnate programului principal prin intermediul aceloraşi variabile, dar care vor conține valorile inversate ale lor. Deoarece variabilele trebuie să returneze valori către programul principal, ele trebuie declarate ca parametrii actuali variabili, deci în listă declarația lor va fi precedată de cuvântul **VAR**. Algoritmul de inversare a valorilor a două variabile utilizează variabila **x** suplimentară pentru inversare, care este declarată în interiorul procedurii, deci va avea un caracter local (există numai pe durata execuției procedurii **invers**).

```
Introduceti 3 numere intregi
30 25 10
Numerele inversate sunt
  10  25  30
```

Fig. 11.15

În programul principal procedura **invers** este apelată de trei ori, de fiecare dată însă cu alte două numere, pentru care se impune inversarea valorilor.

Se remarcă instrucțiunea de apel a procedurii **invers**, care este format din numele acesteia, urmat de lista (încadrată între paranteze rotunde) parametrilor actuali, despărțiți prin separatorul “**,**”.

De asemenea se remarcă claritatea programului rezultată din încadrarea instrucțiunilor de inversare într-o unică procedură, apelată de trei ori.

11.7.2 Sumarea a două matrici

E 11.4 Program TurboPascal pentru sumarea a două matrici

Ne propunem sumarea a două matrici, efectuând sumarea elementelor din poziții corespondente din matrici.

Fig. 11.16

```
PROGRAM matrice;
{Insumarea a doua matrici patratice
cu valori generate aleator}
CONST N=5;
TYPE mat=ARRAY[1..N, 1..N] OF INTEGER;
VAR A,B,C:mat;
   i,j:INTEGER;

PROCEDURE genmat(nm:INTEGER;VAR x:mat);
{Generarea valori aleatoare intr-o matrice
patratica-valori cuprinse intre 0 si "nm"}
VAR i,j : INTEGER;
BEGIN
RANDOMIZE;
FOR i:=1 TO nm DO
   FOR j:=1 TO nm DO
      x[i,j]:=RANDOM(nm+1);
END;

PROCEDURE scriumat (nm:INTEGER;x:mat);
{Afisare valori matrice patratica}
VAR i,j : INTEGER;
BEGIN
FOR I:=1 TO nm DO
   BEGIN
      FOR J:=1 TO nm DO  WRITE(x[i,j],' ');
      {Trecere cursor pe linia urmatoare}
      WRITELN;
   END;
END;
```

IC

```
{Program principal}
BEGIN
genmat(N,A);
WRITELN('Matricea A este:');
scriumat(N,A);
WRITELN('Matricea B este:');
genmat(N,B);
scriumat(N,B);
{Insumare matrici A+B}
FOR i:=1 TO N DO
  FOR j:=1 TO N DO
      C[i,j]:=A[i,j]+B[i,j];
WRITELN('Matricea suma C este:');
scriumat(N,C);
READLN
END.
```

```
Matricea A este:
3 3 5 4 2
5 1 5 2 1
0 3 5 0 5
4 4 0 2 1
2 5 3 1 0
Matricea B este:
4 2 5 4 4
0 1 2 0 0
1 0 2 1 2
4 2 5 1 4
5 4 2 0 1
Matricea suma C este:
7 5 10 8 6
5 2 7 2 1
1 3 7 1 7
8 6 5 3 5
7 9 5 1 1
```

Fig. 11.17

Programul cuprinde două proceduri ce operează cu matrici pătratice:

- **genmat** – pentru generarea aleatoare a valorilor;
- **scriumat** - pentru afişarea valorilor.

Programul debutează cu declarația valorii constantei **N**, utilizată la specificarea dimensiunii matricii pătratice în instrucțiunea **TYPE**, unde se declară caracteristicile acesteia, prin definirea tipului tabloului. În continuare se definesc variabilele **A**, **B** şi **C**, toate de tipul **mat**, deci matrici pătratice de dimensiune **N**, precum şi variabilele **i**, **j** care se vor utiliza ca şi contori de ciclu în programul principal. Toate aceste variabile sunt variabile globale.

Procedura **genmat** conține în lista parametrilor formali doi parametrii:

- parametrul formal constantă **nm** care reprezintă dimensiunea matricii;
- parametrul formal variabilă **x** de tip **mat**; prin acest parametru valorile generate aleator în interiorul procedurii vor fi transmise programului principal.

În procedura **genmat** se redeclară ca variabile locale variabilele **i**, **j**, care se vor utiliza ca şi contori de ciclu în interiorul procedurii, existența acestora fiind menținută numai pe perioada de execuție a procedurii.. Elementele matricii **x** se vor genera prin valori aleatoare prin funcția **RANDOM** (& 8.2).

Procedura **scriumat** conține în lista parametrilor formali doi parametrii:

- parametrul formal constantă **nm** care reprezintă dimensiunea matricii;
- parametrul formal constantă **x** de tip **mat**; prin acest parametru se transmit, de către programul principal spre procedură, valorile matricii.

În procedura **scriumat** se redeclară ca variabile locale variabilele **i**, **j**, care se vor utiliza ca şi contori de ciclu în interiorul procedurii, existența acestora fiind menținută numai pe perioada de execuție a procedurii.

Programul principal conține următoarea secvență de instrucțiuni:

- generarea valorilor aleatoare ale matricii **A** prin apelul procedurii **genmat**, transmițând acesteia parametrii actuali **N** respectiv **A**, ultimul parametru având structură matriceală;
- afişarea unui mesaj informativ, urmat de afişarea valorilor matricii **A** prin apelul procedurii **scriumat**, transmițând acesteia parametrii actuali **N** respectiv **A**, ultimul parametru având structură matriceală;
- generarea valorilor aleatoare ale matricii **B** prin apelul procedurii **genmat**, transmițând acesteia parametrii actuali **N** respectiv **B**, ultimul parametru având structură matriceală;
- afişarea unui mesaj informativ, urmat de afişarea valorilor matricii **B** prin apelul procedurii **scriumat**, transmițând acesteia parametrii actuali **N** respectiv **B**, ultimul parametru având structură matriceală;
- sumarea elementelor matricii **C**, prin sumarea element cu element a valorilor corespondente din matricile **A** şi **B**; sumarea se realizează prin îmbricarea a două cicluri, pentru parcurgerea tuturor elementelor; contorii acestor cicluri sunt variabilele globale **i**, **j**, declarate anterior definițiilor celor două proceduri;
- afişarea unui mesaj informativ, urmat de afişarea valorilor matricii **C** prin apelul procedurii **scriumat**, transmițând acesteia parametrii actuali **N** respectiv **C**.

În programul principal procedura **genmat** este apelată de două ori, iar procedura **scriumat** este apelată de trei ori. Rezultă cu claritate economia de instrucțiuni realizată prin scrierea o singură dată a instrucțiunilor prin gruparea lor într-un subprogram şi apelarea repetată a lor fără rescrierea instrucțiunilor, economie care devine substanțială în cazul în care numărul de instrucțiuni din procedură creşte considerabil. De asemenea un alt avantaj important al acestei tehnici este sistematizarea programelor, prin standardizarea programării algoritmilor des utilizați realizată prin intermediul procedurilor.

11.7.3 Calcul combinări

E 11.5 Program TurboPascal pentru calculul combinărilor

Exemplul următor prezintă sub forma a două programe calculul combinărilor, din care primul calculează factorialul prin apelul de trei ori a aceleiaşi secvenţe de instrucţiuni, iar al doilea foloseşte funcţia **FACTORIAL** pentru calculul factorialului unui număr.

$$C_N^R = \frac{N!}{R! \cdot (N-R)!}$$

Fig. 11.18a

```
PROGRAM combinari1;
{Varianta 1-fara functie}
VAR i, N,R:INTEGER;
   NF, RF, NMRF, combinari:REAL;
BEGIN
WRITELN('Introduceti N, R cu N>R !');
READLN (N,R);
NF := 1;
FOR I:=1 TO N DO NF:=NF*i;
RF := 1;
FOR I:=1 TO R DO RF:=RF*i;
NMRF := 1;
FOR I:=1 TO N-R DO NMRF:=NMRF*i;
combinari := NF/(RF*NMRF);
WRITELN('Combinari=',combinari:10:4);
READLN
END.
```

Fig. 11.18b

```
PROGRAM combinari2;
{Varianta 2-cu functie}
VAR N,R:INTEGER;
   combinari:REAL;

{Definire functie calcul factorial}
FUNCTION factorial(n1:INTEGER):REAL;
VAR i:integer;
   f:real;
BEGIN
   f:=1;
   FOR i:=1 TO n1 DO f:=f*i;
   factorial:=f;
END;

{Program principal}
BEGIN
WRITELN('Introduceti N, R cu N>R');
READLN (N,R);
combinari := factorial(N)/(factorial(R)*factorial(N-R));
WRITELN('Combinari=',combinari:10:4);
READLN
END.
```

```
Introduceti N, R cu N>R
10
5
Combinari=  252.0000
```

Fig. 11.19

Factorialul unui număr dat "N" exprimat matematic prin N!= 1x2x3x4...xN, reprezintă produsul primelor "N" numere naturale, care se efectuează printr-un ciclu **FOR**. Se observă calcularea acestui factorial prin rescrierea de trei ori a aceloraşi instrucţiuni (iniţializarea variabilei factorial rezultat cu 1 şi execuţia ciclului **FOR**) în programul din fig. 11.18a.

În programul din fig 11.18b, se defineşte funcţia **factorial** al cărui rezultat va fi de tip **REAL** şi care conţine un singur parametru formal **n1** de tip **INTEGER**, cu semnificaţia numărului până la care se calculează factorialul. Variabilele **i** şi **f** sunt locale funcţiei, existenţa lor fiind asigurată numai pe durata execuţiei acesteia. Instrucţiunea de calcul a combinărilor din programul principal conţine 3 apeluri ale funcţiei, aplicată de fiecare dată altui număr.

11.7.4 Generare pătrate magice

E 11.6 Program TurboPascal pentru generare pătrate magice

Un pătrat magic este o matrice la care suma elementelor pe linii, coloane şi pe cele două diagonale sunt egale. Algoritmul de generare a pătratelor magice impare este:

- din şirul 1,2,3,4,..., N se aleg aleator N numere (care pot fi şi multiple) şi se depun în prima linie a primului pătrat; restul liniilor (a 2-a la ultima) se completează începând cu numărul din prima coloană de după mijlocul liniei şi continuând cu primele;
- din şirul 0xN, 1xN, 2xN,...., (N-1)xN se aleg aleator N numere (care pot fi şi multiple) şi se depun în prima linie a celui de-al doilea pătrat; restul liniilor se completează la fel ca la primul pătrat, dar pornind de la mijlocul liniei;
- prin adunarea element cu element a celor două pătrate rezultă un pătrat magic.

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

Primul pătrat

+

5	0	20	10	15
20	10	15	5	0
15	5	0	20	10
0	20	10	15	5
10	15	5	0	20

Al doilea pătrat

=

7	1	24	13	20
23	15	17	6	4
16	9	3	25	12
5	22	11	9	8
14	18	10	2	21

Pătrat magic

Programul este prezentat în fig. 11.20 a şi b, iar rezultatele acestuia în fig. 11.21. Se definesc variabilele **A**, **B**, **C** cu structură de matrice pătratică de dimensiuni maximale 50 elemente şi variabilele globale **i**, **j**, **N** de tip **INTEGER**.

Procedura **calcmat** are rolul de a genera elementele matricii de la linia 2 până la ultima linie, conform logicii impuse de algoritm; procedura primeşte ca date de intrare dimensiunea matricii **nm**, **coloana** de referinţă pentru translaţia valorilor, ambele de tip **INTEGER**, precum şi matricea subiect **X**, ale cărei elemente din prima linie sunt date de intrare, restul liniilor fiind date de ieşire, motiv pentru care parametrul formal **X** este declarat ca variabilă. Logica de generare a valorilor matricii este următoarea: elementul curent al matricii **X[l,c]** preia valorile elementului **X[l-1,contor]** (din linia anterioara **l –1** şi coloana specificată de variabila **contor)**; preluarea începe de la coloana de referinţă, a cărei valoare este memorată iniţial în variabila **contor**, variabilă a cărei valoare creşte cu 1 la fiecare preluare; dacă, prin mărire cu 1, valoarea variabilei **contor** depăşeşte numărul de coloane **nm** din matricea **X (**s-a depăşit frontiera matricii**)**, atunci variabila **contor** se iniţializează cu 1.

Procedura **scriumat** are rolul de a afişa elementele unei matrici pătratice; fiind preluată din & 11.7.2, exemplul 11.4, toate semnificaţiile variabilelor rămân valabile.

Programul principal debutează cu afişarea unui mesaj informativ şi citirea dimensiunii matricii în variabila **N**, ambele instrucţiuni fiind incluse într-o buclă **WHILE** (& 7.1) care forţează reluarea introducerii valorii lui **N** dacă: numărul introdus este par, este mai mic decât 5 sau mai mare de 49. În continuare se generează valorile aleatoare în prima linie a matricii **A** şi **B** conform logicii algoritmului şi pentru domeniu de valori impuse de acesta.

Fig. 11.20a

```
PROGRAM patrat_magic;
TYPE matrice=ARRAY[1..50, 1..50] OF INTEGER;
VAR A,B,C: matrice;
      i,j,N: INTEGER;

PROCEDURE calcmat(nm,coloana:INTEGER; VAR X:matrice);
{Generare valori aleatoare in liniile 2..N ale matricii}
VAR l, c, contor : INTEGER;
BEGIN
FOR l:= 2 TO nm DO
   BEGIN
        contor:=coloana;
        FOR c:= 1 TO nm DO
          BEGIN
IC        X[l,c]:=X[l-1,contor];
IC        INC (contor);
          IF contor > N THEN contor:=1;
          END
   END;
END;

PROCEDURE scriumat (nm:INTEGER;x:matrice);
{Afisare valori matrice patratica}
VAR i,j : INTEGER;
BEGIN
FOR i:=1 TO nm DO
   BEGIN
IC   FOR j:=1 TO nm DO  WRITE(x[i,j]:3);
     {Trecere cursor pe linia urmatoare}
     WRITELN;
   END;
END;
```

Programul principal continuă cu generarea restului de valori în matricea **A**, prin apelarea procedurii **calcmat** (impunând ca şi coloană de referinţă valoarea calculată a expresiei : **N div 2+2**) precum şi afişarea acesteia prin apelul rutinei **scriumat**. Procesul se repetă şi pentru matricea **B**, cu diferenţa expresiei de calcul a coloanei de referinţă: **N div 2+1**, unde operatorul **div** oferă ca rezultat câtul întreg al împărţirii a două numere întregi (& 4.2.1, tabel 4.4). Prin sumarea celor două matrici se obţine matricea pătratului magic **C**, care se afişează prin apelul procedurii **scriumat**.

Fig. 11.20b

```
{Program principal}
BEGIN
WHILE (ODD(N)=False) or (N<5) or (N>49) DO
IC   BEGIN
     WRITELN('Gradul patratului, intre 5-49 ');
     READLN(N);
     END;
{Generare prima linie in matricile A si B}
RANDOMIZE;
FOR j:=1 TO N DO
IC   BEGIN
       A[1,j]:=1+RANDOM(N);
       B[1,j]:=N*RANDOM(N);
     END;
{Calcul restul de linii in matrici si afisare}
calcmat(N,N div 2+2,A);
WRITELN('Primul patrat');
scriumat(N,A);
calcmat(N,N div 2+1,B);
WRITELN('Al doilea patrat');
scriumat(N,B);
{Calcul patrat magic}
FOR i:=1 TO n DO
  FOR j:=1 TO N DO C[i,j]:=A[i,j]+B[i,j];
WRITELN('Patratul magic');
scriumat(N,C);
READLN
END.
```

```
Gradul patratului, intre 5-49
5
Primul patrat
  4  5  2  1  2
  1  2  4  5  2
  5  2  1  2  4
  2  4  5  2  1
  2  1  2  4  5
Al doilea patrat
  0  0 10 20  5
 10 20  5  0  0
  5  0  0 10 20
  0 10 20  5  0
 20  5  0  0 10
Patratul magic
  4  5 12 21  7
 11 22  9  5  2
 10  2  1 12 24
  2 14 25  7  1
 22  6  2  4 15
```

Fig. 11.21

11.7.5 Ordonarea crescătoare diagonală principală matrice pătratică

11.7 Program TurboPascal pentru ordonarea crescătoare diagonală principală matrice pătratică

Se cere ordonarea crescătoare a diagonalei principale a unei matrici pătratice, prin inversarea liniilor şi coloanelor.

```
PROGRAM ordonare_matrice;
{Ordonare crescatoare diagonala matrice
patratica prin inversare linii si coloane}
TYPE matrice=ARRAY[1..50, 1..50] OF INTEGER;
VAR A: matrice;
   i,j,N: INTEGER;
   schimb:BOOLEAN;

PROCEDURE inv_lin(nm,lin:INTEGER; VAR X:matrice);
{Inversare linia lin cu lin+1 in matrice}
VAR c, vint : INTEGER;
BEGIN
FOR c:= 1 TO nm DO
   BEGIN
      vint := X[lin,c];
IC    X[lin,c] := X[lin+1,c];
      X[lin+1,c] := vint
   END;
END;

PROCEDURE inv_col(nm,col:INTEGER;VAR X:matrice);
{Inversare coloana col cu col+1 in matrice}
VAR l, vint : INTEGER;
BEGIN
FOR l:= 1 TO nm DO
   BEGIN
      vint := X[l,col];
IC    X[l,col] := X[l,col+1];
      X[l,col+1] := vint
   END;
END;

PROCEDURE scriumat (nm:INTEGER;x:matrice);
{Afisare valori matrice patratica}
VAR i,j : INTEGER;
BEGIN
FOR i:=1 TO nm DO
   BEGIN
      FOR j:=1 TO nm DO  WRITE(x[i,j]:3);
IC    {Trecere cursor pe linia urmatoare}
      WRITELN;
   END;
END;
```

Fig. 11.22a

```
{Program principal}
BEGIN
WRITELN('Gradul matricii=');
READLN(N);
{Generare elemente in matrice}
RANDOMIZE;
FOR i:=1 TO N DO
 FOR j:=1 TO N DO
    A[i,j]:=10+RANDOM(10*N);
WRITELN('Matricea initiala');
scriumat(N,A);
{Ordonare diagonala}
REPEAT
   schimb:=FALSE;
   FOR i:=1 TO N-1 DO
    IF A[i,i] > A[i+1,i+1] THEN
      BEGIN
         inv_lin(N,i,A);
IC       inv_col(N,i,A);
         schimb:=TRUE
      END;
UNTIL schimb=FALSE;
WRITELN('Matricea ordonata');
scriumat(N,A);
READLN
END.
```

Fig. 11.22b

Pentru ordonarea componentelor diagonalei principale se va utiliza metoda BUBBLE SORT (& 9.3.10, exemplul 9.10), aplicată elementelor aparţinând acesteia. Elementele de pe diagonala principală se bucură de proprietatea egalităţii indicelui de linie cu a celui de coloană.

Programul conţine trei proceduri şi debutează cu definirea structurii **matrice** de tip tablou matriceal pătratic cu dimensiunea maximă de 50, după care se declară variabilele **A** de tip matrice, **i**, **j**, **N** de tip **INTEGER** şi **schimb** de tip **BOOLEAN**.

```
Gradul matricii=
5
Matricea initiala
 41 33 22 35 47
 42 14 29 16 49
 32 29 45 19 42
 38 46 40 27 54
 33 51 58 36 43
Matricea ordonata
 14 16 42 49 29
 46 27 38 54 40
 33 35 41 47 22
 51 36 33 43 58
 29 19 32 42 45
```

Fig. 11.23

Procedura **inv_lin** este dedicată inversării elementelor a două linii într-o matrice (linia **lin** cu linia **lin+1**) şi conţine ca parmetrii formali constantă: **nm** – gradul matricii, linia de referinţă a inversării **lin** şi parametru formal variabilă - matricea **X**, care va fi returnată programului principal cu liniile inversate.

Procedura **inv_col** este dedicată inversării elementelor a două coloane într-o matrice (coloana **col** cu coloana **col+1**) şi conţine ca parmetrii formali constantă: **nm** – gradul matricii, coloana de referinţă a inversării **col** şi parametru formal variabilă - matricea **X**, care va fi returnată programului principal cu coloanele inversate.

Procedura **scriumat** are rolul de a afişa elementele unei matrici pătratice; fiind preluată din & 11.7.2, exemplul 11.4, toate semnificaţiile variabilelor rămân valabile.

Programul principal generează aleator elementele matricii, după care execută ordonarea diagonalei principale, într-un ciclu **REPEAT-UNTIL** (& 7.2), a cărei logică este aceeaşi cu ordonarea componentelor unui vector (& 9.3.10, exemplul 9.10), cu precizările suplimentare:

- elementele de ordonat aparţin diagonalei principale a matricii, deci vor fi comparate componentele **A[i,i]** cu **A[i+1, i+1]**, adică elementele pentru care indicele de linie este egal cu cel de coloană;
- inversarea liniei curente **i** a matricii cu linia i+1 se va produce prin intermediul procedurii **inv_lin**;
- inversarea coloanei curente **i** a matricii cu coloana i+1 se va produce prin intermediul procedurii **inv_col**;
- la fiecare inversare efectuată, variabila **schimb** primeşte valoarea logică **TRUE**.

Parametrii actuali transmişi de către programul principal celor două proceduri sunt:

- variabila **N** care reprezintă gradul matricii;
- variabila corespunzătoare liniei respectiv coloanei, adică variabila **i**; coincidenţa variabilei **i** atât pentru linie cât şi pentru coloană este impusă de identitatea valorii numerice a liniei şi coloanei pentru elementele de pe diagonala principală;
- matricea **A** asupra căreia se acţionează; matricea intră în procedură pentru a fi modificată în sensul inversării valorilor a două linii sau coloane.

Procesul se repetă până când variabila **schimb** rămâne la valoarea logică **FALSE** (valoare cu care este iniţializată la fiecare repetiţie), deci până când nu se mai execută nici o inversare, ceea ce corespunde cu poziţionarea ordonată a elementelor diagonalei principale.

Afişarea finală a matricii modificate conform cerinţelor impuse ale problemei se realizează prin apelarea procedurii **scriumat**.

11.7.6 Generare număr printr-o logică impusă

E 11.8 Program TurboPascal pentru generare număr

Cu un număr natural se pot face următoarele operații:

A.) se adaugă la sfârşit cifra 4;
B.) se adaugă la sfârşit cifra 0;
C.) se împarte la 2 (dacă este par).

Fiind dat un număr **X**, se cere ca pornind de la valoarea inițială 4 şi aplicând operațiile descrise anterior să se genereze numărul X şi să se afişeze şirul transformărilor intermediare. Exemplu: pentru numărul **5** şirul transformărilor este: **4**, **2**, **1**, **10**, **5**.

Problema nu comportă probleme deosebite din punct de vedere al operațiilor de aplicat, dar generează incertitudini datorită opțiunii de selecție a tipului de operație care să fie aplicată, cu efecte privind ramificarea multiplă a algoritmului. Din acest motiv, metoda de rezolvare apelează la o metodă care oferă o generare unică a şirului: pornind de la numărul **X** dat, se vor aplica operațiile inverse celor din enunț, generând astfel şirul transformărilor inverse; afişând acest şir în ordine inversă, va rezulta şirul care s-ar fi obținut prin aplicarea operațiilor directe. Unicitatea acestei soluții constă în caracterul determinist al operației care trebuie aplicată, după următoarea logică: dacă ultima cifră este 0 sau 4 aceasta se elimină, în caz contrar numărul se dublează. Exemplu: plecând de la numărul **5** şi aplicând operațiile inverse conform logicii enunțate, rezultă şirul **5**, **10**, **1**, **2**, **4**, şir care afişat în ordine inversă reprezintă şirul căutat.

Programul este prezentat în fig. 11.24. Secțiunea declarațiilor conține definiția vectorului **a** cu maxim 1000 componente de tip **INTEGER**, variabilele globale **x**, **lx**, **i**, **k** de tip **INTEGER**, **uc** de tip **STRING**[1] şi **xstr** de tip **STRING**. Programul conține două funcții, ambele având aceeaşi funcționalitate, deci oferă acelaşi rezultat, dar prin două metode diferite. Desigur că una dintre ele este suficientă pentru problema în cauză, dar s-au prezentat comparativ ambele din motive didactice. Funcționalitatea acestor funcții este de a elimina ultima cifră a unui număr specificat.

Funcția **elimin** posedă ca parametrii formali variabila **şir** de tip **STRING** prin care numărul subiect al eliminării (exprimat sub formă de şir de caractere) este furnizat procedurii şi variabila **lsir** de tip **INTEGER**, ce reprezintă lungimea şirului echivalent numărului, adică numărul de cifre al acestuia. Rezultatul funcției este o valoare de tip **INTEGER**, care reprezintă numărul inițial din care s-a eliminat ultima cifră. Procedura lucrează cu variabilele locale **numar** şi **eroare** de tip **INTEGER**. Programarea eliminării ultimei cifre constă în: din şirul inițial **sir** se extrage, prin funcția **COPY,** un subşir care conține caracterele şirului inițial de la prima la penultima poziție. Asupra acestui şir se aplică funcția **VAL**, pentru a genera, în variabila numerică **număr**, numărul rezultat din eliminarea ultimei cifre. Prin instrucțiunea de atribuire, valoarea variabilei număr se atribuie funcției **elimin**, pentru returnarea valorii spre programul principal.

Funcția **elimin1** posedă un singur parametru formal: variabila numerică **nr** de tip **INTEGER** prin care numărul subiect al eliminării (exprimat sub formă numerică) este furnizat procedurii. Rezultatul funcției este o valoare de tip **INTEGER**, care reprezintă numărul inițial din care s-a eliminat ultima cifră. Procedura nu necesită variabilele locale.

```
PROGRAM generare_numar;
VAR a:ARRAY[1..1000] OF INTEGER;
   x,lx,i, k:INTEGER;
   uc:STRING[1];
   xstr:STRING;

FUNCTION elimin(sir:STRING;lsir:INTEGER):INTEGER;
{Eliminare ultima cifra dintr-un numar
transmis sub forma de sir de caractere}
VAR numar,eroare:INTEGER;
BEGIN
VAL(COPY(sir,1,lsir-1),numar,eroare);
elimin:=numar
END;

FUNCTION elimin1(nr:INTEGER):INTEGER;
{Eliminare ultima cifra dintr-un numar
transmis sub forma numerica}
BEGIN
elimin1:=nr div 10
END;

{Program principal}
BEGIN
WRITE('Nr=');
READLN(x);
i:=1;
a[1]:=x;
WHILE a[i]<>4 DO
   BEGIN
      STR(a[i],xstr);
      lx:=LENGTH(xstr);
      uc:=COPY(xstr,lx,1);
      INC(i);
      IF  (uc='0') OR (uc='4') THEN
            {a[i]:=elimin(xstr,lx)}
         a[i]:=elimin1(a[i-1])
      ELSE
         a[i]:=a[i-1]*2;
   END;
{Afisare vector in ordine inversa}
FOR k:=i DOWNTO 1 DO WRITELN(a[k]);
READLN
END.
```

IC

Fig. 11.24

```
Nr=542
4     Nr =13
2     4
24    2
12    1
6     10
3     104
34    52
344   26
172   13
86    ----
864   Nr=7
432   4
216   2
108   1
1084  14
542   7
      72
```

Fig. 11.25

Programarea eliminării ultimei cifre constă în împărţirea întreagă a numărului iniţial cu 10, folosind funcţia **div** şi atribuirea rezultatului acestei împărţiri funcţiei **elimin1**, pentru returnarea valorii spre programul principal.

Programul principal debutează cu citirea numărului în variabila numerică **x**, iniţializarea cu 1 a variabilei contor i şi memorarea numărului **x** în prima poziţia a vectorului **a**. În continuare, prin ciclul **WHILE** se execută repetitiv (cât timp valoarea din poziţia **i** a vectorului **a** este diferită de 4) următoarele instrucţiuni:

- transformarea numărului în şirul echivalent **xstr** prin funcţia **STR**;
- calculul numărului de caractere al şirului **xstr** prin funcţia **LENGTH**;
- extragerea ultimului caracter din şirul **xstr** în variabila **uc** prin funcţia **COPY**;
- incrementarea unitară a variabilei **i**;
- testarea prin instrucţiunea **IF** a coincidenţei caracterului final cu una din caracterele ‘0’ sau ‘4’:
- în caz de identitate (ramura **THEN**) se aplică una din funcţiile **elimin** sau **elimin1**; pentru exemplificare ambele apeluri sunt prezentate în program, primul apel fiind însă încadrat între acolade, deci constituie un comentariu;
- pentru situaţia contrară identităţii (ramura **ELSE**) numărul curent se obţine prin dublarea numărului anterior;
- în final vectorul **a**, care memorează şirul transformărilor numerice, este afişat în ordine inversă.

Fig. 11.25 prezintă trei variante ale execuţiei programului.

11.7.7 Eliminarea valorilor eronate dintr-un şir

11.9 Program TurboPascal pentru eliminarea valorilor eronate dintr-un şir

Eliminarea valorilor eronate dintr-un şir constituie o problemă statistică. Rezultatele unui şir de măsurători pot fi afectate de erori sistematice sau grosolane, acestea din urmă trebuie eliminate din şir. Metoda statistică denumită "*metoda celor 3 sigme*" ne oferă o soluţie matematică de eliminare a punctelor susceptibile de erori: considerând un şir de valori numerice $\mathbf{X_i}$, unde **i=1..N** şi o valoare de analizat $\mathbf{X_d}$ aparţinând şirului, valoarea $\mathbf{X_d}$ trebuie eliminată din şir dacă este îndeplinită condiţia:

$$\frac{X_d - Xmed}{s} > 3(sigme) \quad unde: \quad Xmed = \frac{\sum_{i=1}^{N} X_i}{N} \; ; s = \sqrt{\frac{\sum_{i=1}^{N} X_i^{\,2} - \frac{(\sum_{i=1}^{N} X_i)^2}{N}}{N-1}}$$

Xmed - media valorilor şirului ; **s** – abaterea pătratică standard.

Programul este prezentat în fig. 11.26, iar rezultatele execuţiei în fig. 11.27, pentru şirul de valori **2**, **50**, **52**, **54** şi valoarea de analizat **Xd=2**. Secţiunea declaraţiilor conţine definiţia tipului de date vectorial **vct** cu maxim 1000 componente de tip **INTEGER** şi secţiunea variabilelor: vectorul **a** de tip **vct**, variabilele globale **N**, **i** de tip **INTEGER**, **xd**, **media_sir**, **s**, **criteriu** de tip **REAL**. Programul conţine două funcţii.

Funcţia **suma_vect** calculează suma elementelor unui vector, aplicând algoritmul din & 9.3.6, exemplul 9.6 şi posedă ca parametrii formali variabila numerică **nv** de tip **INTEGER** cu semnificaţia numărului de componente a vectorului cu componente reale **xv** de tip **vct**. Procedura utilizează variabila locală **sumav** ca variabilă de sumare. Sumarea se calculează într-un ciclul **FOR**. Prin instrucţiunea de atribuire, valoarea calculată a sumei se atribuie funcţiei **suma_vect**, pentru returnarea valorii spre programul principal.

Funcţia **sumap_vect** calculează suma pătratelor elementelor unui vector, aplicând algoritmul din & 9.3.6, exemplul 9.6 şi posedă ca parametrii formali variabila numerică **nv** de tip **INTEGER** cu semnificaţia numărului de componente a vectorului cu componente reale **xv** de tip **vct**. Procedura utilizează variabila locală **sumavp** ca variabilă de sumare. Sumarea se calculează într-un ciclul **FOR**. Prin instrucţiunea de atribuire, valoarea calculată a sumei se atribuie funcţiei **sumap_vect**, pentru returnarea valorii spre programul principal.

Programul principal debutează cu introducerea datelor de intrare:

- numărul de componente ale şirului de valori analizat **N**;
- şirul de valori de analizat, memorate în vectorul **a**;
- valoarea de analizat din punct de vedere al eliminării, memorată în variabila **xd**.

În continuare se memorează suma componentelor vectorului în variabila **media_sir**, calculat[prin apelul funcţiei **suma_vect** aplicată parametrilor actuali **N** respectiv **a** şi se calculează abaterea pătratică medie în variabile **s**; în interiorul expresiei acesteia se apelează funcţia **sumap_vect** aplicată parametrilor actuali **N** respectiv **a**, pentru calcularea sumei pătratelor elementelor vectorului. În final se calculează criteriul celor 3 sigma în variabila **criteriu** şi se afişează rezultatele analizei.

Se remarcă structura asemănătoare a celor două funcţii şi diferenţele minimale dintre ele. Desigur că în locul utilizării a două funcţii, atât suma elementelor cât şi a pătratelor lor se poate calcula într-o singură procedură, ceea ce va avea efectul reducerii numărului de instrucţiuni. S-a preferat aceasta variantă din motive didactice. De altfel, în general, un program poate fi compus în mai multe feluri posibile, iar găsirea celei mai eficiente modalităţi de programare se obţine printr-o experienţă câştigată prin exerciţii.

```
PROGRAM statistica;
{Eliminare statistica valori eronate din sir}
TYPE vct=ARRAY[1..1000] OF REAL;
VAR a: vct;
   N, i :INTEGER;
   xd, media_sir, s, criteriu :REAL;

FUNCTION suma_vect(nv:INTEGER;xv:vct):REAL;
{Suma elementelor unui vector}
VAR i:INTEGER;
  sumav:REAL;
BEGIN
sumav:=0;
FOR i:=1 TO nv DO sumav:=sumav+xv[i];
suma_vect := sumav
END;

FUNCTION sumap_vect(nv:INTEGER;xv:vct):REAL;
{Suma patratelor elementelor unui vector}
VAR i:INTEGER;
  sumavp:REAL;
BEGIN
sumavp:=0;
FOR i:=1 TO nv DO sumavp:=sumavp+SQR(xv[i]);
sumap_vect := sumavp
END;
```

Fig. 11.26a

```
{Program principal}
BEGIN
WRITE('Cite numere ? ');
READLN(N);
FOR i:=1 TO N DO
   BEGIN
IC    WRITE('Nr. ',i,'  ');
      READLN(a[i]);
   END;
WRITELN('Nr. de analizat');
READLN(xd);
media_sir := suma_vect(N, a);
s:=SQRT((sumap_vect(N, a)-SQR(media_sir)/N)/(N-1));
criteriu:=ABS((xd- media_sir /N)/s);
WRITELN('Abatere standard=',s:8:2);
WRITELN('Criteriu=',criteriu:8:2);
IF criteriu < 3 THEN
  WRITELN ('Numar corect')
ELSE
  WRITELN ('Numar incorect');
READLN
END.
```

Fig. 11.26b

```
Cite numere ? 4
Nr. 1  2
Nr. 2  50
Nr. 3  52
Nr. 4  54
Nr. de analizat
2
Abatere standard=  25.05
Criteriu=    1.50
Numar corect
```

Fig. 11.27

11.7.8 Generare pătrate greco-latine

E 11.10 Program TurboPascal pentru generare pătrate greco-latine

Un pătrat greco-latin este o matrice pătratică compusă din două şiruri de simboluri, plasate astfel ca împerecherea simbolurilor din cele două şiruri să fie unică şi fiecare simbol să apară o singură dată pe linie sau pe coloană. Algoritmul de generare a pătratelor greco-latine de grad **p** este următorul:

- În primul pătrat se aşează numărul **i+j-2** în celula (**i**,**j**), scăzând **p** dacă valoarea din celulă depăşeşte (**p-1**);
- În al doilea pătrat se aşează numărul **2xi+j-3** în celula (**i**,**j**), scăzând **p** sau **2xp** dacă valoarea din celulă depăşeşte (**p-1**);
- În primul pătrat se substituie simbolurile primului şir **a**,**b**,**c**,**d**,**e**,... cifrelor corespondente **0**,**1**,**2**,**3**,**4**,...;
- În al doilea pătrat se substituie simbolurile celui de-al doilea şir **A**,**B**,**C**,**D**,**E**,... cifrelor corespondente **0**,**1**,**2**,**3**,**4**,...;
- Se suprapun cele două pătrate pentru a obţine pătratul greco-latin.

a	b	c	d
b	a	d	c
c	d	a	b
d	c	b	a

Primul pătrat

A	B	G	D
G	D	A	B
D	G	B	A
B	A	D	G

Al doilea pătrat

aA	bB	cG	dD
bG	aD	dA	cB
cD	dG	aB	bA
dB	cA	bD	aG

Pătrat greco-latin

Programul este prezentat în fig. 11.28, iar rezultatele execuţiei în fig. 11.29 a şi b, pentru gradul pătratului **p=5**. Secţiunea declaraţiilor conţine definiţia a 2 constante de tip şir, tipului de date matriceal **matr** cu maxim 26 componente de tip **INTEGER** şi secţiunea variabilelor: matricile **a** şi **b** de tip **matr**, variabilele globale **p**, **i**, **j** de tip **INTEGER**. Programul conţine două funcţii.

Procedura **scriumat** are rolul de a afişa elementele unei matrici pătratice; fiind preluată din & 11.7.2, exemplul 11.4, toate semnificaţiile variabilelor rămân valabile.

Procedura **gen_elem** are rolul de a genera elementele celor două pătrate conform logicii algorimului. Parametrii formali sunt de tip constantă: **opt** cu semnificaţia opţiunii de calcul pentru primul (**opt=1**) sau al doilea pătrat (**opt=2**) şi **pm** gradul matricii, ambele fiind de tip **INTEGER**; parametru formal variabilă **x** memorează valorile calculate a matricii şi le transmite programului principal. Calculul se parcurge în două cicluri **FOR** îmbricate, diferit pentru cele două pătrate (selecţia se realizează prin instrucţiunea **IF** funcţie de valoarea variabile **opt**), după care funcţie de valoarea din celulă se scade valoarea **pm-1**, scădere care se aplică suplimentar pe ramura **ELSE** a primului **IF**, pentru programarea condiţiei algoritmului "*scăzând **p** sau **2xp***" pentru al doilea pătrat. Procedura se finalizează prin apelarea procedurii **scriumat**, pentru afişarea valorilor calculate ale pătratului.

Programul principal apelează succesiv procedura **gen_elem**, pentru generarea valorilor în cele 2 pătrate iniţiale, cu valorile parametrilor actuali: pentru primul pătrat **gen_elem**(1,p,a), iar pentru al doilea pătrat **gen_elem**(2,p,b).

```
Grad patrat ? 5
Primul patrat
  0  1  2  3  4
  1  2  3  4  0
  2  3  4  0  1
  3  4  0  1  2
  4  0  1  2  3
Al doilea patrat
  0  1  2  3  4
  2  3  4  0  1
  4  0  1  2  3
  1  2  3  4  0
  3  4  0  1  2
Patrat greco-latin
aA bB cC dD eE
bC cD dE eA aB
cE dA eB aC bD
dB eC aD bE cA
eD aE bA cB dC
```

Fig. 11.29a

Programul principal continuă cu afişarea pătratului greco-latin, cu o structură de tip matriceal, prin două cicluri **FOR** îmbricate, valorile fiind preluate din cele două şiruri din poziţiile specificate de valorile celulelor celor două pătrate mărite cu 1.

Fig. 11.28a

```
PROGRAM patrat_greco_latin;
{Generare patrate greco-latine}
CONST sir1:STRING[26]=('abcdefghilkjmnopqrstuvwxyz');
    sir2:STRING[26]=('ABCDEFGHIJKLMNOPQRSTUVWXYZ');
TYPE matr=ARRAY[1..26, 1..26] OF INTEGER;
VAR a,b: matr;
   p, i, j :INTEGER;

PROCEDURE scriumat (nm:INTEGER;x:matr);
{Afisare valori matrice patratica}
VAR i,j : INTEGER;
BEGIN
FOR i:=1 TO nm DO
   BEGIN
      FOR j:=1 TO nm DO  WRITE(x[i,j]:3);
      {Trecere cursor pe linia urmatoare}
      WRITELN;
   END;
END;

PROCEDURE gen_elem(opt,pm:INTEGER;VAR x:matr);
{Generare elemente matrice}
VAR l,c:INTEGER;
BEGIN
FOR l:=1 TO pm DO
  FOR c:=1 TO pm DO
     BEGIN
        IF opt=1 THEN
              X[l,c]:=l+c-2
        ELSE
           BEGIN
              X[l,c]:=2*l+c-3;
              IF x[l,c]>pm-1 THEN x[l,c]:=x[l,c]-pm;
           END;
        IF x[l,c]>pm-1 THEN x[l,c]:=x[l,c]-pm;
     END;
scriumat(pm,x)
END;
```

```
{Program principal}
BEGIN
WRITE('Grad patrat ? ');
READLN(p);
WRITELN('Primul patrat');
gen_elem(1,p,a);
WRITELN('Al doilea patrat');
gen_elem(2,p,b);
WRITELN('Patrat greco-latin');
FOR i:=1 TO p DO
   BEGIN
   FOR j:=1 TO p DO WRITE(sir1[a[i,j]+1]+sir2[b[i,j]+1]+' ');
   {Trecere cursor pe linia urmatoare}
   WRITELN;
   END;
READLN
END.
```

Fig. 11.28b

```
Grad patrat ? 3
Primul patrat
  0  1  2
  1  2  0
  2  0  1
Al doilea patrat
  0  1  2
  2  0  1
  1  2  0
Patrat greco-latin
aA bB cC
bC cA aB
cB aC bA
```

Fig. 11.29b

11.8 DECLARAŢIA ANTICIPATĂ A SUBPROGRAMELOR

Limbajul TurboPascal impune declararea identificatorilor şi a subprogramelor anterior utilizării acestora, ceea ce va asigura efectuarea unei singuri treceri pentru compilarea programului sursă. Sunt situaţii însă în care subprograme se apelează reciproc şi nu neapărat în ordinea definirii lor. De exemplu, două proceduri P1 şi P2 nu se vor putea apela reciproc, deoarece una dintre ele va fi în mod necesar plasată înaintea definiţiei celeilalte proceduri, situaţia care va produce eroare "**Error 3: Unknown Identifier**" la compilare din cauza apelului procedurii anterior definiţiei acesteia.

Limbajul TurboPascal oferă posibilitatea rezolvării acestei probleme prin directiva **FORWARD** prin care se pot separa, din punct de vedere al poziţionării în program, cele două componente ale unui subprogram: antetul şi blocul acestuia. Directiva **FORWARD** plasată imediat după antet (care include şi lista parametrilor) înlocuieşte simbolic blocul (corpul programului) şi permite poziţionarea acestuia din urmă mai jos în cadrul programului precedat numai de cuvântul cheie **PROCEDURA** sau **FUNCTION** urmat de numele subprogramului. Deci lista parametrilor formali se specifică numai la declaraţia **FORWARD**, fără a se mai repeta la declaraţia corpului procedurii sau funcţiei. Această este valabilă şi pentru specificarea tipului valorii returnate de funcţie. Între directiva **FORWARD** a unui subprogram şi corpul său pot să apară mai multe subprograme, care să apeleze sau nu subprogramul declarat. Directiva **FORWARD** se desparte prin separatorul ";" de antetul procedurii la care se aplică.

Astfel, problema iniţială se rezolvă, deoarece anterior procedurii P1 se declară anticipativ procedura P2 (prin numele, lista de parametrii şi directiva **FORWARD**), după care se declară complet procedura P1, iar în final se declară corpul procedurii P2. În urma acestei ordini a declaraţiilor, procedura P1 ia cunoştinţă de procedura P2, datorită declaraţiei anticipate a acesteia. Fig. 11.30 prezintă macheta declaraţiei **FORWARD** pentru o procedură:

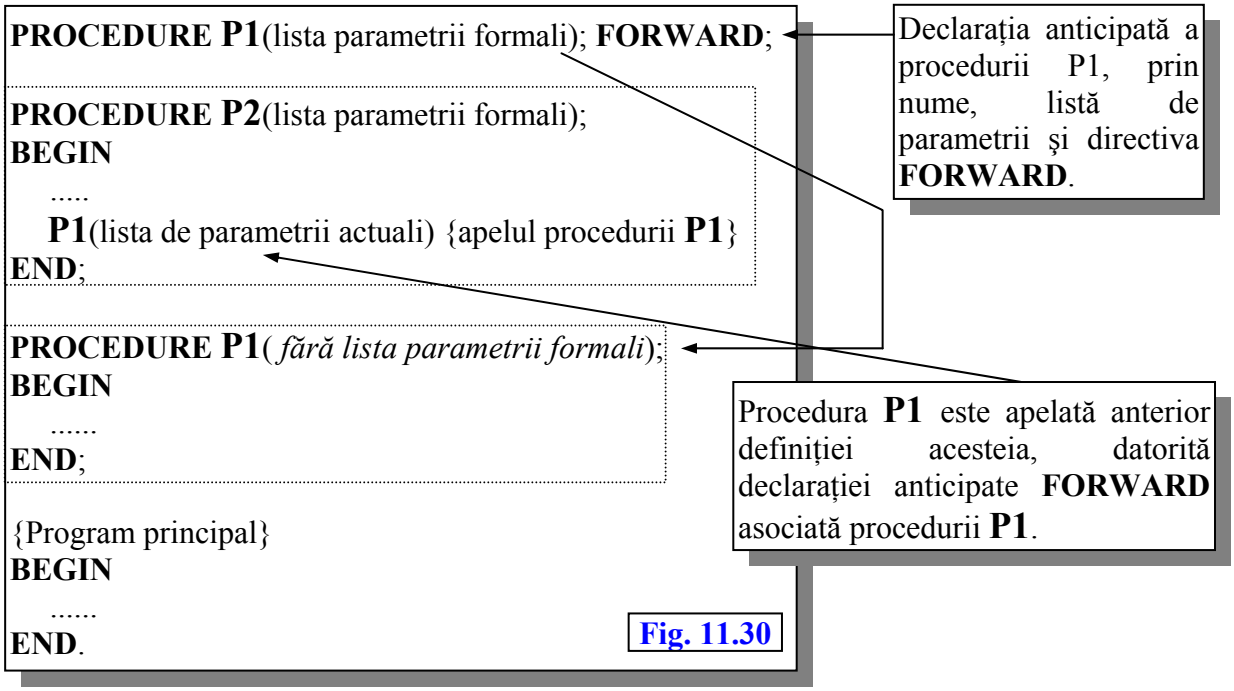


Fig. 11.30

Fig. 11.31a

```
PROGRAM matrice;
{Insumarea a doua matrici patratice cu valori generate aleator}
CONST N=5;
TYPE mat=ARRAY[1..N, 1..N] OF INTEGER;
VAR A,B,C:mat;
   i,j:INTEGER;

PROCEDURE scriumat (nm:INTEGER;x:mat);FORWARD;

PROCEDURE genmat(nm:INTEGER;VAR x:mat);
{Generarea valori aleatoare intr-o matrice
patratica-valori cuprinse intre 0 si "nm"}
VAR i,j : INTEGER;
BEGIN
RANDOMIZE;
FOR i:=1 TO nm DO
  FOR j:=1 TO nm DO
      x[i,j]:=RANDOM(nm+1);
scriumat(nm,x);
END;

PROCEDURE scriumat;
{Afisare valori matrice patratica}
VAR i,j : INTEGER;
BEGIN
FOR i:=1 TO nm DO
   BEGIN
      FOR j:=1 TO nm DO  WRITE(x[i,j],' ');
      {Trecere cursor pe linia urmatoare}
      WRITELN;
   END;
END;
```

Declarație anticipată

Apel anticipat

IC

Pentru exemplificare vom relua problema sumării a două matrici din & 11.7.2, exemplul 11.4. Apelul procedurii **scriumat** din interiorul procedurii **genmat** nu este posibil deoarece procedura **scriumat** este declarată ulterior procedurii **genmat**, fig. 11.16. Programul este modificat în fig. 11.31, astfel ca procedura **scriumat** să poată fi apelată din interiorul procedurii **genmat**, ceea ce a impus următoarele modificări:

- declarația anticipată **FORWARD** a procedurii **scriumat**;
- plasarea instrucțiunii de apel a procedurii **scriumat** în interiorul procedurii **genmat**, după generarea valorilor;

Fig. 11.31b

```
{Program principal}
BEGIN
WRITELN('Matricea A este:');
genmat(N,A);
WRITELN('Matricea B este:');
genmat(N,B);
{Insumare matrici A+B}
FOR i:=1 TO N DO
  FOR j:=1 TO N DO
      C[i,j]:=A[i,j]+B[i,j];
WRITELN('Matricea suma C este:');
scriumat(N,C);
READLN
END.
```

- de asemenea s-au eliminat, din programul principal, instrucțiunile de apel a procedurii **scriumat**, corespunzătoare afișărilor matricilor **A** și **B**, deoarece acestea sunt automat afișate ulterior generării valorilor lor prin procedura **genmat**.

Se remarcă absența parametrilor formali în cadrul blocului propriu-zis al procedurii **scriumat**, fig. 11.31a, aceștia fiind precizați în declarația anticipativă marcată de directiva **FORWARD**.

11.9 TIPURI PROCEDURALE

Pentru limbajul TurboPascal subprogramele reprezintă blocuri de program a căror execuție este activată prin instrucțiunea de apel plasată în programul apelant. Limbajul permite însă şi definirea unor tipuri procedurale, în sensul tratării procedurilor şi funcțiilor ca valori ce pot fi atribuite unor variabile şi transmise ca parametrii de subprograme.

Declarația tipurilor procedurale se poate face în secțiunea **TYPE** prin sintaxa:

TYPE *tip procedural* = **PROCEDURE** (*listă de parametrii*)

TYPE *tip_functional* = **FUNCTION**(*listă de parametrii*): *tip_funct*

unde:

- *tip_procedural* şi *tip_functional* – reprezintă identificatori asociați tipurilor procedurale;
- *tip_funct* – este tipul rezultatului funcției de tip *tip_functional*, care poate fi ordinal, real, string, dar nu poate fi de tip procedural.

Se remarcă faptul că declarația tipurilor procedurale include toate elementele antetelor procedurilor şi funcțiilor, mai puțin denumirea acestora.

Numele parametrilor utilizați în declarația tipurilor procedurale nu au nici o semnificație.

Două tipuri procedurale sunt compatibile dacă au liste de parametrii compatibile (acelaşi număr, tip şi ordine a parametrilor). Două tipuri funcționale sunt compatibile dacă au liste de parametrii compatibile şi au acelaşi tip de rezultat.

Declarația *variabilelor procedurale* se face în secțiunea **VAR**, similar variabilelor normale ale limbajului, astfel:

```
VAR var_proc : tip_procedural;
    var_func: tip_functional;
```

Variabilelor procedurale li se pot atribui valori procedurale, care pot fi alte variabile procedurale sau identificatori de proceduri respectiv funcții. De asemenea, variabilele procedurale pot fi utilizate ca parametrii pentru proceduri sau funcții.

Subprogramele implicate ca tipuri şi valori procedurale trebuie compilate cu directiva **{$F+}** (model de apel depărtat), care permite transferul controlului între segmente de cod diferite. Varianta opusă acestui model este **{$F-}** (model de apel apropiat), care permite transferul controlului la locații diferite în interiorul aceluiaşi segment de cod. Directiva **$F** permite redefinirea modelului aplicat de către compilator.

În cele ce urmează vom exemplifica utilizarea tipurilor şi a variabilelor procedurale.

11.9.1 Integrarea numerică prin metoda Gauss

E 11.11 Program TurboPascal de integrare numerică prin metoda Gauss

Formula generală de cuadratură Gauss este:

$$\int_a^b f(x)dx = h_1 \cdot \sum_{i=1}^{16} a_i \cdot f(x_i)$$

$$x_i = h_1 * xg_i + h_2$$

$$h_1 = \frac{b-a}{2}$$

$$h_2 = \frac{b+a}{2}$$

deci transformă calculul integralei într-o sumă de 16 termeni, în care valorile funcției **f** sunt calculate în abscise $\mathbf{x_i}$ impuse, valorile coeficienților $\mathbf{a_i}$ și $\boldsymbol{xg_i}$ fiind date în tabel. Acești coeficienți sunt numere iraționale, calculați din condiția ca formula de mai sus să fie exactă pentru orice polinom algebric de grad (**2n-1**), unde **n**=16.

i	a_i	xg_i
1.	0.02715245	-0.98940093
2.	0.06225352	-0.94457502
3.	0.09515851	-0.86563120
4.	0.12462897	-0.75540440
5.	0.14959598	-0.61787624
6.	0.16915651	-0.45801677
7.	0.18260341	-0.28160355
8.	0.18945061	-0.09501250
9.	0.18945061	0.09501250
10.	0.18260341	0.28160355
11.	0.16915651	0.45801677
12.	0.14959598	0.61787624
13.	0.12462897	0.75540440
14.	0.09515851	0.86563120
15.	0.06225352	0.94457502
16.	0.02715245	0.98940093

Programul este prezentat în fig. 11.32 a,b,c, iar rezultatele acestuia, pentru 3 tipuri de funcții sunt prezentate în fig. 11.33. Programul s-a aplicat pentru următoarele funcții:

Funcția	**a**	**b**	**Valoarea exactă**	**Valoarea din integrare Gauss (fig. 11.33)**
$f(x) = x^2$ (vezi & 7.4.13)	1	3	8,6667	8,6666663
	1	50	41.666,3333	41.666,3313100
$\int_0^1 x^2\sqrt{1-x^2}\,dx$	0	1	$\frac{\pi}{16} = 0{,}19634954$	0,1963822
$\int_0^1 \frac{1}{1+x^2}\,dx$	0	1	$\frac{\pi}{4} = 0{,}78539816$	0,7853981

Programul principal debutează prin declararea tipului procedural **fct** de tip funcție cu un singur parametru de tip real, rezultatul funcției fiind tot de tip real. Pe baza acestui tip se declară variabila **var_fct** de tip **fct**. De asemenea, din fig. 11.32b, se observă că toate funcțiile **functie**, **functie1** și **functie2** au aceleași caracteristici (1 parametru real și rezultat real) ca și tipul funcțional **fct**, bineînțeles însă diferind prin expresia matematică a funcției. De asemenea declararea funcțiilor este anticipată de directiva de compilare **{$F+}**.

În program se declară și variabilele globale **lmin**, **lmax**, **integrala** ca fiind reale.

Fig. 11.32a

```
PROGRAM integrare_Gauss;
TYPE fct=FUNCTION(x:REAL):REAL;
VAR lmin, lmax, integrala:REAL;
      var_fct:fct;
PROCEDURE gauss(f:fct;a,b:REAL;VAR suma_int:REAL);
TYPE vct=ARRAY[1..16] OF REAL;
VAR xg, ag:vct;
   i: INTEGER;
   h1,h2,xi:REAL;
BEGIN
ag[9] :=0.18945061  ; ag[10]:=0.18260341 ;
ag[11]:=0.16915651 ; ag[12]:=0.14959598 ;
ag[13]:=0.12462897 ; ag[14]:=0.09515851 ;
ag[15]:=0.06225352 ; ag[16]:=0.02715245 ;

xg[9] :=0.09501250 ; xg[10]:=0.28160355 ;
xg[11]:=0.45801677 ; xg[12]:=0.61787624 ;
xg[13]:=0.75540440 ; xg[14]:=0.86563120 ;
xg[15]:=0.94457502 ; xg[16]:=0.98940093 ;
FOR i:=1 TO 8 DO
   BEGIN
IC    ag[i]:=ag[17-i]; xg[i]:=-xg[17-i];
   END;
h1:=(b-a)/2; h2:=(b+a)/2; suma_int:=0;
FOR i:=1 TO 16 DO
   BEGIN
IC    xi:=h1*xg[i]+h2;
      suma_int:=suma_int+ag[i]*f(xi)
   END;
suma_int:=suma_int*h1
END;
```

Declarații
tip procedural
variabila procedurală

OBS. La scrierea programului în editorul Turbo Pascal se va respecta succesiunea din fig. 11.32 literele a, b, c.

Fig. 11.32b

```
{$F+}
FUNCTION functia(x:REAL):REAL;
BEGIN
  functia := x*x
END;

FUNCTION functia1(x:REAL):REAL;
BEGIN
  functia1:=x*x*SQRT(1-x*x)
END;

FUNCTION functia2(x:REAL):REAL;
BEGIN
  functia2:=1/(1+x*x)
END;
{$F-}
```

Directive de compilare

Fig. 11.32c

```
BEGIN   {Program principal}
lmin:=1.0; lmax:=3.0;
gauss(functia,lmin, lmax, integrala);
WRITELN('Functia 1-3  Solutia=',integrala:14:7);

lmin:=1.0; lmax:=50.0;
gauss(functia,lmin, lmax, integrala);
WRITELN('Functia 1-50 Solutia=',integrala:14:7);

var_fct := functia1;
gauss(var_fct,0.0, 1.0, integrala);
WRITELN('Functia1 0-1 Solutia=',integrala:14:7);

var_fct := functia2;
gauss(var_fct,0.0, 1.0, integrala);
WRITELN('Functia2 0-1 Solutia=',integrala:14:7);
READLN
END.
```

Procedura **gauss** este dedicată sumării gaussiene conform relației din enunțul problemei. Se declară tipul de date vectorial **vct** cu 16 componente, pe baza căruia se declară variabilele vector **ag** și **xg** în care se vor memora coeficienții de integrare tabelați mai sus. De asemenea se mai declară variabilele locale **i** de tip **INTEGER** și **h1**, **h2** și **xi** de tip **REAL**. Valorile coeficienților pozițiilor 9..16 sunt depozitate prin atribuire în vectori, restul valorilor (1 la 9) se generează prin simetrie în primul ciclu **FOR**.

Fig. 11.33

```
Functia 1-3  Solutia=     8.6666663
Functia 1-50 Solutia= 41666.3313100
Functia1 0-1 Solutia=     0.1963822
Functia2 0-1 Solutia=     0.7853981
```

În continuare, procedura **gauss** calculează valorile constantelor **h1** şi **h2**, iniţializează variabila de sumare **suma_int** cu 0. În al doilea ciclu **FOR**, parcurs pentru 16 valori, se determină valoarea abscisei **xi**, în care se va calcula valoarea funcţiei, transmisă procedurii prin intermediul parametrului formal **f** de tip **fct**. De asemenea, procedura mai are ca parametrii formali limitele de integrare **a**, **b** de tip real; procedura returnează programului principal valoarea calculată a sumei (echivalentul numeric al integralei) prin variabila reală **suma_int**, declarată ca parametru formal variabilă.

Apelurile acestei proceduri sunt plasate în programul principal, numărul de apeluri fiind patru, din care primele două, cu limite de integrare diferite, pentru funcţia **functia** specificată prin nume ca parametru actual, iar ultimele două pentru funcţia **functia1** respectiv **functia2**, specificate prin intermediul variabilei **var_fct**. Pentru primele două apeluri, limitele de integrare sunt specificate prin parametrii actuali (variabilele **lmin** şi **lmax**), iar pentru ultimele două apeluri, direct prin valori numerice specificate în instrucţiunea de apel a procedurii. Variabila **integrala** colectează pentru fiecare apel în parte valoarea integralei calculată în procedura **gauss**, după fiecare apel urmând afişarea valorii calculate a integralei.

Programul a fost aplicat pentru integrarea numerică a următoarelor funcţii, care impun suplimentările programului cu instrucţiunile din fig. 11.34 a şi b:

Funcţia	**a**	**b**	**Valoarea exactă**	**Valoarea din integrare Gauss**
$\int_0^{\pi/2} SIN(x)dx$	0	$\frac{\pi}{2}$	1	1,000000
$\int_0^{2\pi} x \cdot SIN(x)dx$	0	2π	$-2\pi = -6{,}2831853$	-6,2831850
$\int_{-1}^{1} \sqrt{1-x^2}\,dx$	-1	1	$\frac{\pi}{2} = 1{,}5707963$	1,5709803
$\int_0^{\pi} \ln[1-8\cdot COS(x)+16]\,dx$	0	π	$2\pi\ln(4) = 8.7103444$	8.7103440

```
FUNCTION functia3(x:REAL):REAL;
BEGIN
  functia3:=SIN(x)
END;
FUNCTION functia4(x:REAL):REAL;
BEGIN
  functia4:=x*SIN(x)
END;
FUNCTION functia5(x:REAL):REAL;
BEGIN
  functia5:=SQRT(1-x*x)
END;
FUNCTION functia6(x:REAL):REAL;
BEGIN
  functia6:=LN(1-8*COS(x)+16)
END;
```

Fig. 11.34a

Fig. 11.34b

```
gauss(functia3,0.0, PI/2, integrala);
WRITELN('Functia3 Solutia=',integrala:14:7);

gauss(functia4,0.0, 2*PI, integrala);
WRITELN('Functia4 Solutia=',integrala:14:7);

gauss(functia5,-1.0, 1.0, integrala);
WRITELN('Functia5 Solutia=',integrala:14:7);

gauss(functia6,0.0, PI, integrala);
WRITELN('Functia6 Solutia=',integrala:14:7);
```

Formula de integrare numerică a lui Gauss îmbină un înalt grad de precizie raportat la numărul relativ mic de puncte prin care se calculează integrala şi cu simplitatea deosebită a procedeului de calcul.

BIBLIOGRAFIE

1. BĂLĂNESCU, T., GAVRILĂ, S., GEORGESCU, H. & COLECTIV – ***Pascal şi TurboPascal***, Editura Tehnică, Bucureşti, 1992.
2. BEU, T. – ***Analiză numerică în Turbo Pascal***, Editura MicroInformatica, Cluj-Napoca, 1992.
3. CHIOREAN, L. , CHIOREAN, M. - ***PC iniţiere hard şi soft***, Editura Albastră, Cluj-Napoca, 2000.
4. CRISTEA, V. & COLECTIV – ***Dicţionar de informatică***, Editura Ştiinţifică şi Enciclopedică, Bucureşti, 1981.
5. DOGARU, O., BOCŞAN, Gh. & COLECTIV – ***Informatică. Tematică pentru definitivare şi grad***, Editura de Vest, Timişoara, 1998
6. GROZA, D. – ***Programarea calculatoarelor***, Tipografia Universităţii "Eftimie Murgu", Reşiţa, 1995.
7. IOSEP, M., IOSEP, C. – ***Lecţii de programare***, Editura Semnalul, Bucureşti, 1991.
8. IOSEP, M., IOSEP, C. – ***Programare în Pascal***, Editura Semnalul, Bucureşti, 1992.
9. KASSERA, W., KASSERA, V. – ***TurboPascal 6.0***, Editura TipoMur, Târgu Mureş, 1992.
10. KOVACS, S., KOVACS, A. – ***Ghid de utilizare TURBO PASCAL 7.0***, Editura Albastră, Cluj-Napoca, 1995.
11. KOVACS, S. – ***Memento unit-uri TURBO PASCAL 5.5***, Editura Romanian Software Comp., Cluj-Napoca, 1991.
12. KOVACS, S. – ***Memento TURBO PASCAL 5.5***, Editura Romanian Software Comp., Cluj-Napoca, 1991.
13. LARIONESCU, D. – ***Metode numerice***, Editura Tehnică, Bucureşti, 1989.
14. LIVOVSCHI, L. – ***Bazele informaticii***, Editura Albatros, Bucureşti, 1979.
15. LIVOVSCHI, L. – ***Bazele informaticii***, Editura Didactică şi Pedagogică, Bucureşti, 1981.
16. LUPULESCU, M., MUNTEAN, M. & COLECTIV – ***Bazele computerelor. Hard & soft,*** Editura Mirton, Timişoara, 1999.
17. MARINESCU, D., TRANDAFIRESCU, M. – ***PC Manualul începătorului***, Editura Teora, Bucureşti, 1994.
18. MUNTEANU, FL., IONESCU, T. & COLECTIV – ***Programarea calculatoarelor***, Editura Didactică şi Pedagogică, Bucureşti, 1995.
19. ROŞCULEŢ, M., BALEA, P., MOLDOVEANU, Ş. – ***Programarea şi utilizarea maşinilor de calcul şi elemente de calcul numeric şi informatică***, Editura Didactică şi Pedagogică, Bucureşti, 1980.
20. PÂRV, B., VANCEA, A. – ***Fundamentele limbajelor de programare***, Editura Albastră, Cluj-Napoca, 1996.
21. PĂTRUŢ, B. – ***Algoritmi şi limbaje de programare***, Editura Teora, Bucureşti, 1998.
22. SANDOR, K. – ***TurboPascal 6.0. Ghid de utilizare***, Editura MicroInformatica, Cluj-Napoca, 1992.
23. SIMIONESCU, I., DRANGA, M., MOISE, V. – ***Metode numerice în tehnică. Aplicaţii în FORTRAN***, Editura Tehnică, Bucureşti, 1995.
24. TOMA, M., ODĂGESCU, I. – ***Metode numerice şi subrutine***, Editura Tehnică, Bucureşti, 1980

25. TOVISSI, L., VODĂ, V. – ***Metode statistice. Aplicații în producție***, Editura Ştiințifică şi Enciclopedică, Bucureşti, 1982
26. VLADA, M., POSEA, A, NISTOR, I, CONSTANTINESCU, C. – ***Grafică pe calculator în limbajele PASCAL şi C***, Editura Tehnică, Bucureşti, 1992.
27. VRACIU, G., POPA, A. – ***Metode numerice cu aplicații în tehnica de calcul***, Editura Scrisul Românesc, Craiova, 1982.
28. VĂLEANU, I,HÎNCU, M. – ***Elemente de statistică generală***, Editura Litera, Bucureşti, 1990
29. ******** - ***Gazeta de Informatică***, Colecția revistei pe anii 1992-1993, Imprimeria Ardealul, Cluj-Napoca

Anexa 1

În **Anexa 1** sunt tabelate algoritmii şi programele incluse în prezenta lucrare, în ordinea apariției lor, cu referire la capitolul, exemplul şi figura corespunzătoare.

Anexa 1 - MEMORATOR ALGORITMI ŞI PROGRAME

Algoritm	Cap.	Ex.	Fig.	Observații
Calculul modulului unui număr real	2	2.1	2.2	Limbaj natural
Rezolvarea ecuației de grad I	2	2.2	2.3	Limbaj natural
Calculul modulului unui număr real	2	2.3	2.5	Schemă logică
Rezolvarea ecuației de grad I	2	2.4	2.6	Schemă logică
Calculul modulului unui număr real	2	2.5	2.8	Limbaj pseudocod
Rezolvarea ecuației de grad I	2	2.6	2.9	Limbaj pseudocod
Calcul arie triunghi prin formula lui Heron	2	2.7	2.11	Limbaj pseudocod
			2.12	Schemă logică
Rezolvarea ecuației de grad II	2	2.8	2.17	Limbaj pseudocod
			2.18	Schemă logică
Calcul radical prin recurență – varianta ciclu cu test inițial	2	2.9	2.21	Limbaj pseudocod
			2.22	Schemă logică
Calcul radical prin recurență – varianta ciclu cu test final	2	2.10	2.25	Limbaj pseudocod
			2.26	Schemă logică
Determinarea maximului a N numere	2	2.11	2.29	Limbaj pseudocod
			2.30	Schemă logică
Calcul mediei aritmetice a trei numere	2	2.12	2.31	Limbaj pseudocod
			2.32	Schemă logică
Inversarea valorilor a două variabile	2	2.13	2.33	Limbaj pseudocod
			2.34	Schemă logică
Minim dintre două numere	2	2.14	2.35	Limbaj pseudocod
			2.36	Schemă logică
Maxim dintre două numere	2		2.37	Limbaj pseudocod
			2.38	Schemă logică
Intersecția a două intervale	2	2.15	2.40	Limbaj pseudocod
			2.41	Schemă logică
Minim dintre patru numere (Varianta 1)	2	2.16	2.42	Limbaj pseudocod
			2.43	Schemă logică
Minim dintre patru numere (Varianta 2)			2.44	Limbaj pseudocod
Calcul suma / produs a două numere	2	2.17	2.45	Limbaj pseudocod
			2.46	Schemă logică
Rezolvare sistem 2 ecuații cu 2 necunoscute	2	2.18	2.47	Limbaj pseudocod
			2.48	Schemă logică
Afişare numere impare până la un număr N impus – varianta nestructurată	2	2.19	2.49	Limbaj pseudocod
			2.50	Schemă logică

Anexa 1 - MEMORATOR ALGORITMI ŞI PROGRAME

Algoritm	Cap.	Ex.	Fig.	Observații
Afişare numere impare până la un număr N impus – varianta ciclu cu test inițial	2	2.19	2.51	Schemă logică
Afişare numere impare până la un număr N impus – varianta ciclu cu test final	2	2.19	2.52	Schemă logică
Suma a N numere. Algoritmul de sumare.	2	2.20	2.53	Limbaj pseudocod
			2.54	Schemă logică
Suma numerelor pozitive şi negative dintr-un şir de N numere.	2	2.21	2.55	Limbaj pseudocod
			2.56	Schemă logică
Produsul a N numere. Algoritmul produsului.	2	2.22	2.57	Limbaj pseudocod
			2.58	Schemă logică
Contorizarea numerelor pozitive, nule şi negative dintr-un şir de N numere. Algoritmul contorizării.	2	2.23	2.59	Limbaj pseudocod
			2.60	Schemă logică
Sumare serie numerică	2	2.24	2.61	Limbaj pseudocod
			2.62	Schemă logică
Calcul mediei aritmetice a trei numere	5	5.5	5.1	Program TurboPascal
Calcul arie triunghi prin formula lui Heron	5	5.6	5.3	Program TurboPascal
Inversarea valorilor a două variabile	5	5.7	5.5	Program TurboPascal
Calcul perimetru şi arie cerc	5	5.8	5.7	Program TurboPascal
Calculul modulului unui număr real	6	6.1	6.1	Program TurboPascal
Rezolvarea ecuației de grad I	6	6.2	6.3	Program TurboPascal
Rezolvarea ecuației de grad II	6	6.3	6.5	Program TurboPascal
Minim dintre două numere	6	6.4	6.7	Program TurboPascal
Maxim dintre două numere	6	6.4	6.9	Program TurboPascal
Intersecția a două intervale	6	6.5	6.11	Program TurboPascal
Minim dintre patru numere (Varianta 1)	6	6.6	6.13	Program TurboPascal
Minim dintre patru numere (Varianta 2)	6	6.6	6.14	Program TurboPascal
Rezolvare sistem 2 ecuații cu 2 necunoscute	6	6.7	6.16	Program TurboPascal
Identificare număr introdus de la tastatură	6	6.8	6.18	Program TurboPascal
Identificare caracter introdus de la tastatură	6	6.9	6.20	Program TurboPascal
Calcul radical prin recurență – varianta ciclu cu test inițial	7	7.1	7.1	Program TurboPascal
Calcul radical prin recurență – varianta ciclu cu test final	7	7.2	7.3	Program TurboPascal
Determinarea maximului a N numere	7	7.3	7.5	Program TurboPascal
Afişare numere impare până la un număr N impus – varianta ciclu cu test inițial	7	7.4	7.7	Program TurboPascal
Afişare numere impare până la un număr N impus – varianta ciclu cu test final	7		7.8	Program TurboPascal
Suma a N numere citite succesiv	7	7.5	7.10	Program TurboPascal

Anexa 1 - MEMORATOR ALGORITMI ŞI PROGRAME

Algoritm	Cap.	Ex.	Fig.	Observații
Suma numerelor pozitive şi negative dintr-un şir de N numere.	7	7.6	7.12	Program TurboPascal
Produsul a N numere	7	7.7	7.14	Program TurboPascal
Contorizarea numerelor pozitive, nule şi negative dintr-un şir de N numere.	7	7.8	7.16	Program TurboPascal
Sumare serie numerică	7	7.9	7.18	Program TurboPascal
Produs a două numere întregi prin adunări repetate	7	7.10	7.21	Program TurboPascal
Împărțire a două numere întregi prin scăderi repetate	7	7.11	7.23	Program TurboPascal
Căutare numere cubice în intervalul 100-499	7	7.12	7.25	Program TurboPascal
Integrare numerică prin metoda dreptunghiurilor	7	7.13	7.28	Program TurboPascal
Media aritmetică unui şir de numere	7	7.14	7.30	Program TurboPascal
Sumare numere cu ieşire forțată din ciclu prin procedura BREAK	7	7.15	7.33	Program TurboPascal
Afişare numere impare intre 1 şi 10 – exemplificare procedura CONTINUE	7	7.16	7.35	Program TurboPascal
Contorizare numere pozitive, negative şi nule din N numere - varianta programării cu instrucțiunea GOTO	8	8.1	8.2	Program TurboPascal
Contorizare numărului de apariții a fețelor unui zar dintr-un total de N aruncări	8	8.2	8.4	Program TurboPascal
Citirea componentelor unui vector şi afişarea valorilor sale	9	9.1	9.6	Program TurboPascal
Afişarea în ordine inversă a valorilor unui vector generate aleator	9	9.2	9.8	Program TurboPascal
Inversarea valorilor unui vector în alt vector	9	9.3	9.10	Program TurboPascal
Inversarea valorilor unui vector în acelaşi vector	9	9.4	9.12	Program TurboPascal
Minimul şi maximul unui vector	9	9.5	9.14	Program TurboPascal
Sumarea componentelor unui vector	9	9.6	9.16	Program TurboPascal
Contorizarea componentelor unui vector	9	9.7	9.18	Program TurboPascal
Căutarea secvențială într-un vector	9	9.8	9.20	Program TurboPascal
Ordonarea crescătoare a unui vector prin metoda selecției minimului	9	9.9	9.22	Program TurboPascal
Ordonarea crescătoare a unui vector prin metoda BUBBLE-SORT	9	9.10	9.24	Program TurboPascal
Contorizare în vector a aparițiilor fețelor la	9	9.11	9.26	Program TurboPascal

Anexa 1 - MEMORATOR ALGORITMI ŞI PROGRAME

Algoritm	Cap.	Ex.	Fig.	Observații
aruncarea unui zar				
Sumare şi produse de elemente în matrice	9	9.12	9.28	Program TurboPascal
Inversare prima şi ultima linie şi coloană într-o matrice	9	9.13	9.32	Program TurboPascal
Maxime şi sume pe coloană într-o matrice	9	9.14	9.34	Program TurboPascal
Desfăşurarea unei matrici într-un vector	9	9.15	9.37	Program TurboPascal
Suma elementelor marginale ale matricii	9	9.16	9.40	Program TurboPascal
Căutare numere cubice în intervalul 100-499 - varianta programării prin şiruri	10	10.1	10.2	Program TurboPascal
Căutare numere automorfice în intervalul 1-1000	10	10.2	10.4	Program TurboPascal
Construire şir echivalent unui şir dat	10	10.3	10.6	Program TurboPascal
Descompunere propoziție în cuvinte	10	10.4	10.8	Program TurboPascal
Calculul unei suprafețe dreptunghiulare prin însumarea ariilor dreptunghiurilor elementare- procedură cu parametrii globali	11	11.1	11.9	Program TurboPascal
Calculul unei suprafețe dreptunghiulare prin însumarea ariilor dreptunghiurilor elementare- procedură cu lista de parametrii	11	11.2	11.11	Program TurboPascal
Ordonarea crescătoare a două numere	11	11.3	11.14	Program TurboPascal
Sumarea a două matrici	11	11.4	11.16	Program TurboPascal
Calcul combinări– fără funcție (fig. 11.18a)	11	11.5	11.18	Program TurboPascal
Calcul combinări– cu funcție (fig. 11.18b)	11	11.5	11.18	Program TurboPascal
Generare pătrate magice	11	11.6	11.20	Program TurboPascal
Ordonarea crescătoare diagonală principală matrice pătratică	11	11.7	11.22	Program TurboPascal
Generare număr printr-o logică impusă	11	11.8	11.24	Program TurboPascal
Eliminarea valorilor eronate dintr-un şir	11	11.9	11.26	Program TurboPascal
Generare pătrate greco-latine	11	11.10	11.28	Program TurboPascal
Integrare numerică prin metoda Gauss	11	11.11	11.32	Program TurboPascal

Anexa 2

Lista erorilor frecvente de compilare

Cod eroare	Eroare	Cauze posibile
3	Unknown Identifier	• Identificator nedeclarat • Cuvânt cheie rezervat al limbajului TurboPascal scris greşit • Un şir de caractere finalizat prin separatorul ‘ nu are marcat începutul prin caracterul ‘.
4	Duplicate Identifier	Identificator declarat de două ori în secţiunea declaraţiilor
8	String constant exceeds line	Succesiunea de caractere nu se finalizează prin separatorul ‘.
10	Unexpected end of file	Programul nu se finalizează cu instrucţiunea **END** sau lipseşte marcatorul **.** după **END**.
11	Line too long	Numărul de caractere din instrucţiune depăşeşte limita maximală de 127 de caractere.
25	Invalid string length	Depăşirea lungimii admise pentru tipul **STRING** (255).
36	Begin Expected	Cuvânt cheie **BEGIN** nedeclarat pentru marcarea începutului unui bloc.
40	Boolean expression expected	Expresia nu oferă un rezultat de tip logic: • În condiţia din instrucţiunea **IF** se utilizează operatorul de atribuire **:=** în loc de cel relaţional =.
42	Error in expression	Expresie incorectă: • Expresia matematică nu este scrisă corect sintactic sau din punct de vedere al operatorilor utilizaţi • Lista argumentelor instrucţiunii **WRITE** sau **WRITELN** se finalizează cu **,**
50	DO expected	Cuvântul cheie **DO** lipseşte din instrucţiunea **WHILE** sau **FOR**.
64	Cannot Read or Write variable of this type	Se încearcă citirea sau scrierea unei variabile incompatibile ca tip: • Pot fi citite numai variabile de tip **CHAR**, **INTEGER**, **REAL** sau **STRING**; • Pot fi scrise numai variabile de tip **CHAR**, **INTEGER**, **REAL, STRING** şi **BOOLEAN**.
85	“;” Expected	Separator **;** inexistent pe una din liniile anterioare.
89	“)” Expected	Paranteză neânchisă
91	“:=” Expected	Instrucţiunea de atribuire conţine semnul = dar nu conţine semnul **:** sau invers
97	Invalid **FOR** control variable	Se utilizează ca şi contor de ciclul **FOR** o variabila incompatibilă (real).
113	Error in statement	Separatorul **;** este plasat incorect anterior cuvântului cheie **ELSE** din instrucţiunea **IF**.

Anexa 3
Coduri asociate caracterelor (Cod ASCII)

Anexa conține caracterele codului ASCII, mai puține primele 32 caractere neimprimabile (codificate de la 0 la 31).

Cod	Car.	Cod	Car.	Cod	Car.	Cod	Car.	Cod	Car.	Cod	Car.
32		70	F	108	l	146	’	184	¸	222	Þ
33	!	71	G	109	m	147	“	185	¹	223	ß
34	"	72	H	110	n	148	”	186	º	224	à
35	#	73	I	111	o	149	•	187	»	225	á
36	$	74	J	112	p	150	–	188	¼	226	â
37	%	75	K	113	q	151	—	189	½	227	ã
38	&	76	L	114	r	152	˜	190	¾	228	ä
39	'	77	M	115	s	153	™	191	¿	229	å
40	(	78	N	116	t	154	š	192	À	230	æ
41	)	79	O	117	u	155	›	193	Á	231	ç
42	*	80	P	118	v	156	œ	194	Â	232	è
43	+	81	Q	119	w	157		195	Ã	233	é
44	,	82	R	120	x	158	ž	196	Ä	234	ê
45	-	83	S	121	y	159	Ÿ	197	Å	235	ë
46	.	84	T	122	z	160		198	Æ	236	ì
47	/	85	U	123	{	161	¡	199	Ç	237	í
48	0	86	V	124	|	162	¢	200	È	238	î
49	1	87	W	125	}	163	£	201	É	239	ï
50	2	88	X	126	~	164	¤	202	Ê	240	ð
51	3	89	Y	127		165	¥	203	Ë	241	ñ
52	4	90	Z	128	€	166	¦	204	Ì	242	ò
53	5	91	[	129		167	§	205	Í	243	ó
54	6	92	\	130	‚	168	¨	206	Î	244	ô
55	7	93	]	131	ƒ	169	©	207	Ï	245	õ
56	8	94	^	132	„	170	ª	208	Đ	246	ö
57	9	95	_	133	…	171	«	209	Ñ	247	÷
58	:	96	`	134	†	172	¬	210	Ò	248	ø
59	;	97	a	135	‡	173	-	211	Ó	249	ù
60	<	98	b	136	ˆ	174	®	212	Ô	250	ú
61	=	99	c	137	‰	175	¯	213	Õ	251	û
62	>	100	d	138	Š	176	°	214	Ö	252	ü
63	?	101	e	139	‹	177	±	215	×	253	ý
64	@	102	f	140	Œ	178	²	216	Ø	254	þ
65	A	103	g	141		179	³	217	Ù	255	ÿ
66	B	104	h	142	Ž	180	´	218	Ú		
67	C	105	i	143		181	µ	219	Û		
68	D	106	j	144		182	¶	220	Ü		
69	E	107	k	145	‘	183	·	221	Ý		