

LIMBAJUL PASCAL

Adina Corici
Florin Mânz
Elena Simulescu
Florinel Șerban



LIBRIS

LIMBAJUL PASCAL

Corina Corici
Dorin Mânz
Adriana Simulescu
Marinel Șerban



CLUJ 1991

COORDONATOR: CLARA IONESCU

**Control științific: lect.dr.FLORIAN BOIAN
prof.DOINA RANCEA**

Cartea a fost elaborată astfel:

Corina Corici: cap.II,III
Dorin Mânz: cap.VI
Adriana Simulescu: cap.IV
Marinel Șerban: cap.I,V,VII,VIII

Redactor: Clara Ionescu

Tehnoredactare computerizată: Cristina Tămaș

Grafica: Baka József

Grafica computerizată: Eugen Ionescu și Adrian Mosu

Coperta: Sipos László

Bun de tipar: 8.05.1991.
Coli de tipar 14
ISBN 973-95118-0-5

CUPRINS

CAPITOLUL I. NOTIUNI INTRODUCTIVE

1.1. Evoluția limbajelor de programare. Principii ale programării structurate.....	5
1.2. Structura programelor PASCAL.....	7
1.3. Descrierea sintaxei limbajului PASCAL cu ajutorul diagramelor.....	8

CAPITOLUL II. ELEMENTE DE BAZA ALE LIMBAJULUI PASCAL

2.1. Vocabularul limbajului PASCAL.....	11
2.1.1. Setul de caractere.....	11
2.1.2. Identificatori.....	11
2.1.3. Separatori și comentarii.....	12
2.2. Constantele limbajului PASCAL.....	13
2.2.1. Constante întregi.....	13
2.2.2. Constante reale.....	13
2.2.3. Șir de caractere.....	14
2.2.4. Constante desemnate prin identificatori.....	14
2.3. Definirea constantelor utilizator.....	15
2.4. Noțiunea de tip.....	15
2.5. Variabilele limbajului PASCAL și declararea lor....	16
2.6. Expresii. Funcții standard.....	16
2.7. Operații de citire/scriere simple.....	19

CAPITOLUL III. STRUCTURI FUNDAMENTALE DE CONTROL IN LIMBAJUL PASCAL

3.1. Structura secvențială.....	23
3.1.1. Instrucțiunea de atribuire.....	23
3.1.2. Instrucțiunea compusă.....	24
3.2. Structura alternativă.....	26
3.2.1. Instrucțiunea IF.....	26
3.2.2. Instrucțiunea CASE.....	30
3.3. Structura repetitivă.....	34
3.3.1. Instrucțiunea WHILE.....	35
3.3.2. Instrucțiunea REPEAT.....	37
3.3.3. Instrucțiunea FOR.....	40
3.4. Instrucțiunea GOTO.....	43

CAPITOLUL IV. TIPURI DE DATE

4.1. Tipuri simple.....	48
4.1.1. Tipuri simple standard.....	49
4.1.2. Tipuri simple definite de utilizator.....	57
4.2. Tipuri structurate.....	60
4.2.1. Tipul tablou.....	60
4.2.2. Tipul înregistrare.....	67
4.2.3. Tipul mulțime.....	71

CAPITOLUL V. SUBPROGRAME

5.1. Domeniul de valabilitate al identificatorilor și	
---	--

	etichetelor.....	76
5.2.	Dezvoltarea programelor PASCAL.....	77
5.3.	Proceduri.....	79
5.3.1.	Declararea procedurilor. Parametri formali, parametri efectivi.....	79
5.3.2.	Proceduri standard de intrare/ieșire.....	81
5.3.3.	Reprezentarea datelor în memorie.....	93
5.3.4.	Accesul direct al memoriei.....	94
5.3.5.	Proceduri HP4TM pentru lucrul cu caseta.....	98
5.3.6.	Alocarea dinamică a memoriei.....	100
5.3.7.	Proceduri definite în program.....	100
5.4.	Funcții.....	107
5.4.1.	Declararea funcțiilor.....	107
5.4.2.	Funcții standard specifice limbajului PASCAL HP4TM.....	108
5.4.3.	Funcții definite în program.....	120
5.4.4.	Stocarea datelor în memorie.....	122
5.5.	Recursivitate.....	126
5.5.1.	Generalități.....	126
5.5.2.	Program recursiv de ilustrare a mecanismului recursivității.....	127
5.5.3.	Utilizarea recursivității.....	129
5.5.4.	Tipuri de algoritmi recursivi.....	132

CAPITOLUL VI. ALOCAREA DINAMICĂ A MEMORIEI

6.1.	Tipul referință (pointer).....	149
6.2.	Structuri de date de tip listă liniară.....	152
6.2.1.	Implementarea listelor cu ajutorul tipului pointer (liste înlănțuite).....	152
6.2.2.	Tehnici de inserare a nodurilor.....	153
6.2.3.	Tehnici de suprimare (ștergere) a nodurilor unei liste.....	159
6.2.4.	Traversarea unei liste înlănțuite.....	161
6.2.5.	Aplicații ale listelor înlănțuite.....	165
6.2.6.	Liste dublu înlănțuite.....	168
6.2.7.	Structuri de date de tip stivă.....	172
6.2.8.	Structuri de date de tip coadă.....	174
6.2.9.	Structuri de date de tip arbore binar.....	176

CAPITOLUL VII. ELEMENTE DE GRAFICĂ ȘI SUNET SPECIFICE LIMBAJULUI PASCAL HP4TM

7.1.	Generalități.....	194
7.2.	Variabile globale.....	194
7.3.	Procedurile pachetului TURTLE.....	195

CAPITOLUL VIII. MIC MANUAL DE OPERARE PASCAL HP4TM

8.1.	Introducere.....	208
8.1.1.	Punerea în funcțiune.....	208
8.1.2.	Compilarea și lansarea în execuție.....	209
8.2.	Editorul.....	210
8.2.1.	Introducere în editor.....	210
8.2.2.	Comenzile editorului.....	210
8.3.	Opțiuni ale compilatorului.....	215

ANEXA 1. CUVINTE SI IDENTIFICATORI PREDEFINITI.....218

ANEXA 2. HP4TM -COMPILATORUL PASCAL ZX Spectrum - MEMORATOR219

Bibliografie.....	222
-------------------	-----

I. NOȚIUNI INTRODUCTIVE

1.1. EVOLUȚIA LIMBAJELOR DE PROGRAMARE. PRINCIPII ALE PROGRAMARII STRUCTURATE

Limbajele de programare sînt mijloace de comunicare între utilizator și sistemul de calcul. Prin intermediul lor, omul transmite calculatorului ordinele necesare pentru executarea unei anumite operații. Descrierea cu ajutorul unui limbaj de programare a etapelor necesare rezolvării unei anumite probleme se numește *program*. Astfel, un program apare ca o succesiune de comenzi transmise calculatorului pe care acesta le execută. O asemenea comandă se numește *instrucțiune*. Este evident că programul transmis calculatorului pentru a fi executat nu este altceva decît descrierea, în limbajul de programare ales, a algoritmului de rezolvare a problemei respective.

Se știe că un calculator nu "cunoaște" decît un singur limbaj - limbajul calculatorului, numit *cod-mașină*. Primele generații de calculatoare erau programate direct în cod-mașină, lucru deosebit de dificil de realizat. Odată cu apariția necesităților tot mai mari de prelucrare a informațiilor și odată cu progresele înregistrate în domeniul construcției de calculatoare, s-a impus un salt calitativ în ceea ce privește programarea calculatoarelor și anume trecerea de la programarea în cod-mașină la *programarea simbolică*. În acest fel programatorul se preocupă mai mult de metoda de rezolvare a problemei concrete, fără a se mai preocupa de amănunte legate de calculator. În acest fel, pe lîngă o comoditate sporită în scrierea programelor, această schimbare a însemnat și cîștigarea unei independențe a programării față de caracteristicile fizice ale calculatorului.

În evoluția limbajelor de programare simbolice există cîteva repere care trebuie menționate: anul 1955 marchează apariția limbajului FORTRAN, destinat calculelor tehnico-științifice cu caracter numeric (*FORM*ula *TRAN*slation = traducerea formulelor). În prezent se utilizează pe calculatoarele mari existente în țară varianta FORTRAN IV, iar pe minicalculatoare, FORTRAN 77, care diferă mult de versiunea inițială.

Un alt punct de referință în evoluția limbajelor de programare simbolice este anul 1960; în acest an un grup de cercetători, coordonat de Peter Naur, a definit un limbaj de programare, numit ALGOL-60 (*ALGO*rithmic *LAN*guage = limbaj algoritmic), a cărui definiție se remarcă prin precizie și o sintaxă complet formalizată. Deși limbajul ALGOL a avut o utilizare relativ restrînsă, multe din conceptele introduse cu prilejul definirii limbajului sînt folosite și astăzi, ALGOL-60 fiind un model pentru proiectanții altor limbaje de programare. Tot în 1960 a apărut și prima versiune a limbajului COBOL (*CO*mmon *B*ussiness *O*riented *L*anguage = limbaj orientat pe probleme economice).

Intre anii 1960-1970 au apărut nenumărate limbaje de programare, unele rezistind timpului, fiind folosite mai mult sau mai puțin, altele, din contră, dispărind la fel de rapid cum au apărut.

Dintre limbajele care au rezistat timpului, care au adus ceva nou și care sînt folosite în prezent din ce în ce mai mult, remarcăm limbajul PASCAL, definit în anul 1971 de Niklaus Wirth.

Apariția acestui limbaj este un rezultat al conceptelor dezvoltate ca urmare a crizei ce caracteriza domeniul programării calculatoarelor la sfîrșitul anilor '60. În această perioadă problemele abordate de către programatori au devenit tot mai complexe. Programele construite pentru rezolvarea acestor probleme au devenit și ele, în mod evident, complicate, greu accesibile chiar și pentru creatorii lor. Depanarea și modificarea unor astfel de programe a devenit extrem de complicată. Acest fenomen a apărut datorită absenței unor principii clare, care să impună o "disciplină a programării". Bineînțeles, a apărut întrebarea: *nu se poate oare elabora o tehnologie generală care să permită realizarea sistematică a unor programe elegante, ușor de depanat și de modificat?* Ca răspuns la această întrebare a apărut **metoda programării structurate**.

Un program structurat este format din unități funcționale bine conturate, ierarhizate conform naturii problemei. În interiorul unei astfel de unități, structurarea se manifestă atît la nivelul instrucțiunilor cît și la nivelul datelor.

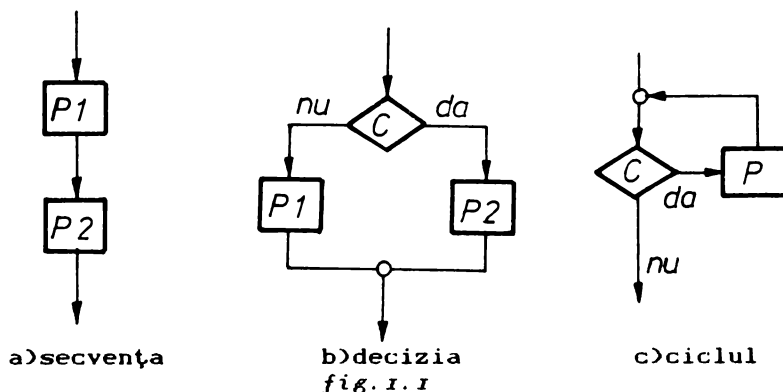
Programarea structurată este o *metodă independentă de limbajul de programare*, ea acționînd la nivelul stilului de lucru. Metoda programării structurate implică în primul rînd principiul de programare "de sus în jos" - adică descompunerea programului în elemente logice independente, numite *module*, fiecare modul avînd o funcție bine definită în cadrul programului.

Avantajele unei astfel de metode de lucru sînt:

- a)-scrierea programului poate fi făcută de mai multe persoane, fiecare ocupîndu-se de cîte un modul;
- b)-modificarea unui program se poate face foarte ușor, modificarea unui modul neafectînd celelalte module;
- c)-programul poate fi extins oricînd prin adăugarea de noi module;
- d)-structura modulară este independentă de limbajul de programare utilizat; modulele pot fi scrise în limbaje diferite și apoi asamblate.

Un alt principiu al programării structurate derivă din teorema de structură a lui Böhm și Jacopini care arată că orice organigramă se poate construi folosind doar trei tipuri de structuri de control: *secvențială* (secvența), *alternativă* (decizia), *repetitivă* (ciclul) prezentate în fig.1.1.

Limbajul PASCAL a fost inițial conceput de N. Wirth pentru a înlesni însușirea de către studenți a principiilor "artei" programării. Noutățile pe care le aduce acest limbaj se pot grupa în două mari categorii care sînt legate de conceperea acțiunilor și de structurarea datelor:



A)-conceperea acțiunilor: o serie întreagă de instrucțiuni reprezintă chiar structurile de control impuse de tehnica programării structurate (IF-THEN-ELSE, WHILE, REPEAT, FOR, CASE); acestea sînt alternativele instrucțiunii GOTO, permițînd exprimarea în mod clar și natural a tuturor acțiunilor posibile;

B)-reprezentarea datelor: s-au introdus structuri de date complexe - ca articolul, mulțimea, fișierul; există posibilități de a descrie alte structuri noi, folosind cele deja existente. De asemenea programatorul are posibilitatea să-și definească propriile tipuri de date. La toate acestea se adaugă facilitatea de a defini și manipula structuri dinamice (liste liniare, arbori).

Aceste avantaje ale limbajului PASCAL au făcut ca acesta să fie acceptat foarte repede de programatorii profesioniști - urmarea imediată a fost creșterea spectaculoasă a productivității de programare. Astfel, utilizarea limbajului PASCAL poate face scrierea programelor de 10 ori mai rapid, prețul softului scăzînd cu 30-70% (Business Week, 19 martie 1979).

Evident, odată cu apariția și răspîndirea fulgerătoare a calculatoarelor personale, au apărut implementări ale limbajului PASCAL pentru ele. Avînd în vedere că acest manual se adresează în primul rînd elevilor din clase de informatică și că în general liceele cu asemenea clase au în dotare calculatoare personale Tim-S, HC-85, COBRA - compatibile Spectrum, vom descrie implementarea realizată în 1983/84 de firma HISOFT, care este cea mai reușită implementare pentru acest tip de calculatoare. Vom prezenta conceptele de bază ale limbajului PASCAL, iar acolo unde se impune vom indica specificațiile pentru implementarea aleasă. Menționăm că toate programele prezentate în această carte pot fi rulate pe orice calculator personal compatibil-Spectrum fără nici o modificare.

1.2. STRUCTURA PROGRAMELOR PASCAL

Pentru a putea intui modul în care se scrie un program PASCAL, dăm pentru început un exemplu simplu: adunarea a două numere întregi.

```
PROGRAM ADUN;
VAR X,Y,Z: INTEGER;
BEGIN
  READ(X,Y);
  Z:=X+Y;
  WRITE(Z)
END.
```

Analizînd exemplul de mai sus, vom observa:

a) Orice program PASCAL trebuie să înceapă cu o declarație de program. Aceasta constă din cuvîntul cheie PROGRAM urmat de un identificator (în cazul nostru ADUN) care reprezintă numele programului. Caracterul ';' este un simbol standard și are rol de separator.

b) Toate variabilele cu care se lucrează în program trebuie declarate, indicîndu-se tipul lor (în cazul nostru X, Y și Z sînt întregi).

c) Cuvintele cheie BEGIN și END delimitează corpul de instrucțiuni al blocului; instrucțiunile sînt simple: citește X și Y, atribuie lui Z suma X+Y, scrie valoarea lui Z.

d) Orice program PASCAL se termină cu '.' pus după cuvîntul cheie END.

1.3. DESCRIEREA SINTAXEI LIMBAJULUI PASCAL CU AJUTORUL DIAGRAMELOR

În cele ce urmează vom descrie limbajul PASCAL și construcțiile specifice acestuia cu ajutorul *diagramelor de sintaxă*. O diagramă de sintaxă este un graf orientat avînd o singură intrare și o singură ieșire - orice drum posibil de la intrare la ieșire, care urmează sensul de parcurgere indicat de săgeți, definește o construcție admisibilă sintactic. Nodurile grafului sînt elipse, cercuri și dreptunghiuri. Elipsele și cercurile încadrează cuvinte cheie, respectiv simbolurile standard ale limbajului. Acestea se regăsesc identic în textul sursă al oricărui program. Noțiunile încadrate în dreptunghi se definesc prin alte diagrame de sintaxă (textul din dreptunghi poate fi însoțit de comentarii explicative puse între paranteze).

Utilizînd cele arătate mai sus, diagrama de sintaxă a unui program PASCAL arată astfel:

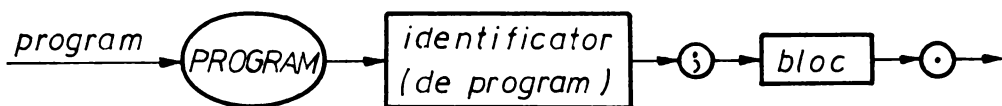


fig. 1.2.

Din diagrama din fig.1.2. se observă că unitatea de bază a programelor PASCAL este *blocul*. Acesta cuprinde descrierile datelor și acțiunile efectuate asupra lor.

Structura generală a unui bloc este dată în fig 1.3.

Singura parte obligatorie a blocului, așa cum se observă din diagramă, este corpul de instrucțiuni. Avînd în vedere că în PASCAL pot exista instrucțiuni vide, următorul exemplu este un program PASCAL corect:

```
PROGRAM SIMPLU;
BEGIN
END.
```

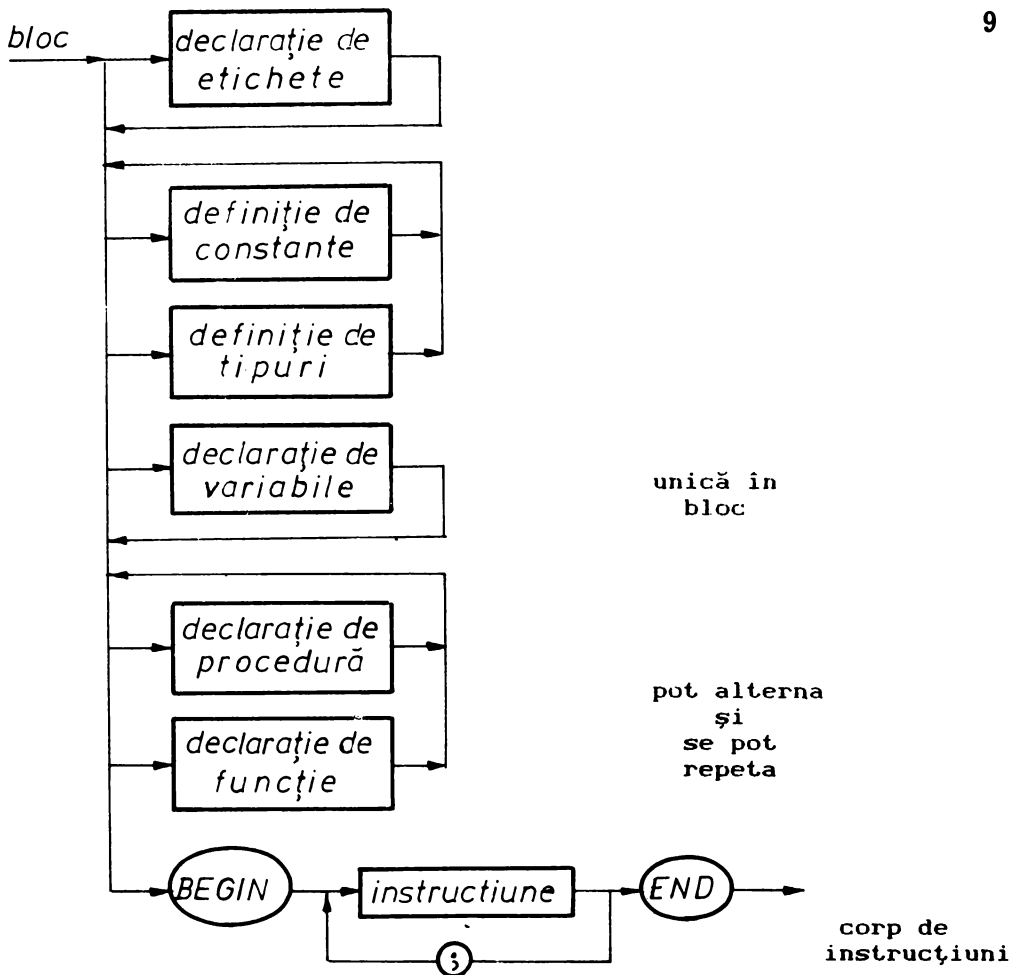


fig. 1.3.

Să urmărim un program PASCAL mai complex și anume calculul tuturor permutărilor ce se pot genera cu 4 elemente.

```

AE75 10 PROGRAM PERMUTARE:
AE75 20
AE75 30 { OBTINEREA PERMUTARILOR DE N ELEMENTE FOLOSIND
AE75 40   APELUL RECURSIV
AE75 50   se fixeaza un element si se obtin permutarile,
AE75 60   pentru celelalte N-1 elemente ...
AE75 70
AE75 80 CONST N=4;CR=CHR(13);PR=CHR(16);
AE75 90 VAR I:INTEGER;
AE7E 100   A:ARRAY[1..N] OF 1..9;
AE7E 110
AE7E 120 PROCEDURE TIPARESTE;
AE81 130 VAR I:INTEGER;
AE81 140 BEGIN
AE99 150   FOR I:=1 TO N DO WRITE(A[I]);
AEE9 160   WRITE(CR)
AEEE 170 END;
AEF5 180
AEF5 190 PROCEDURE PERMUTA(K:INTEGER);
AEF8 200 VAR I,X:INTEGER;
AEF8 210 BEGIN
AF10 220   IF K=1 THEN TIPARESTE ELSE
AF31 230   BEGIN
AF31 240     FOR I:=1 TO K DO
AF5B 250     BEGIN
AF5E 260       X:=A[I];A[I]:=A[K];A[K]:=X;
AFF2 270       PERMUTA(K-1);

```

```

B003 280      X:=A[I];A[I]:=A[K];A[K]:=X
B08C 290      END
B097 300      END
B09B 310      END:
B0A4 320
B0A4 330      BEGIN
B0AD 340      FOR I:=1 TO N DO A[I]:=I;
B0F0 350      WRITE (PR);
B0F5 360      PERMUTA(N);
B0FE 370      WRITE (CR,PR)
B108 380      END {$P}.

```

```

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

```

Componentele programului sint:

linia 10 -declarația de program;
liniile 30-60 -comentarii;
liniile 70, 110, 180, 320 -linii vide pentru evidențierea componentelor programului;
linia 80 -definiția de constante;
liniile 90-100 -declarația de variabile din blocul exterior;
liniile 120-170, 190-310 -două declarații de procedură;
liniile 330-380 -corpul de instrucțiuni al blocului exterior;
Prima procedură (liniile 120-170) se compune din:
linia 120 -numele procedurii;
linia 130 -declarația de variabile a procedurii;
liniile 140-170 -corpul de instrucțiuni al procedurii;
A doua procedură (liniile 190-310) se compune din:
linia 190 -numele procedurii și parametrul de intrare K;
linia 200 -declarația de variabile a procedurii;
liniile 210-310 -corpul de instrucțiuni al procedurii;

Execuția programului începe cu prima instrucțiune din corpul programului (linia 340). În acest sens se poate da o regulă generală: execuția unui program PASCAL începe cu prima instrucțiune din blocul cel mai exterior (programul principal). Linia 360 conține apelul procedurii PERMUTA. În acest moment este lansat în execuție corpul de instrucțiuni al procedurii PERMUTA (liniile 210-300). După terminarea execuției se dezactivează procedura și se revine la instrucțiunea care urmează apelului (linia 370). În corpul de instrucțiuni al procedurii PERMUTA se apelează și procedura TIPARESTE, ba mai mult, se apelează chiar procedura PERMUTA! Acest mod de lucru va fi tratat la timpul potrivit (în capitolul V), urmărind aici să dăm doar o imagine generală a unui program PASCAL.

II. ELEMENTELE DE BAZA ALE LIMBAJULUI PASCAL

2.1. VOCABULARUL LIMBAJULUI PASCAL

2.1.1. Setul de caractere

Elementele constitutive ale limbajului PASCAL se construiesc pe baza următoarelor caractere:

- literele mari și mici ale alfabetului latin, numite *caractere alfabetice*;
- cifrele sistemului de numerație zecimal, numite *caractere numerice*;
- simbolurile +, -, *, /, =, >, <, (,), [,], {, }, ,, ., ;, :, ', , \$, numite *caractere speciale*.

2.1.2. Identificatori

Identificatorii sînt o *succesiune de litere și cifre*, primul caracter fiind obligatoriu o literă. Identificatorii pot avea orice lungime, dar pentru unele calculatoare sînt semnificative doar primele opt caractere. Toate aceste reguli sînt redade de diagrama din fig.2.1.

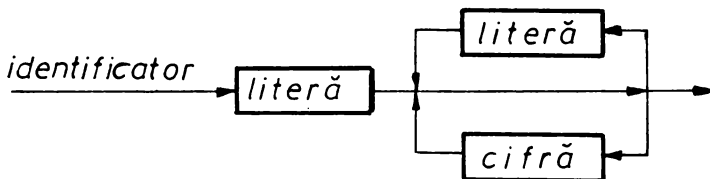


fig.2.1.

Exemple:

X, PROG1, EXEMPLU, AX13B, SUMA, A1

Următoarele nume nu sînt identificatori:

x-y, 7z, gr.el

Orice identificator înainte de a fi referit într-o anumită linie program trebuie să fie definit sau declarat într-o linie anterioară referirii sale.

Identificatorii desemnează nume de: programe, constante, tipuri, variabile, funcții, proceduri, parametri.

În afara acestora există și identificatori cu semnificație predefinită, cunoscuți sub numele de *cuvinte rezervate* (*cuvinte cheie și identificatori predeclarați*) ale limbajului.

Cuvintele cheie sînt predefinite pentru o anumită utilizare, care va fi indicată în diagrame de sintaxă. Cuvintele cheie sînt cuprinse în tabelul T.2.1.

Tabel T.2.1.

AND	ARRAY	BEGIN	CASE	CONST	DIV
DO	DOWNTO	ELSE	END	FOR	FORWARD
FUNCTION	GOTO	IF	IN	LABEL	MOD
NIL	NOT	OF	OR	PACKED	PROCEDURE
PROGRAM	RECORD	REPEAT	SET	THEN	TO
TYPE	UNTIL	VAR	WHILE	WITH	

Observație:

Acestor cuvinte cheie li se mai adaugă cuvântul FILE, dar cum structura fișier nu este implementată în HP4TM, el nu este trecut în tabelul de sus.

Identificatorii predeclarați sint cuprinși în tabelul T.2.2.

Tabel T.2.2.

ABS	ADDR	ARCTAN	BOOLEAN	CHAR	CHR
COS	ENTIER	EOLN	EXP	FALSE	FRAC
HALT	INCH	INLINE	INTEGER	INP	LN
MARK	MAXINT	NEW	ODD	ORD	OUT
PAGE	PEEK	POKE	PRED	RANDOM	READ
READLN	REAL	RELEASE	ROUND	SIN	SIZE
SQR	SQRT	SUCC	TAN	TIN	TOUT
TRUE	TRUNC	USER	WRITE	WRITELN	

Observație:

Printre identificatorii predeclarați în acest tabel sint unii care sint "cunoscuți" numai în HP4TM; de asemenea în alte implementări utilizatorul va întilni o serie de alți identificatori predeclarați.

Acești identificatori sint definiți într-un bloc ipotetic existent care include programul utilizatorului. Spre deosebire de cuvintele cheie, identificatorii predeclarați pot fi redefiniți în cadrul programului și utilizați în sensul nou dorit de programator.

2.1.3. Separatori și comentarii

Separatorii care delimitează două unități sintactice PASCAL sint spațiul (blancul), sfirșitul de linie (<<CR>>) și comentariul. Caracterul ';' se utilizează pentru separarea instrucțiunilor și a declarațiilor.

Comentariile, încadrate între delimitatorii '{' și '}', sint texte din care poate face parte orice caracter cu excepția simbolurilor '{' și '}', și pot fi inserate oriunde în program. Sintaxa comentariilor este prezentată în fig.2.2.

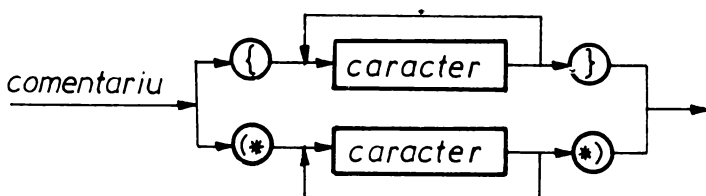


fig.2.2.

Exemple:

```
{SA STUDIEM SI PASCAL}
{REZOLVAREA ECUATIEI DE GRADUL DOI}
```

2.2. CONSTANTELE LIMBAJULUI PASCAL

Limbajul PASCAL admite următoarele constante: constante întregi, constante reale, șiruri de caractere și constante desemnate prin identificatori. Sintaxa constantelor este prezentată în diagrama din fig.2.3.

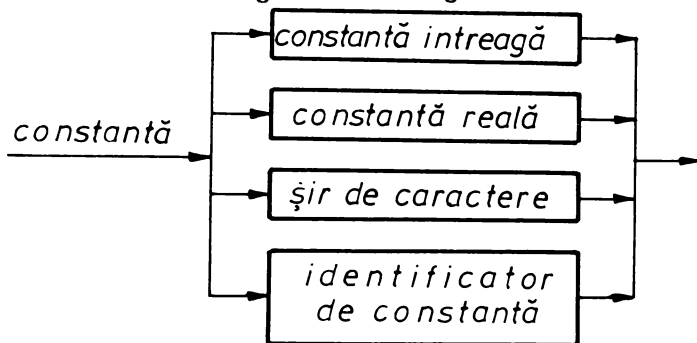


fig.2.3.

2.2.1. Constante întregi

Constantele întregi sînt numere întregi care nu pot depăși valoarea 32767, valoare cunoscută de compilator sub numele de MAXINT. Diagrama de sintaxă a constantelor întregi este cea din fig.2.4.

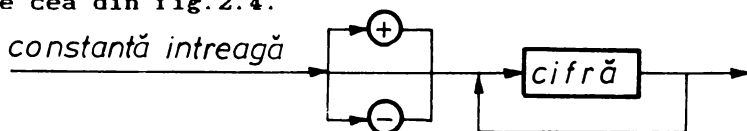


fig.2.4.

Exemple:

13, -25, 625, 1991

Următoarele numere nu sînt constante întregi:

76.8, 12*

MAXINT poate fi utilizat într-un program ca și o constantă definită de programator.

Exemplu:

```
IF I<MAXINT THEN I:=I+1; .
```

2.2.2. Constante reale

Constantele reale sînt numere reale cu valori absolute cuprinse între $5.9E-39$ și $3.4E38$, atît partea întregă cît și partea zecimală trebuind să conțină cel puțin o cifră, chiar dacă această cifră este zero. Diagrama de sintaxă este dată în fig.2.5.

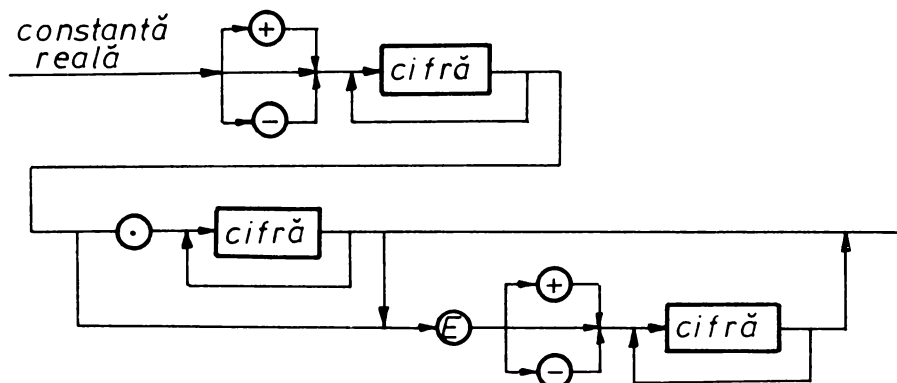


fig.2.5.

Exemple:

8.23145E1, 0.123E2, 1.2E-4, -42.6, 0.0, 64E-33

Următoarele constante nu sînt constante reale corecte:

75, 89 E-2, .77

2.2.3.Șir de caractere

O succesiune de caractere cuprinsă între apostrofuri, formează un șir de caractere. Sintaxa șirului de caractere este dată de diagrama din fig.2.6.

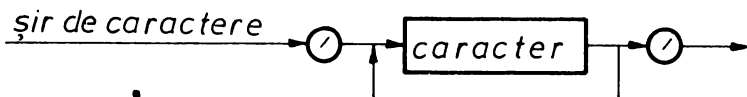


fig.2.6.

Lungimea șirului de caractere este chiar numărul caracterelor componente. Dacă din succesiunea de caractere face parte apostroful, atunci acesta trebuie dublat.

Exemple:

'EXEMPLU DE SIR DE CARACTERE', 'B-42 ', '30 MAI ''91'

Șirul de caractere de lungime 1, reprezintă o constantă caracter.

2.2.4.Constante desemnate prin identificatori

Identificatorii care desemnează constante în limbajul PASCAL aparțin următoarelor categorii:

- identificatorii introduși prin definiții de constante
- identificatorii constantelor unui tip enumerat
- identificatorii standard TRUE și FALSE
- identificatorii NIL și MAXINT

Toți acești identificatori sînt tratați în capitolele următoare.

2.3. DEFINIREA CONSTANTELOR UTILIZATOR

Intr-un program PASCAL, constantele pot fi utilizate direct prin valoare sau printr-un identificator. Acest identificator se definește în program în partea de definiție a constantelor. Astfel, constanta este utilizată în locul valorii sale, prin identificatorul său.

Diagrama de sintaxă este cea din figura 2.7.

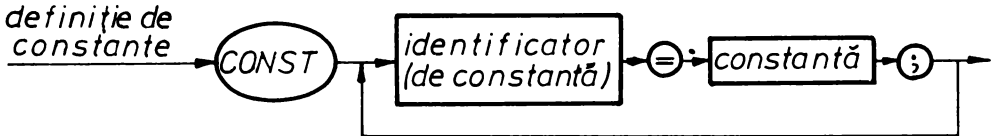


fig.2.7.

Exemple:

```
CONST PI=3.1415926;
      E=2.7182;
      N=42;
      TITLU='PASCAL';
      BL=' ';
```

Odată stabilită valoarea identificatorului de constantă, aceasta nu se mai poate modifica prin instrucțiuni ale programului. Utilizarea constantelor definite prezintă o serie de avantaje:

- economie de memorie (valoarea constantei apare o dată);
- modificarea simplă a programelor (se modifică doar linia de definiție a constantei);
- claritate sporită a programelor prin folosirea de identificatori.

Exemplu:

```
PROGRAM EXEMPLU;
CONST PI=3.1415926;
.....
A:=PI*R*R;
V:=(4*PI*R*R*R)/3;
```

2.4. NOȚIUNEA DE TIP

În limbajul PASCAL, orice dată - fie că reprezintă valoarea unei variabile, a unei constante sau a unei expresii - este de un anumit tip. Tipul de date stabilește mulțimea valorilor care pot fi atribuite variabilelor de tipul respectiv, cât și mulțimea operațiilor posibile asupra acestor valori.

Se evidențiază următoarele trei categorii de tipuri de date: tipuri scalare, tipuri structurate și tipul pointer. Din prima categorie fac parte cele patru tipuri standard: REAL, INTEGER, BOOLEAN și CHAR. Dintre acestea, INTEGER, BOOLEAN și CHAR sînt tipuri ordinale.

Mulțimile de valori care reprezintă tipurile de date, depind de implementare.

O submulțime a numerelor întregi, care conține constantele întregi precedate eventual de semn, formează tipul **INTEGER**.

Submulțimea finită a numerelor reale, formată din constantele reale precedate eventual de semn formează tipul **REAL**.

Mulțimea valorilor logice **TRUE** și **FALSE**, formează tipul **BOOLEAN**.

Tipul **CHAR** este format din mulțimea caracterelor ASCII, elementele mulțimii fiind toate șirurile de caractere de lungime 1.

Tipurile **INTEGER**, **REAL**, **BOOLEAN**, **CHAR**, au fost doar trecute în revistă (pentru a putea scrie primele programe PASCAL), prezentarea în detaliu a fiecăruia făcându-se în capitolul IV și capitolul VI.

2.5. VARIABILELE LIMBAJULUI PASCAL ȘI DECLARAREA LOR

Variabila înmagazinează valoarea unei date și spre deosebire de o constantă, poate fi modificată pe parcursul execuției programului. Fiecărei variabile i se asociază un identificator, care ia valori corespunzătoare unui anumit tip.

În limbajul PASCAL, toate variabilele utilizate într-un program, trebuie declarate. Declarația variabilelor se face conform diagramei de sintaxă din fig 2.8.

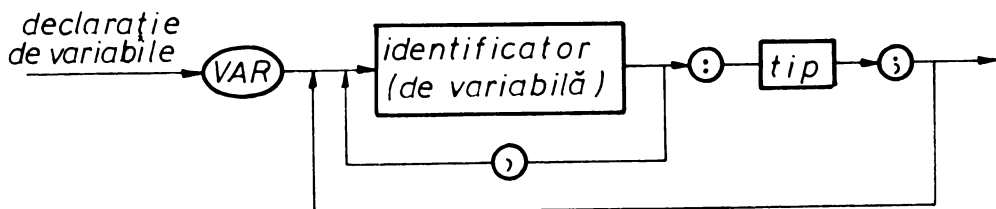


fig. 2.8.

Exemple:

a). VAR X: REAL;
A, B: INTEGER;

b). VAR NR: REAL;
A, A1: CHAR;
F: BOOLEAN;

În exemplul a) sînt declarate variabilele de tip întreg A și B și o variabilă X de tip real. În exemplul b) sînt declarate o variabilă reală NR, două variabile de tip caracter A, A1 și o variabilă de tip logic F.

Noțiunea de variabilă va fi aprofundată în cadrul capitolului IV.

2.6. EXPRESII. FUNCȚII STANDARD

O succesiune de operații asupra unor valori în vederea obținerii unei noi valori formează o expresie. În limbajul PASCAL, sintaxa expresiilor este dată de diagrama care urmează în fig. 2.9.

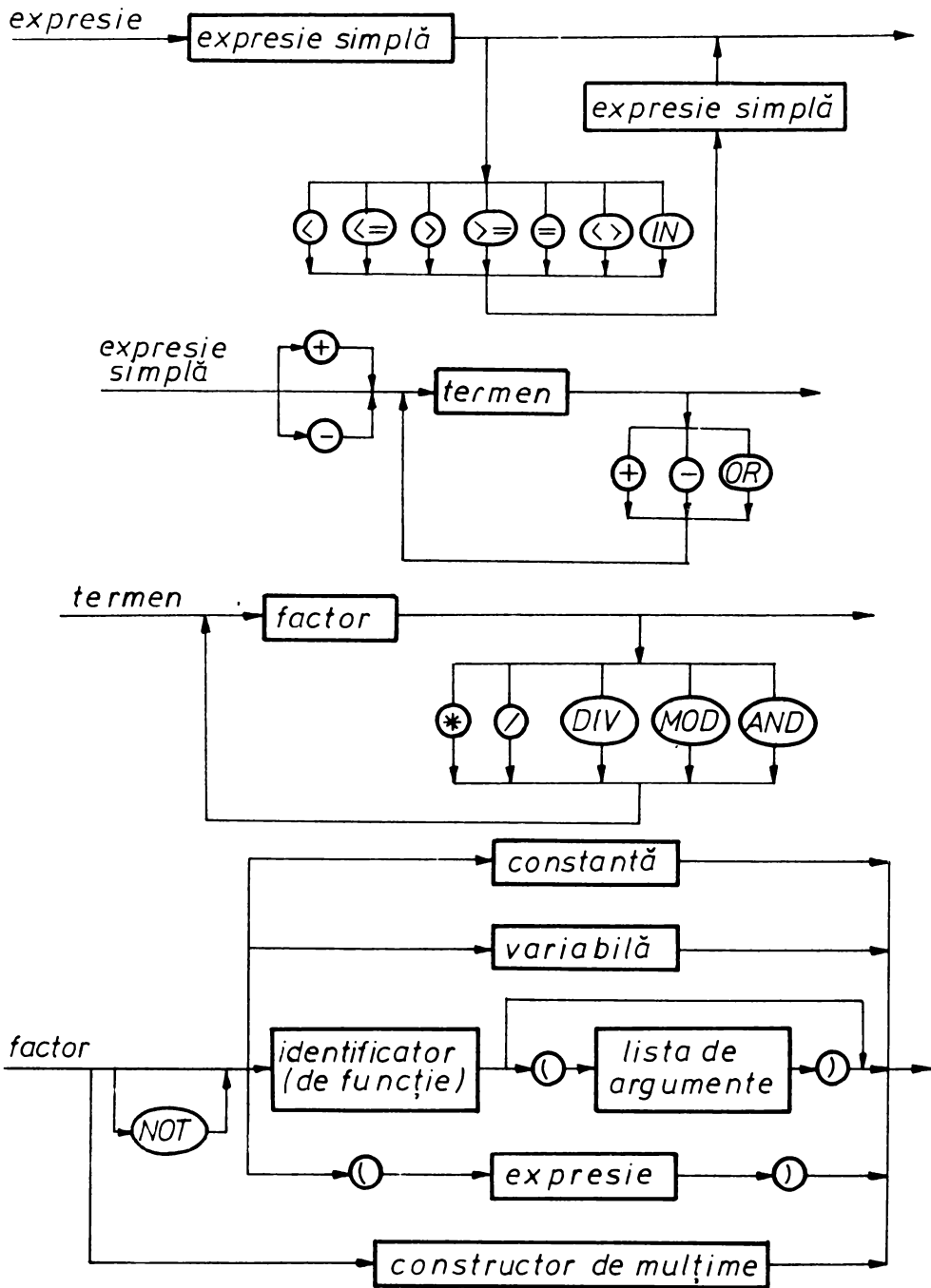


fig. 2.9.

Tipul expresiei este dat de tipul valorii rezultate în urma evaluării expresiei. Acesta depinde atât de tipul operanzilor care intervin în expresia respectivă cât și de operatorii care se aplică asupra lor.

Exemple:

```
VAR A, B, C: REAL;
    I, J, K: INTEGER;
```

In condițiile de mai sus, expresiile $(A+B)/2$, $7.5*I$, J/K sint expresii reale, $(I+J) \text{ DIV } K$, $J \text{ MOD } 3$ sint expresii întregi. iar $A > (B+C)/7$, $(A+B < C) \text{ AND } (J/K > A)$ sint expresii booleene.

Rezultă deci că în funcție de operatorii utilizați, în PASCAL avem expresii aritmetice și expresii logice (booleene).

Pentru înțelegerea exemplelor amintim că în PASCAL avem:

- operatori aritmetici: +, -, *, /, DIV, MOD (DIV este operatorul de împărțire întreagă, MOD este operatorul pentru determinarea restului împărțirii întregi);
- operatori relaționali: <, >, <=, >=, =, <>, IN (IN este operatorul de verificare a apartenenței la o mulțime a unui element);
- operatori logici: NOT, AND, OR.

Operatorii vor fi tratați detaliat, ținând cont de tipul de date asupra cărora se aplică, în capitolul IV.

Între operatorii unei expresii există următoarele reguli de prioritate:

- prioritate 0 - operatorul NOT
- prioritate 1 - operatorii multiplicativi: *, /, DIV, MOD, AND
- prioritate 2 - operatorii aditivi: +, -, OR
- prioritate 3 - operatorii relaționali

Operatorii cu aceeași prioritate se execută de la stînga la dreapta în ordinea în care apar. Prezența parantezelor poate modifica aceste priorități, regula de desfacere a parantezelor fiind cea din aritmetică. Toate aceste reguli se deduc și din diagramele de sintaxă prezentate anterior.

Exemple:

- a) Expresii: $A=43$, $X>Y$, $A=B$;
- b) Expresii simple: $X-Y$, $A*A+B$;
- c) Termeni: $(X<43) \text{ AND } (X>Y)$, $A/(A+B)$;
- d) Factori: $(A+B-C)$, X , 45 , $\text{COS}(X)$, $\text{NOT } A$;

În diagrama de sintaxă a unui factor apare noțiunea de *identificator de funcție*. În PASCAL există o serie de *funcții standard* care pot fi apelate în orice program, prin intermediul identificatorului predeclarat care este numele funcției, urmat de parametrii actuali. Prezentarea detaliată a fiecărei funcții standard se va face în capitolul V. Dintre acestea, evidențiem funcțiile standard din tabelul T.2.3., în vederea scrierii unor programe simple:

Tabel T.2.3.

FUNCȚIA	OPERAȚIA REALIZATA
ABS(X)	valoarea absolută a lui X
ARCTAN(X)	arctangentă de X
TAN(X)	tangentă de X
COS(X)	cosinus de X
SIN(X)	sinus de X
EXP(X)	"e" la puterea X
LN(X)	logaritm natural de X
SQR(X)	X la puterea 2
SQRT(X)	radical din X

Pentru funcțiile trigonometrice unghiurile se dau în radiani. Avînd în vedere că în PASCAL nu există operatori pentru ridicarea la putere și nu a fost implementată o funcție standard, se recomandă ca această operație să se realizeze pe baza formulei:

$$a^x = e^{x \cdot \ln a}, \quad a \in \mathbb{R}, \quad a > 0, \quad x \in \mathbb{Z}.$$

Exemplu:

$X = a^9$ devine în PASCAL $X:=EXP(9*LN(A));$

2.7. OPERAȚII DE CITIRE/SCRIERE SIMPLE

Reamintim că în această lucrare se prezintă concepte de bază PASCAL, dar detaliile se referă la implementarea HP4TM, elaborată de firma HISOFT pentru calculatoare personale compatibile Spectrum. Procedurile de intrare/ieșire prezintă multe particularități de la o implementare la alta. Programatorului i se recomandă, în cazul în care întâlnește alte implementări, să lucreze cu precauție cu procedurile de intrare/ieșire.

Citirea/scrierea datelor în PASCAL se realizează prin intermediul procedurilor READ și WRITE, READLN și WRITELN, a căror expunere completă se va face în capitolul V.

Printr-o procedură READ pot fi citite mai multe valori de tip întreg, real sau caracter, care se atribuie variabilelor desemnate de parametrii efectivi ai procedurii. Valorile citite trebuie să fie compatibile cu tipul variabilelor corespunzătoare, altfel se semnalează eroare la execuție.

Exemplu.

```
VAR A,B: INTEGER;
    X: REAL;
    C: CHAR;
```

citire	valori introduse în timpul execuției de la tastatură	efect
READ(A,B);	10 20	A:=10 B:=20
READ(A,X);	10 3.5	A:=10 X:=3.5
READ(X);	13	X:=13.0
READ(A,B);	4.2 25	eroare
READ(C);	*	C:='*'
READ(A,B,C);	10 20 *	A:=10 B:=20 C:=' '
READ(A,B,C);	10 20*	A:=10 B:=20 C:='*'

Pe marginea exemplurilor de mai sus se impun câteva observații:

- 1)- valorile numerice trebuie separate;
- 2)- blanurile dintre valorile numerice se ignoră;
- 3)- avînd în vedere că blankul este și el un caracter, atunci cînd se citesc caractere și acesta va fi citit și atribuit unei variabile de tip caracter;
- 4)- dacă dorim să citim o valoare de tip caracter după o valoare de tip numeric (întreg sau real) trebuie să o introducem imediat după valoarea numerică, sau de pe o linie

terminal nouă, citirea valorii numerice precedente făcându-se cu READLN.

5)- în cazul introducerii de la tastatură a unei valori întregi pentru citirea unei variabile reale are loc o conversie de la întreg la real, conversie permisă și în caz de atribuire.

READLN este o procedură de citire care poate fi apelată cu sau fără parametri. READLN cu listă de parametri este echivalent cu READ cu aceeași listă și un READLN fără parametri. Printr-un READLN fără parametri se ignoră restul necitit al liniei curente și se așteaptă citirea unei linii noi, ale cărei valori vor fi asociate cu variabilele din procedura READ sau READLN următoare.

Exemplu:

```
VAR A,B: INTEGER;
    C: CHAR;
```

citire	valori introduse de la tastatură	efect
READLN(A,B);	10 20<CR>	A:=10 B:=20
READLN(C);	*<CR>	C:='*'
READLN(A,B);	10<CR>	A:=10
	20<CR>	B:=20
READLN(C);	*<CR>	C:='*'
READLN(A,B,C);	10 20<CR>	A:=10 B:=20, iar valoarea
	*<CR>	lui C va fi caracterul
		sfirșit de linie al cărui
		cod ASCII este 13

Din exemplele acestui paragraf rezultă:

- valorile numerice trebuie separate cu spațiu sau <CR>;
- spațiile și <CR>-urile dintre valorile numerice se ignoră;
- <CR> generează un caracter (cu codul ASCII 13) care se va atribui variabilei de tip caracter imediat următoare unei variabile de tip numeric, dacă în timpul execuției după valoarea numerică se tastează <CR>.

Afișarea uneia sau a mai multor date de tip întreg, real, caracter sau șir de caractere, se face prin apelul procedurii WRITE. Valorile afișate se înscriu pe același rînd în continuare. Valorile reale se afișează sub formă exponențială, formă care se poate modifica utilizînd scrierea cu punct zecimal, fără exponent.

Forma generală a scrierii cu punct zecimal, fără exponent a numerelor reale este:

WRITE(d:m:n)

unde:

- d - expresie de tip real
- m - număr total cifre
- n - număr de cifre al părții zecimale

Exemple:

```
WRITE(43.216:6:1);      va afișa 43.2
WRITE(43.216:4:0);      va afișa 43
WRITE(43.216:4:2);      va afișa 4.32160E+01
WRITE(-85.3);           va afișa -.85300E+02
```

Procedura **WRITELN** realizează funcțiile procedurii **WRITE**, în plus, după efectuarea afișării, cursorul se mută pe linie nouă. Astfel valorile expresiilor dintr-un eventual **WRITE** următor se vor afișa pe linie nouă. **WRITELN** fără parametri are ca efect mutarea cursorului pe o linie nouă.

Exemple:

```
a) WRITELN('A','B');
    WRITELN;
    WRITE('C');
    WRITELN('D');
```

Secvența de mai sus are ca efect afișarea următoarelor rînduri:

```
linie 1   AB
linie 2
linie 3   CD
b) WRITE('PRIMA LINIE');
    WRITELN;
    WRITE('LINIA A 2-A');
```

Această secvență are ca efect afișarea următoarelor două rînduri:

```
linie 1   PRIMA LINIE
linie 2   LINIA A 2-A
```

PROBLEME PROPUSE

1. Să se stabilească dacă următoarea secvență este corectă:

```
PROGRAM CORECT;
CONST NR:17;
      AD:X;
      CULOARE=ROZ;
      ORICE='ROZ';
```

și să se corecteze în caz contrar.

2. Fie variabilele X, Y, Z , avind valorile 2.0, 3.0 și respectiv 4.0. Ce valori au variabilele A și B după executarea următoarelor instrucțiuni (toate variabilele sînt de tip real):

```
a)  $X:=X+Y; Y:=Z; A:=X+Y+Z; B:=Y-X;$ 
b)  $A:=\text{SQR}(X); B:=\text{SQRT}(Z)+A/X;$ 
c)  $X:=\text{SQR}(X); Z:=\text{SQRT}(Z); A:=X-Z; B:=A-Y;$ 
```

3. Să se scrie în PASCAL următoarele expresii:

```
a)  $(2x-3y)^2 - 1$ 
b)  $\sqrt{15x - 3}$ 
c)  $\sqrt[3]{a - b} - \frac{5}{x - y}$ 
d)  $\frac{(a - x)^2}{a + x} - \sin(x^2 + a)$ 
```

4. Fie A, B variabile de tip întreg, C de tip real și D de tip caracter. Cum trebuie introduse valorile de la tastatură pentru următoarele citiri:

```
a) READ (A,B,D);      pentru  A=17, B=42, D='A'
b) READ (B,C);        pentru  B=27, C=7,2
c) READ (A,B);        pentru  A=8,9, B=10
```

5. Variabilele X și Y sînt de tip real și au valorile 0,5 respectiv 123,456. Ce se afișează cu instrucțiunile:

```
a) WRITELN(X);
b) WRITELN(X:4:1,Y:8:3);
c) WRITELN(X:6:4,Y:10:5);
d) WRITELN(X:5:2);
e) WRITE('SUMA = ',X+Y);
```

III. STRUCTURI FUNDAMENTALE DE CONTROL IN LIMBAJUL PASCAL

Programarea structurată urmărește elaborarea de programe pe baza unor structuri fundamentale de control în scopul obținerii unor produse program clare și fiabile. Orice program poate fi reformulat astfel încât să conțină numai structurile fundamentale de control caracteristice programării structurate: *secvențială, alternativă și repetitivă.*

Lista instrucțiunilor PASCAL este prezentată în diagrama din fig.3.1.

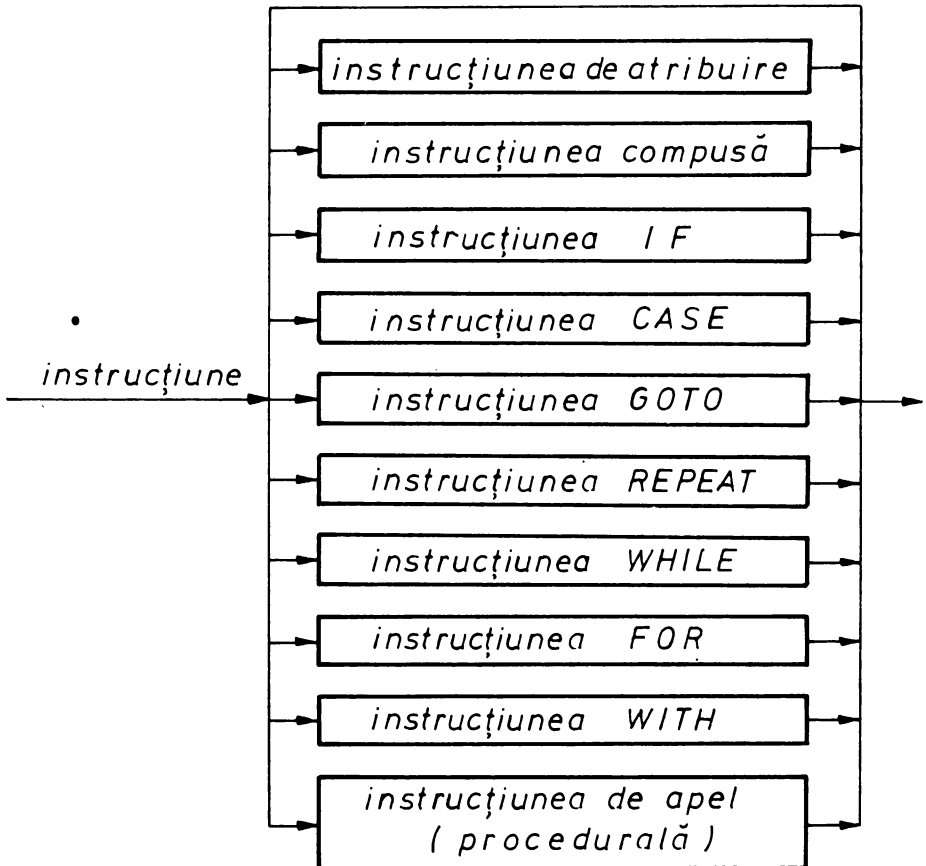


fig.3.1.

Observație:

Dintre aceste instrucțiuni, instrucțiunea WITH va fi tra-

tată în capitolul IV, iar instrucțiunea de apel în capitolul V.

3.1. STRUCTURA SECVENȚIALĂ

3.1.1. Instrucțiunea de atribuire

Forma generală a structurii secvențiale este prezentată în fig.3.2.

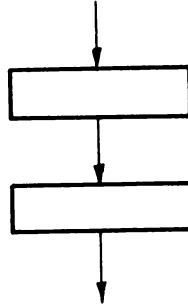


fig. 3.2.

În limbajul PASCAL, această structură se descrie prin intermediul instrucțiunii de atribuire a cărei sintaxă este dată de diagrama din fig.3.3.

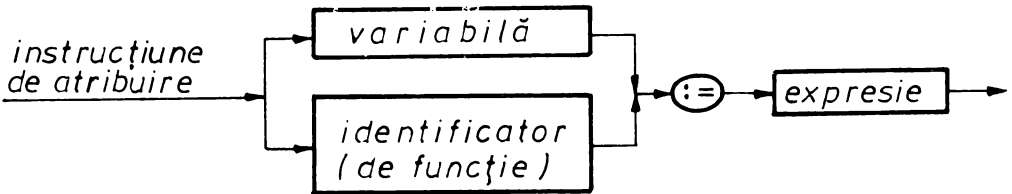


fig. 3.3.

Efect:

- 1) -se evaluează expresia;
- 2) -se compară tipul valorii expresiei cu tipul variabilei;
 - dacă tipurile coincid, are loc atribuirea;
 - dacă tipurile nu coincid, atribuirea eșuează.

Observație:

În PASCAL, în general nu se fac conversii de tip în timpul execuției atribuirii. Unele implementări, printre care și HP4TM, permit atribuirea unei valori întregi unei variabile de tip real, în prealabil efectuându-se conversia necesară.

Exemple:

```
VAR A,I,S: INTEGER;
    N,X: REAL;
.....
S:=7;
A:=13;
N:=S/3;
X:=(SQRT(N)+1)/I;
```

Ca urmare a efectuării instrucțiunilor de atribuire din exemplu, variabila de tip întreg A va avea valoarea 13, întregului S i se asociază valoarea 7. Lui N i se atribuie valoarea reală rezultată din împărțirea reală a întregului 7 cu 3 etc.

În PASCAL HP4TM nu este permisă inițializarea variabilelor la declararea lor.

3.1.2. Instrucțiunea compusă

Sintaxa instrucțiunii compuse este prezentată în fig.3.4.

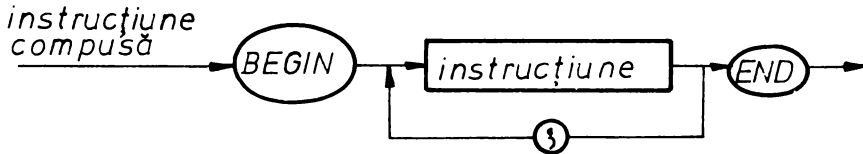


fig. 3. 4.

Efectul instrucțiunii compuse se obține din înlanțuirea efectelor instrucțiunilor componente care sînt executate strict în ordinea scrierii lor. Instrucțiunea compusă permite regruparea mai multor instrucțiuni într-una singură. Va fi folosită ori de cîte ori sînt necesare mai multe instrucțiuni într-o instrucțiune unde sintaxa autorizează utilizarea unei singure instrucțiuni.

Exemplu:

```
BEGIN
  X1:=k+1;
  X2:=(A1+A2)/2;
END;
```

Din diagrama de sintaxă a programului PASCAL rezultă că și corpul de instrucțiuni al unui bloc are forma unei instrucțiuni compuse. De asemenea, instrucțiunea compusă poate face parte din ansamblul instrucțiunilor unei alte instrucțiuni compuse, caz în care perechile BEGIN-END se grupează după regula din matematică a parantezelor. Prezența separatorului ';' este obligatorie între instrucțiuni și opțională înainte de END. Dacă se pune ';' înainte de END, atunci între ';' și END se consideră o instrucțiune vidă. Efectul instrucțiunii vide este nul.

PROGRAMUL III.1.

Să se calculeze volumul unui con circular drept cu generatoarea și raza bazei date.

```

algoritmul VOL este:
  citeste r,g
  pi ← 3,1415926
  h ← √(g2-r2)
  v ← pi*r*h/3
  scrie v
  stop
```

Programul PASCAL corespunzător este programul VOL (P.III.1). În linia 20 este definită constanta PI, iar în liniile 30-40 sînt declarate variabilele utilizate. Deoarece funcția standard SQRT are valoare reală, variabila H, care identifică înălțimea, se declară ca o variabilă de tip real. De asemenea variabila V (volumul) este declarată tot de tip real, deoarece operatorul aritmetic '/' produce rezultat real. Corpul de instrucțiuni este format din liniile 60-120 care realizează

citirea razei și a generatoarei conului, calculul înălțimii și a volumului, cit și afișarea valorii calculate pentru volum. La afișare s-a folosit reprezentarea cu punct zecimal, fără exponent a variabilei reale V pe 10 poziții, din care 2 zecimale.

În vederea utilizării imprimantei, în program apare instrucțiunea `WRITELN(CHR(16))`, care asigură comutarea ieșirii de la monitor la imprimantă și viceversa. În același scop, în linia 130 apare și opțiunea de compilare `P` care comută canalul de ieșire de la monitor la imprimantă.

```

ACEC 10 PROGRAM VOL;
ACEC 20 CONST PI=3.1415926;
ACEC 30 VAR R,G:INTEGER;
ACF5 40     V,H:REAL;
ACF5 50 BEGIN
ACFE 60     READ(R,G);
AD0A 70     H:=SQRT(G*G-R*R);
AD3F 80     V:=PI*SQR(R)*H/3;
AD71 90     WRITELN(CHR(13));
AD7B 100    WRITELN('RAZA: ',R,'     GENERATOAREA: '.G);
AD8C 110    WRITELN('VOLUMUL CONULUI: ',V:10:2);
ADEE 120    WRITELN(CHR(13));
ADFS 130 END {$P}.

```

```

RAZA:5     GENERATOAREA:8
VOLUMUL CONULUI: 163.49

```

P. III.1.

PROGRAMUL III.2.

Dându-se lungimile laturilor unui triunghi, să se calculeze aria triunghiului și lungimile înălțimilor sale.

algoritmul ARIA1 este:

```

citeste a,b,c

$$p \leftarrow \frac{a+b+c}{2}$$


$$s \leftarrow \sqrt{p*(p-a)*(p-b)*(p-c)}$$

ha ← 2*s/a
hb ← 2*s/b
hc ← 2*s/c
scrie s,ha,hb,hc
stop

```

Implementarea în PASCAL este dată în programul `ARIA1` (P. III.2).

```

AD75 10 PROGRAM ARIA1;
AD75 20 VAR A,B,C:INTEGER;
AD7E 30     P,S,HA,HB,HC:REAL;
AD7E 40 BEGIN
AD87 50     READ(A,B,C);
AD99 60     WRITELN(CHR(13),'A: ',A);
ADB9 70     WRITELN('B= ',B);
ADD2 80     WRITELN('C= ',C);
ADEB 90     P:=(A+B+C)/2;
AE19 100    S:=SQRT((P*(P-A)*(P-B)*(P-C)));
AE7B 110    HA:=2*S/A;
AE9F 120    HB:=2*S/B;
AEC3 130    HC:=2*S/C;
AE7 140     WRITELN('ARIA= ',S:10:2);
AF0E 150    WRITELN('INALTIMEA HA= ',HA:10:2);
AF3D 160    WRITELN('INALTIMEA HB= ',HB:10:2);
AF6C 170    WRITELN('INALTIMEA HC= ',HC:10:2,CHR(13));
AF9F 180 END {$P}.

```

```

A=30
B=40
C=50
ARIA= 600.00
INALTIMEA HA= 40.00
INALTIMEA HB= 30.00
INALTIMEA HC= 24.00

```

P. III. 2.

3.2. STRUCTURA ALTERNATIVA

Forma generală a structurii alternative, numită și IF-THEN-ELSE, este prezentată în fig.3.5.

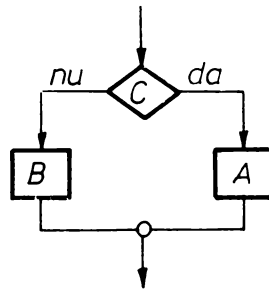


fig.3.5.

Dacă A sau B lipsește, structura se numește structură alternativă simplă sau pseudoalternativă (IF-THEN).

3.2.1. Instrucțiunea IF

În limbajul PASCAL, structura alternativă se descrie cu ajutorul instrucțiunii IF, a cărei sintaxă este prezentată în fig.3.6.

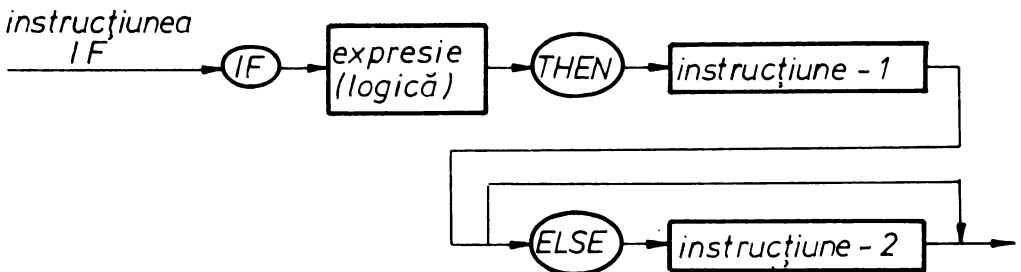


fig.3.6.

Efect:

- 1)-se evaluează expresia logică;
- 2)-dacă valoarea logică a expresiei este TRUE, se execută instrucțiune-1;
- 3)-dacă valoarea logică a expresiei este FALSE, ori se execută instrucțiune-2 (dacă este prezentă alternativa ELSE), ori nu se execută nimic;
- 4)-se părăsește instrucțiunea IF.

Instrucțiune-1 și instrucțiune-2 pot fi orice instrucțiune PASCAL, inclusiv instrucțiunea vidă.

Exemple:

- a) IF X=0 THEN S:=S+I
ELSE P:=P*I;
- b) IF A>B THEN
BEGIN
X:=A+B;
Y:=A;
Z:=B;
WRITE(X,Y,Z)
END
ELSE WRITE(A,B);
- c) IF (X>A) AND (X<B) THEN WRITE(X);
- d) IF X=0 THEN
ELSE X:=X+1;
- e) IF Y>10 THEN Y:=Y-1
ELSE;

Observații:

1)-înaintea cuvântului cheie ELSE nu se pune ';

2)-dacă oricare ramură a instrucțiunii IF conține mai multe instrucțiuni atunci acestea se grupează într-o singură instrucțiune compusă;

3) atât instrucțiune-1 cât și instrucțiune-2 pot conține alte instrucțiuni IF, caz în care asocierea unui ELSE se face cu THEN-ul cel mai apropiat care îl precede și care nu a fost asociat încă. Dacă se impune o altă asociere, pentru evitarea oricărei ambiguități se recomandă soluțiile ilustrate în exemplul următor.

Exemplu:

Fie următoarea structură:

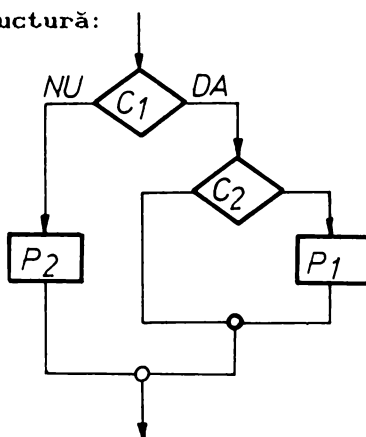


fig.3.7.

- | | | | |
|----|--|----|---|
| 1) | if c1
then if c2
then p1
else
else p2; | 2) | if c1
then
begin
if c2
then p1
end
else p2; |
|----|--|----|---|

Soluția 1) utilizează `else vid`, astfel evitând atașarea ramurii `else p2` cazului `c2=false`.

Soluția 2) delimitează instrucțiunea IF (care trebuie executată când `c1=true`) într-o instrucțiune compusă, chiar dacă acest lucru nu se impune de sintaxă deoarece avem o singură instrucțiune.

4) O instrucțiune IF formulată neglijent poate fi foarte costisitoare. Să considerăm n condiții reciproc independente c_1, c_2, \dots, c_n care provoacă, fiecare, execuția cite unei secvențe.

Fie condițiile:

if c_1 then s_1 ;

if c_2 then s_2 ;

.....

if c_{n-1} then s_{n-1} ;

if c_n then s_n ;

Pentru a grăbi ieșirea din IF, vom așeza condițiile în ordinea descrescătoare a probabilității îndeplinirii lor.

În cazul în care din condițiile de mai sus numai una poate fi îndeplinită la un moment dat, se recomandă următorul model:

```

if  $c_1$ 
  then  $s_1$ 
  else if  $c_2$ 
    then  $s_2$ 
    else.....
      .....
        else if  $c_{n-1}$ 
          then  $s_{n-1}$ 
          else if  $c_n$ 
            then  $s_n$ ;

```

5)-Pentru a evita utilizarea abuzivă a instrucțiunii IF se recomandă ca în loc de

```

IF CHEIE=VALOARE
  THEN GASIT:=TRUE
  ELSE GASIT:=FALSE;

```

să se scrie:

```

GASIT:=CHEIE=VALOARE;

```

PROGRAMUL III.3.

Se dau numerele reale X, Y și numărul întreg N . Dacă N este diferit de zero, să se calculeze suma numerelor X, Y , în caz contrar să se calculeze produsul lor.

```

algoritmul PIII3 este:
  citeste  $x, y, n$ 
  daca  $n=0$  atunci  $p \leftarrow x*y$ 
    scrie  $p$ 
  altfel  $s \leftarrow x+y$ 
    scrie  $s$ 
  sfdaca
stop

```

Programul PASCAL corespunzător este P1 (P. III.3),
el exemplifică structura alternativă completă.

```

AD05 10 PROGRAM P1;
AD05 20 VAR X,Y,P,S:REAL;
AD0E 30     N:INTEGER;
AD0E 40 BEGIN
AD17 45     READ(N,X,Y);
AD31 50     WRITELN;
AD34 60     WRITELN('N=',N);
AD4D 70     WRITELN('X=',X:5:2);
AD71 80     WRITELN('Y=',Y:5:2);
AD95 90     IF N#0 THEN
ADA7 100        BEGIN
ADA7 110            P:=X*Y;
ADC1 120            WRITELN('PRODUSUL=',P:10:2)
ADE9 130        END
ADEC 140        ELSE
ADEF 150            BEGIN
ADEF 160                S:=X+Y;
AE09 170                WRITELN('SUMA=',S:10:2)
AE2D 180            END;
AE30 190     WRITELN
AE30 200 END {$P}.

```

```

N=0
X=45.78
Y=13.13
PRODUSUL=    601.09

```

```

N=72
X=53.12
Y=75.73
SUMA=    120.85

```

P. III.3.

PROGRAMUL III.4.

Să se calculeze maximul a trei numere reale date.

```

algoritmul MAX este:
┌
├   citeste a,b,c
├   m ← a
├   daca m<b atunci m ← b sfdaca
├   daca m<c atunci m ← c sfdaca
├   scrie m
└   stop

```

Transcrierea în limbaj PASCAL este dată în programul
MAX (P. III.4) care ilustrează structura pseudoalternativă ; în
exemplul dat, ramura corespunzătoare neindeplinirii condiției
este vidă.

```

ACD7 10 PROGRAM MAX;
ACD7 20 VAR A,B,C,M:REAL;
ACE0 30 BEGIN
ACE9 40     READ(A,B,C);
AD07 50     M:=A;
AD15 60     IF M<B THEN M:=B;
AD42 70     IF M<C THEN M:=C;
AD6F 80     WRITELN(CHR(13));
AD79 90     WRITELN('A=',A:6:2,' ',B=' ',B:6:2,' ',C=' ',C:6:2);
ADF9 100    WRITELN('MAXIMUL ESTE:',M:6:2);
AE28 110    WRITELN(CHR(13));
AE2F 120 END {$P}.

```

```

A= 23.34 B= 40.00 C= 0.76
MAXIMUL ESTE: 40.00

```

```

A= 34.00 B=789.65 C= 54.00
MAXIMUL ESTE:789.65

```

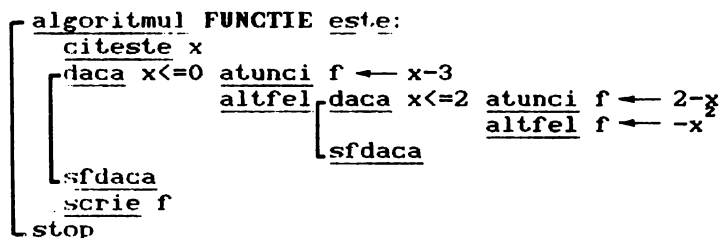
P. III.4.

PROGRAMUL III.5.

Să se calculeze valoarea funcției reale de variabilă reală, definită prin formula:

$$f(x) = \begin{cases} x-3 & \text{dacă } x \leq 0 \\ 2-\frac{x}{2} & \text{dacă } 0 < x \leq 2 \\ -x & \text{dacă } x > 2 \end{cases}$$

Valoarea variabilei x se citește.



Implementarea acestui algoritm în PASCAL este dată de programul FUNCTIE (P.III.5). În acest program se poate observa modul de asociere a unui ELSE cu THEN-ul cel mai apropiat care-l precede în cazul unui IF inclus în IF.

```

AE1B 10 PROGRAM FUNCTIE;
AE1B 20 VAR X,F:REAL;
AE24 30 BEGIN
AE2D 40 WRITE('X=');READ(X);
AE44 50 WRITELN('X=',X:6:2);
AE68 60 IF X<=0 THEN F:=X-3
AEA1 70 ELSE
AE30 80 IF X<=2 THEN F:=2-X
AEE0 90 ELSE F:=-X*X;
AF1A 100 WRITELN('VALOAREA FUNCTIEI ESTE:');
AF3F 110 WRITELN('F(',X:6:2,')=',F:10:4);
AF84 120 END {$P}.
  
```

```

X= 10.00
VALOAREA FUNCTIEI ESTE:
F( 10.00)=-100.0000
  
```

```

X= -3.00
VALOAREA FUNCTIEI ESTE:
F( -3.00)= -6.0000
  
```

```

X= 45.30
VALOAREA FUNCTIEI ESTE:
F( 45.30)=-2052.0887
  
```

P. III.5.

3.2.2. Instrucțiunea CASE

Instrucțiunea CASE este o generalizare a instrucțiunii IF în cazul selecției pentru mai mult de două alternative posibile. Sintaxa instrucțiunii este dată de diagrama din fig.3.8.

Expresia din sintaxa instrucțiunii CASE se numește *selector*, iar constantele se numesc *constante CASE*. Selectorul și constantele CASE trebuie să fie de același tip *ordinal*

Efect:

- 1)-se evaluează selectorul;
- 2)-se caută alternativa CASE în a cărei listă de constante CASE se regăsește valoarea selectorului;

3)-în cazul în care se găsește o asemenea alternativă, se execută instrucțiunea respectivă;

4)-în caz contrar se execută instrucțiunea specificată în ELSE (dacă există);

5)-se iese din instrucțiunea CASE.

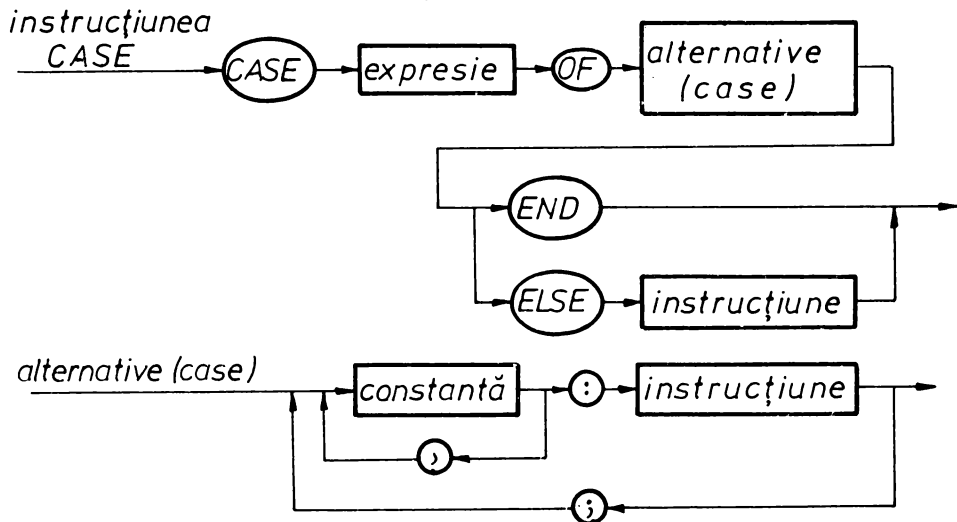


fig. 3.8.

Exemple:

a) VAR A: INTEGER;

```
.....
CASE A OF
    0: X:=100;
    1,7: X:=200;
    5,3,4: X:=300
END;
```

b) VAR ZI: CHAR;

```
.....
CASE ZI OF
    'L': WRITE('LUNI');
    'M': WRITE('MARTI SAU MIERCURI');
    'J': WRITE('JOI');
    'V': WRITE('VINERI')
    ELSE WRITE('ZI DE ODIHNA')
END;
```

Observații:

1)-constantele CASE trebuie să fie distincte între ele, dar nu trebuie să fie puse în ordine;

2)-pot fi cel mult 255 constante CASE;

3)-alternativele CASE nu pot fi etichetate, deci nu pot fi referite cu o instrucțiune GOTO;

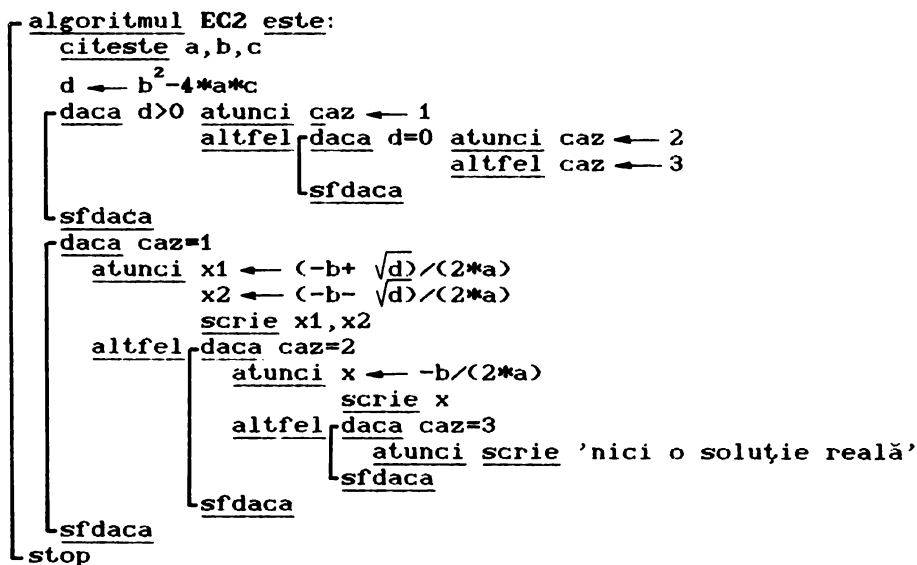
4)-implementare instrucțiunii CASE în HP4TM nu permite caracterul ';' în fața alternativei ELSE și nici în fața cuvântului cheie END;

5)-nu toate implementările permit forma cu ELSE a instrucțiunii CASE.

PROGRAMUL III.6.

Să se determine soluțiile reale ale ecuației de gradul doi cu coeficienți numere reale, nenule.

Programul EC2 (P.III.6) conține în liniile 20-30 un comentariu. În liniile 120-150 calculează discriminantul D al ecuației și după cum acesta este pozitiv, zero sau negativ, variabila CAZ ia respectiv valorile 1, 2 sau 3. Variabila CAZ este selectorul instrucțiunii CASE, iar 1, 2 și 3 constituie constantele CASE corespunzătoare. Se poate observa că selectorul și constantele CASE sînt de același tip ordinal, tipul INTEGER. Programul a fost executat de mai multe ori pentru a fi ilustrate toate cazurile prevăzute în rezolvarea ecuației de gradul doi.



```

AFD7 10 PROGRAM EC2:
AFD7 20 {APLICATIE LA CASE-OF;
AFD7 30 REZOLVAREA ECUAȚIEI DE GRADUL DOI}
AFD7 40 VAR A,B,C,D,X,X1,X2:REAL;
AFE0 50 CAZ:INTEGER;
AFE0 60 BEGIN
AFE9 70 WRITELN;WRITELN('INTRODU A,B,C');WRITELN;
B00A 80 READ(A,B,C);
B028 90 WRITELN;
B02B 100 WRITELN('A=',A:6:2,' ',',','B=',B:6:2,' ',',','C=',C:6:2);
B0AD 110 WRITELN;
B0B0 120 D:=B*B-4*A*C;
B0F1 130 IF D>0 THEN CAZ:=1
B11B 140 ELSE IF D=0 THEN CAZ:=2
B145 150 ELSE CAZ:=3;
B150 160 CASE CAZ OF
B153 170 1:BEGIN
B15D 180 X1:=(-B+SQRT(D))/(2*A);
B199 190 X2:=(-B-SQRT(D))/(2*A);
B1D9 200 WRITELN('DOUA SOLUTII:');WRITELN;
B1F7 210 WRITELN('X1=',X1:6:2);WRITELN;
B21F 220 WRITELN('X2=',X2:6:2);
B244 230 END;
B247 240 2:BEGIN
B251 250 X:=-B/(2*A);
B27E 260 WRITELN('O SOLUTIE');WRITELN;
B298 270 WRITELN('X=',X:6:2);
B2BC 280 END;
B2BF 290 3:WRITELN('NICI O SOLUTIE')
B2E2 300 END;
B2E5 310 WRITELN
B2E5 320 END {$P}).
  
```

A= 1.20 B= 1.00 C= 1.00

NICI O SOLUTIE

A= 1.00 B= -2.00 C= 1.00

O SOLUTIE

X= 1.00

A= 1.00 B= -5.00 C= 6.00

DOUA SOLUTII:

X1= 3.00

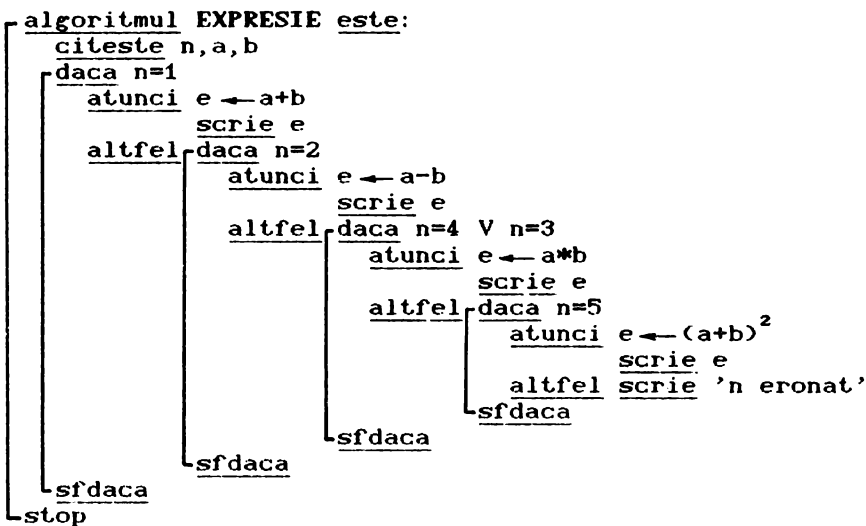
X2= 2.00

P. III.6

PROGRAMUL III.7.

Fiind date numerele reale A, B și N natural, să se calculeze valoarea E astfel:

$$E = \begin{cases} A+B & \text{dacă } N=1 \\ A-B & \text{dacă } N=2 \\ A*B & \text{dacă } N=3 \text{ sau } N=4 \\ (A+B)^2 & \text{dacă } N=5 \end{cases}$$



Avind în vedere că în acest algoritm structurile alternative ramifică execuția în funcție de valoarea unei variabile de tip ordinal, în programul EXPRESIE (P.III.7) vom lucra cu instrucțiunea CASE, astfel mărind lizibilitatea programului. Alternativa ELSE din CASE corespunde tuturor cazurilor când N nu este din intervalul [1,5].

```

AD67 10 PROGRAM EXPRESIE:
AD67 20 VAR A,B,E:REAL:
AD70 30 N:INTEGER:
AD70 40 BEGIN
AD79 50 READ(N,A,B);
AD93 60 WRITELN('N=',N);
ADAC 70 WRITELN('A=',A:8:2);
ADD0 80 WRITELN('B=',B:8:2);
ADF4 90 CASE N OF
ADF7 100 1:BEGIN
AE01 110 E:=A+B;
AE18 120 WRITELN('E=',E:10:2)
AE3C 130 END;
    
```

```

AE42 140      2:BEGIN
AE4C 150      E:=A-B;
AE6A 160      WRITELN('E=',E:10:2)
AE88 170      END;
AE71 180      4.3:BEGIN
AEA5 190      E:=A*B;
AE9F 200      WRITELN('E=',E:10:2)
AEE0 210      END;
AEE6 220      S:BEGIN
AEF0 230      E:=SQR(A+B);
AF0D 240      WRITELN('E=',E:10:2)
AF2E 250      END
AF31 260      ELSE WRITELN('N ERONAT');
AF4A 270 END (*P).

```

```

N=1
A= 4.60
B= 7.00
E= 11.60

```

```

N=2
A= 4.00
B= 6.20
E= -2.20

```

```

N=4
A= 6.00
B= 2.00
E= 12.00

```

```

N=5
A= 7.76
B= 8.45
E= 262.76

```

```

N=11
A= 1.00
B= -7.00
N ERONAT

```

P. III.7.

3.3. STRUCTURA REPETITIVA

Structura repetitivă este descrisă cu ajutorul instrucțiunilor repetitive, care permit repetarea în anumite condiții a unei instrucțiuni sau a unei secvențe de instrucțiuni.

Instrucțiunea repetată formează *corpul ciclului*, ciclu care constituie instrucțiunea repetitivă.

Cele trei tipuri ale structurii repetitive se descriu în PASCAL cu ajutorul următoarelor instrucțiuni:

1)-instrucțiunea WHILE (numită *ciclu cu test inițial*, sau ciclu anterior condiționat) descrie structura de tipul WHILE-DO;

2)-instrucțiunea REPEAT (numită *ciclu cu test final* sau ciclu posterior condiționat) descrie structura de tipul DO-UNTIL;

3)-instrucțiunea FOR (numită *ciclu cu contor* sau ciclu cu număr de repetiții cunoscut) descrie structura de tipul DO-FOR.

3.3.1. Instrucțiunea WHILE

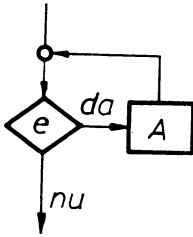


fig. 3.9.

Structura repetitivă de tipul WHILE-DO are forma generală din fig.3.9.

În limbajul PASCAL, această structură se descrie cu instrucțiunea WHILE, a cărei sintaxă este dată de diagrama din fig.3.10.

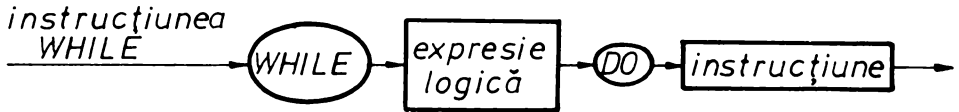


fig. 3.10.

Efect:

- 1)-se evaluează expresia logică;
- 2)-cât timp expresia logică are valoarea adevărat (TRUE) se execută corpul ciclului (instrucțiunea specificată după cuvântul cheie DO). Dacă corpul ciclului se compune din mai multe instrucțiuni atunci acestea se grupează într-o instrucțiune compusă.

Exemple:

a) X:=0;
 WHILE X<=N DO
 X:=X+1;

b) I:=1;P:=1;
 WHILE I<=N DO
 BEGIN
 P:=P*I;
 I:=I+1
 END;

Observații:

- 1)- dacă condiția de ciclare nu este îndeplinită la intrarea în ciclu atunci instrucțiune nu se va executa niciodată;
- 2)- deoarece expresia logică se evaluează la fiecare iterație, ea trebuie să fie cât se poate de simplă.

PROGRAMUL III.8.

Pentru primele N numere naturale să se listeze o tabelă de forma N, N^2, N^3, \sqrt{N} , unde N este dat.

```

algoritmul LIST este:
  citeste n
  i ← 1
  cît timp i ≤ n executa
    scrie i, i2, i3, √i
    i ← i+1
  sf_cît timp
  stop

```

Programul LIST (P.III.8) realizează tabelarea cerută pe patru coloane.

```

ADF1 10 PROGRAM LIST;
ADF1 20 VAR I,N:INTEGER;
ADFA 30 BEGIN
AE03 40 READ(N);
AE09 50 WRITELN(CHR(13));
AE13 55 I:=1;
AE19 60 WHILE I<=N DO
AE31 70 BEGIN
AE31 80 WRITELN(I:2,I*I:5,I*I*I:7,SQRT(I):8:5);
AE36 90 I:=I+1;
AE8E 100 END;
AE90 110 WRITELN(CHR(13));
AE97 120 END {#P}.

```

```

1 1 1 1.00000
2 4 9 1.41421
3 9 27 1.73205
4 16 64 2.00000
5 25 125 2.23607
6 36 216 2.44949
7 49 343 2.64575
8 64 512 2.82843
9 81 729 3.00000
10 100 1000 3.16228
11 121 1331 3.31662
12 144 1728 3.46410
13 169 2197 3.60555
14 196 2744 3.74166
15 225 3375 3.87298
16 256 4096 4.00000
17 289 4913 4.12311
18 324 5832 4.24264
19 361 6859 4.35890
20 400 8000 4.47214

```

P. III. 8.

PROGRAMUL III. 9.

Să se efectueze împărțirea a două numere naturale prin scăderi succesive.

```

algoritmul CIT este:
  citește a,b
  c ← 0
  cittimp a>=b executa
    d ← a-b
    c ← c+1
    a ← d
  sfccittimp
  scrie c,a
  stop

```

Algoritmul este implementat în programul CIT (P.III.9). Corpul ciclului din P.III.9. este o instrucțiune compusă, descrisă în liniile 70-110. Numerele care se împart sînt afișate pe prima linie, citul și restul împărțirii pe o a doua linie.

```

AE2C 10 PROGRAM CIT;
AE2C 20 VAR A,B,C,D:INTEGER;
AE35 30 BEGIN
AE3E 40 READ(A,B);
AE4A 50 WRITELN('A=',A,' ',B=' ',B);
AE88 60 C:=0;
AE8E 70 WHILE A>=B DO
AEA7 80 BEGIN
AEA7 90 D:=A-B;
AEB9 100 C:=C+1;
AEC0 110 A:=D;
AEC0 120 END;
AED9 130 WRITELN(CHR(13));
AED3 140 WRITELN('CITUL=',C,' ',RESTUL=',',A);
AF1A 150 WRITELN(CHR(13));
AF21 160 END {#P}.

```

A=3 B=12
CITUL=0 RESTUL=3

A=27 B=12
CITUL=2 RESTUL=3

A=63 B=9
CITUL=7 RESTUL=0

P. III. 9.

3.3.2. Instrucțiunea REPEAT

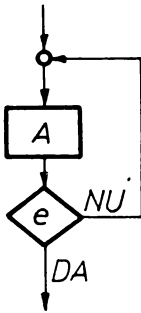


fig. 3. 11.

Forma generală a structurii repetitive de tipul DO-UNTIL este cea din fig.3.11.

Structura prezentată se descrie în PASCAL cu instrucțiunea REPEAT, a cărei sintaxă este dată de diagrama din fig.3.12.

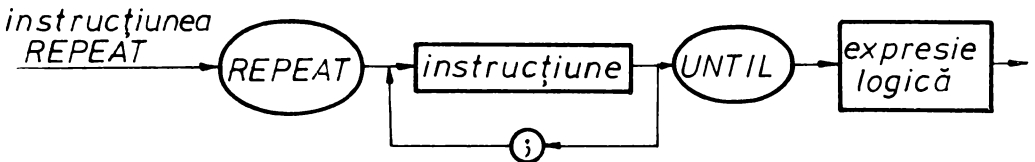


fig. 3. 12.

Efect:

-corpul ciclului (cuprins între REPEAT și UNTIL) se execută atît timp cît valoarea expresiei logice este falsă; se iese din ciclu atunci cînd valoarea expresiei logice devine adevărat.

Observații:

1)- spre deosebire de instrucțiunea WHILE, în cazul instrucțiunii REPEAT corpul ciclului este executat cel puțin o dată;

2)- în anumite implementări, înaintea cuvintului cheie UNTIL nu se pune ";"

3)- deoarece expresia logică se evaluează la fiecare iterație, ea trebuie să fie cît se poate de simplă.

Exemple:

a) REPEAT
 X:=X+1
 UNTIL X>0;

b) REPEAT
 READ(A,B,C);
 F:=A*B*C;
 WRITE(F)
 UNTIL F>100;

PROGRAMUL. III. 10.

Fie numărul real A, $A \geq 0$. Să se calculeze valoarea

rădăcinii pătrate a lui A cu o aproximație fixată prin constanta ε . Se va folosi formula de recurență:

$$x_{n+1} = \frac{1}{2} \left(x_n + \frac{A}{x_n} \right), \quad n=0,1,2,\dots \quad \text{unde } x_0 = A.$$

```

algoritmul RAD este:
  citeste a,  $\varepsilon$ 
   $x_1 \leftarrow a$ 
  repetă
     $x_2 \leftarrow x_1$ 
     $x_1 \leftarrow (x_2 + a/x_2)/2$ 
  ieșimcînd  $(x_2 - x_1) < \varepsilon$ 
  scrie  $x_1$ 
stop

```

Programul PASCAL corespunzător este RAD (P.III.10)

```

AE22 20 PROGRAM RAD;
AE22 30 CONST EPS=1E-8;
AE22 40 VAR A,X1,X2:REAL;
AE2B 50 BEGIN
AE34 60 READ(A);
AE3E 70 X1:=A;
AE4C 80 REPEAT
AE4C 90     X2:=X1;
AESD 100    X1:=(X2+A/X2)/2
AE86 110   UNTIL (X2-X1)<EPS;
AE8C 120   WRITELN(CHR(13));
AEC6 130   WRITELN('RADACINA PATRATA DIN ',A:10:2,'=',X1:10:5);
AF16 140   WRITELN(CHR(13))
AF1D 150 END {$P}.

```

```

RADACINA PATRATA DIN      2.00=   1.41421
RADACINA PATRATA DIN     11.10=   3.33167
RADACINA PATRATA DIN    123.21=  11.10000
RADACINA PATRATA DIN      9.00=   3.00000

```

P. III.10

PROGRAMUL III.11.

Să se ordoneze crescător trei numere naturale.

```

algoritmul ORD1 este:
  citeste a,b,c
  repetă
    sw  $\leftarrow$  0
    daca  $a > b$  atunci  $sw \leftarrow 1$ 
       $x \leftarrow a$ 
       $a \leftarrow b$ 
       $b \leftarrow x$ 
    sfdaca
    daca  $b > c$  atunci  $sw \leftarrow 1$ 
       $x \leftarrow b$ 
       $b \leftarrow c$ 
       $c \leftarrow x$ 
    sfdaca
    ieșimcînd  $sw=0$ 
    scrie a,b,c
  stop

```

Programul de rezolvare a problemei este ilustrat în ORD1 (P.III.11). Corpul ciclului este descris de liniile

90-180, care cuprind două structuri alternative simple. Valorile citite inițial sînt afișate pe prima linie, iar pe linia a doua sînt afișate aceleași valori ordonate crescător.

```

AD0F 20 PROGRAM ORD1:
AD0F 30 VAR A,B,C,SW,X: INTEGER;
AD18 40 BEGIN
AD21 50   READ(A,B,C);
AD33 60   WRITELN;
AD36 70   WRITELN('A=',A,'   B=',B,'   C=',C);
AD83 80   REPEAT
AD83 90     SW:=0;
AD8C 100    IF A>B THEN
AD9F 110      BEGIN
AD9F 120        SW:=1;X:=A;
ADA0 130        A:=B;B:=X
ADB1 140      END;
ADB7 150    IF B>C THEN
ADCA 160      BEGIN
ADCA 170        SW:=1;X:=B;
ADD6 180        B:=C;C:=X
ADDC 190      END
ADE2 200    UNTIL SW=0;
ADF4 210    WRITELN;WRITELN;
ADFA 220    WRITELN(A:9,B:9,C:9);
AE21 230 END {$P}.

```

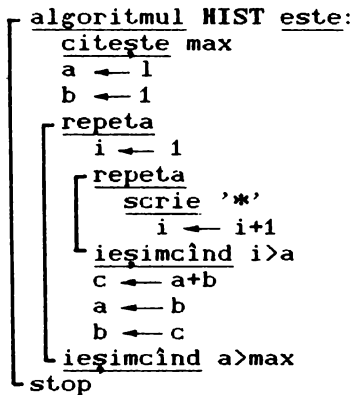
A=42 B=947 C=29

29 42 947

P. III.11.

PROGRAMUL III.12.

Să se afișeze histograma termenilor șirului Fibonacci, termeni care sînt mai mici sau egali decît o valoare maximă dată.



Exemplul descrie două structuri repetitive imbricate, ambele de tipul DO-UNTIL. Ciclul exterior descris de liniile 70-170 ale programului HIST (P. III.12). cuprinde la rîndul său un alt ciclu în liniile 90-120.

```

AE3C 10 PROGRAM HIST:
AE3C 20 VAR A,B,C,I,MAX: INTEGER;
AE45 30 BEGIN
AE4E 40   READ(MAX);
AE54 50   WRITELN(CHR(13));WRITELN;WRITELN('MAXIM:',MAX);
AE7E 60   A:=1;B:=1;
AE8A 70   REPEAT
AE8A 80     I:=1;

```

```

AE93 90      REPEAT
AE93 100     WRITE('* ');
AE9B 110     I:=I+1
AEA3 120     UNTIL I>A;
AEB5 130     WRITELN;
AEB8 140     C:=A+B;
AEC9 150     A:=B;
AECF 160     B:=C
AECF 170     UNTIL A>MAX;
AEE8 180     WRITELN(CHR(13))
AEEF 190     END {#P}.
    
```

```

MAXIM: 70
    
```

```

*
*
**
***
****
*****
*****
*****
*****
*****
*****
*****
*****
*****
    
```

P. III.12.

3.3.3. Instrucțiunea FOR

Dacă numărul de repetiții ale execuției structurii este cunoscut, atunci se recomandă structura repetitivă de tipul DO-FOR a cărei formă generală este cea din fig.3.13.

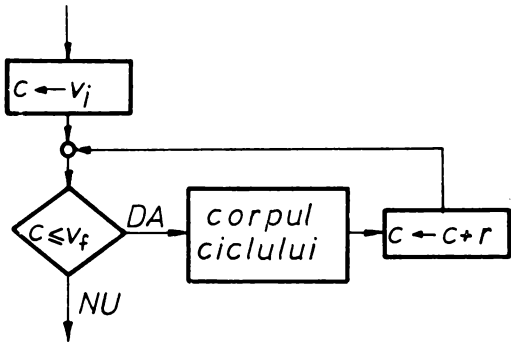


fig. 3. 13

Variabila de control *c* (contorul) se inițializează cu valoarea v_i și crește cu pasul *r* (rația) până la valoarea v_f inclusiv.

In limbajul PASCAL, această structură se descrie cu ajutorul instrucțiunii FOR, a cărei sintaxă este dată în fig.3.14.

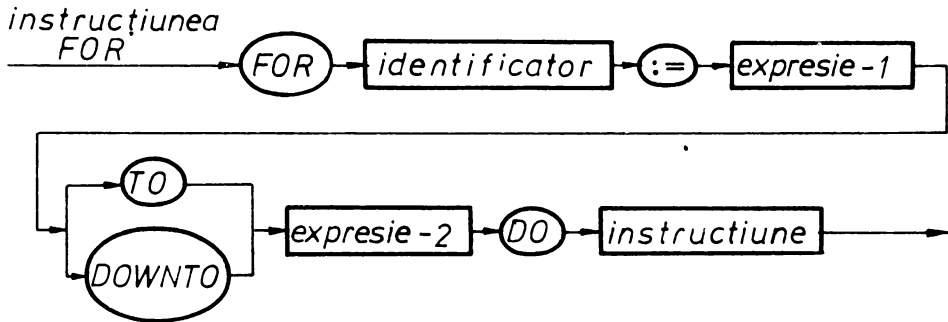


fig. 3. 14.

Corpul ciclului este instrucțiunea (evident, poate fi și instrucțiune compusă sau un alt ciclu FOR) care urmează cuvântului cheie DO, iar contorul ciclului este desemnat prin *identificator*.

Efect:

a) Forma cu TO

- 1)-se evaluează v_1 , v_2 (valorile expresiilor *expresie-1* respectiv *expresie-2*);
- 2)-se inițializează contorul c cu valoarea v_1 ;
- 3)-se compară valoarea contorului cu v_2 :
 - dacă $c > v_2$ se iese din instrucțiunea FOR;
 - dacă $c \leq v_2$ se execută *instrucțiune* și se continuă cu 4;
- 4)-se calculează următoarea valoare a contorului, adică se incrementează c cu o unitate și se reia pasul 3.

b) Forma cu DOWNTO

- 1)-se evaluează v_1 , v_2 (valorile expresiilor *expresie-1* respectiv *expresie-2*);
- 2)-se inițializează contorul c cu valoarea v_1 ;
- 3)-se compară valoarea contorului cu v_2 :
 - dacă $c < v_2$ se iese din instrucțiunea FOR;
 - dacă $c \geq v_2$ se execută *instrucțiune* și se continuă cu 4;
- 4)-se calculează următoarea valoare a contorului, adică se decrementează c cu o unitate și se reia pasul 3.

Observații:

- 1)-contorul și expresiile din instrucțiunea FOR trebuie să fie de același tip *ordinal*;
- 2)-valorile ambelor expresii se determină o singură dată la începutul ciclării; ca urmare, modificarea în ciclu a valorilor variabilelor componente în aceste expresii nu afectează numărul de iterații;
- 3)-pentru majoritatea implementărilor, la ieșirea din ciclu valoarea contorului este *nedefinită*.

Pot exista următoarele situații (forma TO) :

- a) $v_i > v_f$ - ciclul nu este executat niciodată;
- b) $v_i = v_f$ - ciclul este executat numai o dată;
- c) $v_i < v_f$ - ciclul este executat de $(v_f - v_i + 1)$ ori.

Evident, numărul execuțiilor se stabilește în mod similar și pentru forma cu DOWNTO.

Exemple:

- a) FOR I:=1 TO N DO Y:=Y*I;
- b) FOR I:=1 TO N DO
 BEGIN
 WRITE(' ');
 FOR J:=1 TO 2*I-1 DO WRITE('*')
 END;

Exemplul b) ilustrează două cicluri imbricate.

Programul. III. 13.

Să se calculeze și să se afișeze primii i termeni ($i=1,10$) ai sumei:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \dots$$

```

algoritmul SIR este:
  s ← 1
  t ← 1
  pentru i=1,10 executa
    t ← t/i
    s ← s+t
  scrie s
sfpentru
stop

```

Algoritmul este implementat în programul SIR (P. III.13) care afișează pe două coloane numărul de ordine al sumei și valoarea corespunzătoare.

```

AC7D 10 PROGRAM SIR;
AC7D 20 VAR I:INTEGER;
AC86 30 S,T:REAL;
AC86 40 BEGIN
AC8F 50 S:=1;
AC9C 60 T:=1;
ACA9 70 FOR I:=1 TO 10 DO
ACC3 80 BEGIN
ACC6 90 T:=T/I;
ACDF 100 S:=S+T;
ACF9 110 WRITELN(I,2,S:10:5)
AD19 120 END;
AD1F 130 END {$P}.

```

```

1 2.00000
2 2.50000
3 2.66667
4 2.70833
5 2.71667
6 2.71805
7 2.71825
8 2.71828
9 2.71828
10 2.71828

```

P. III. 13.

PROGRAMUL III. 14.

Se citesc pe rînd N numere reale. Să se calculeze și să se afișeze media aritmetică a cite două numere citite succesiv.

```

algoritmul MARIT este:
  citește n,a
  pentru i=1,n-1 executa
    citește b
    ma ← (a+b)/2
    scrie ma
    a ← b
  sfpentru
stop

```

Programul de rezolvare a problemei în PASCAL este MARIT (P. III.14).

```

AE66 20 PROGRAM MARIT;
AE66 30 VAR A,B,MA:REAL;
AE6F 40 I,N:INTEGER;
AE6F 50 BEGIN
AE73 62 READ(N,A);
AE88 70 WRITELN(CHR(16),CHR(13),'N=',N,CHR(13),CHR(16));
AE8D 80 FOR I:=1 TO N-1 DO
AEDC 90 BEGIN
AEDF 100 READ(B);
AEE9 110 MA:=(A+B)/2;
AF0E 120 WRITELN(CHR(16));
AF18 130 WRITELN('A=',A:6:2,' ', 'B=',B:6:2);
AF6A 140 WRITELN('MA=',MA:6:2);
AF8F 150 WRITELN(CHR(16));
AF99 160 A:=B
AF99 170 END
AFA7 180 END ($P).

```

N=7

A= 12.50 B= 34.00
MA= 23.25

A= 34.00 B= 34.00
MA= 34.00

A= 34.00 B=124.78
MA= 79.39

A=124.78 B= 67.30
MA= 96.04

A= 67.30 B= 7.95
MA= 37.62

A= 7.95 B= 20.00
MA= 13.97

P. III.14.

3.4. INSTRUCȚIUNEA GOTO

Sintaxa instrucțiunii GOTO este dată de diagrama din fig.3.15.



fig.3.15.

Efect:

-execuția continuă *necondiționat* cu instrucțiunea ce poartă eticheta din GOTO;

Etichetele se pot forma din maxim patru cifre zecimale și se declară într-o declarație LABEL, înaintea declarării constantelor și variabilelor.

Când o etichetă este folosită pentru a marca o instrucțiune, ea trebuie pusă la începutul instrucțiunii și va fi urmată de ':'. Diagrama de sintaxă a declarației de etichete este cea din fig.3.16.

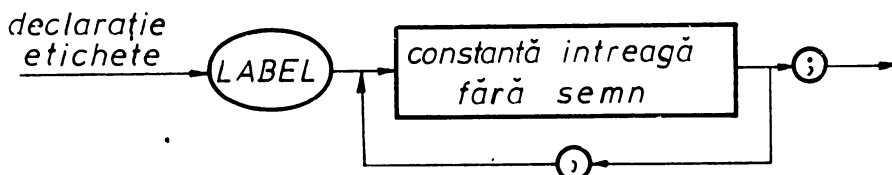


fig.3.16.

Exemple:

```

a) LABEL 13;          b) LABEL 25;
   .....
   BEGIN              BEGIN
   .....
   13:A:=I+1;         REPEAT
   .....
   GOTO 13;           GOTO 25;
   .....
   END                UNTIL I>N;
                       25:WRITE(I)
                       .....
                       END

```

Transferul controlului execuției este permis doar din cadrul unei instrucțiuni structurate în afara instrucțiunii (ex.b), *invers* nu. În acest sens, următorul exemplu este incorect:

```

LABEL 42;
.....
BEGIN
.....
GOTO 42;
.....
BEGIN
.....
42:IF ...
.....
END;
.....
END

```

Observație:

În general se recomandă evitarea utilizării instrucțiunii GOTO. Prezența ei se justifică doar în acele cazuri când din interiorul a mai multor structuri repetitive vrem să "evadăm".

PROGRAMUL P.III.15.

Să se citească un număr oarecare de numere întregi. Să se calculeze media aritmetică a elementelor pe grupuri. Grupurile sînt precedate de un număr care reprezintă numărul elementelor din grup. Dacă acest număr este 0 sau negativ, nu avem grup. Elementele grupurilor trebuie să fie numere pozitive. Dacă un element este negativ sau 0, se semnalează eroare și se oprește execuția.

```

algoritmul SUME este:
  citeste n
  cittimp n>0 executa
    suma ← 0
    pentru i=1,n executa
      citeste elem
      daca elem<=0
        atunci scrie 'dată eronată'
          stop
      sfdaca
        suma ← suma + elem
    sfpentru
    medie ← [suma/n]
    scrie medie
    citeste n
  sfcittimp
stop

```

Cum un program PASCAL nu poate avea două instrucțiuni "END.", (corespunzătoare stop-ului din pseudocod) în caz de dată eronată trebuie să indicăm salt la sfârșitul programului. Caracterul ',' din linia 200 a programului SUME (P.III.15) generează o instrucțiune vidă absolut necesară. Dacă ar lipsi, eticheta 1000 nu ar indica o instrucțiune.

```

AE1A 10 PROGRAM SUME;
AE1A 20 LABEL 1000:(ELEMENT ERONAT IN SUBSIR)
AE20 30 VAR N,I,ELEM,SUMA,MEDIE:INTEGER;
AE29 40 BEGIN
AE32 50 READ(N);(NUMAR ELEMENTE IN SUBSIR)
AE38 60 WHILE N>0 DO
AE4E 70 BEGIN
AE4E 80 SUMA:=0;
AE54 90 FOR I:=1 TO N DO
AE72 100 BEGIN
AE75 110 READ(ELEM);
AE78 120 IF ELEM<=0 THEN
AE90 130 BEGIN
AE90 140 WRITELN('DATA ERONATA');
AEAA 150 GOTO 1000;
AEAD 160 END;
AEAD 170 SUMA:=SUMA+ELEM
AE81 180 END;
AE02 190 MEDIE:=SUMA DIV N;(REZULTAT ROTUNJIT)
AE01 200 WRITELN('MEDIA SUBSIRULUI CU ',N,' ELEMENTE POZITIVE ESTE ',
AF18 210 ,MEDIE);
AF28 220 READ(N);
AF2E 230 END;
AF31 240 1000:(VALOARE ERONATA-NU DORIM SA O PRELUCRAM;
AF31 250 NU PUTEM AFISA NIMIC DECARECE EVENTUALUL
AF31 260 MESAJ AR APAREA SI IN CAZUL DATELOR CORECTE);
AF31 270 END{#P}.

```

P. III.15.

Cum orice program poate fi scris fără GOTO, în continuare prezentăm varianta respectivă pentru programul de mai sus. Variabila booleană *eroare* din programul SUMEFARAGOTO (P. III.16) permite evitarea prelucrării unei date eronate.

```

ADAE 10 PROGRAM SUMEFARAGOTO;
ADAE 20 VAR N,I,ELEM,SUMA,MEDIE:INTEGER;
ADAE 30 ERORARE:BOOLEAN;
ADAE 40 BEGIN
ADE7 50 ERORARE:=FALSE;
ADBB 60 READ(N);
ADC1 70 WHILE (N>0) AND NOT ERORARE DO
ADDF 80 BEGIN
ADDF 90 SUMA:=0;
ADE3 100 I:=1;
ADEB 110 REPEAT
ADEB 120 READ(ELEM);
ADF4 130 IF ELEM>0
ADFC 140 THEN
AE07 150 BEGIN
AE07 160 SUMA:=SUMA+ELEM;
AE18 170 I:=I+1;
AE28 180 END;
AE1F 190 ELSE
AE22 200 ERORARE:=TRUE;
AE27 210 UNTIL (I>N) OR ERORARE;
AE40 220 IF ERORARE
AE40 230 THEN
AE47 240 WRITE('ERORARE LA INTRARE');
AE57 250 ELSE
AE66 260 BEGIN
AE66 270 WRITE('MEDIA ROTUNJITA A SUBSIRULUI DIN ');
AE92 280 WRITELN(N,' ELEMENTE ESTE ',SUMA DIV N);
AECA 290 END;
AECA 300 END;
AECA 310 END{#P}.

```

P. III.16.

Comparind cele două variante de rezolvare ale aceleiași probleme se observă că programul cu GOTO este mai lizibil și mai scurt, deci în acest caz se justifică utilizarea acestei instrucțiuni.

PROBLEME PROPUSE

1. Fiind dată valoarea în radiani a unui unghi, să se calculeze valoarea echivalentă în grade, minute și secunde.

2. Să se calculeze valoarea funcției $f: \mathbb{R} \rightarrow \mathbb{R}$:

$$f(x, y) = \begin{cases} x+y & \text{dacă } 0 \leq x-y \leq 9 \\ x+y & \text{dacă } x-y < 0, y \geq 0 \\ x-y & \text{în restul cazurilor} \end{cases}$$

pentru $x, y \in \mathbb{R}$.

3. Se dau trei puncte prin coordonatele lor. Să se calculeze perimetrul și aria triunghiului determinat de punctele date.

4. Să se calculeze produsul a două numere naturale prin adunări repetate.

5. Să se afișeze termenii șirului Fibonacci mai mici decît o valoare dată.

6. Să se calculeze cmmdc și cmmmc a două numere naturale date, folosind algoritmul lui Euclid.

7. Fiind dat n natural, să se calculeze sumele :

a) $s = 1 + 1*2 + 1*2*3 + \dots + 1*2*3* \dots *n$

b) $s = 1 + \frac{1}{1*2} + \frac{2}{1*2*3} + \dots + \frac{n-1}{1*2*3* \dots *n}$

8. Se citesc pe rînd n numere reale. Să se calculeze produsul celor nenule și suma celor mai mici decît 10.

9. Se citesc pe rînd n numere reale. Să se numere cîte dintre ele sînt pare și pozitive.

IV. TIPURI DE DATE

Tipul unei date definește o mulțime de valori pe care le poate lua orice variabilă de tipul respectiv și operațiile care se pot efectua utilizând aceste variabile.

Introducerea unui tip de date într-un program se face folosind *specificarea de tip*. Tipul nou introdus poate purta un nume sau poate rămâne anonim. În primul caz se utilizează definiția de tipuri din diagrama din fig.4.1.

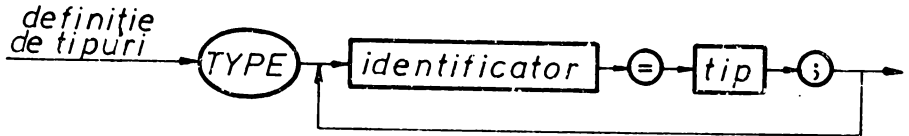


fig. 4. 1.

unde:

-*identificator* este numele asociat prin definiție tipului; identificatorul care a fost prezent într-o definiție de tip va desemna mulțimea valorilor tipului specificat și poate fi folosit ca și identificator de tip în alte definiții de tip sau în declararea variabilelor etc.;

-*tip* este explicitat în diagrama din fig.4.2, respectiv fig.4.3 și 4.4, tipurile de date urmînd a fi tratate în acest capitol și în capitolele următoare.

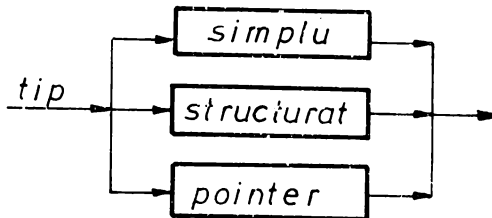


fig. 4. 2.

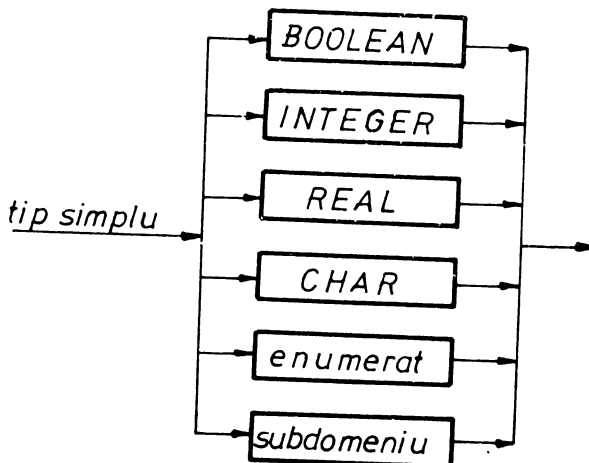


fig. 4. 3.

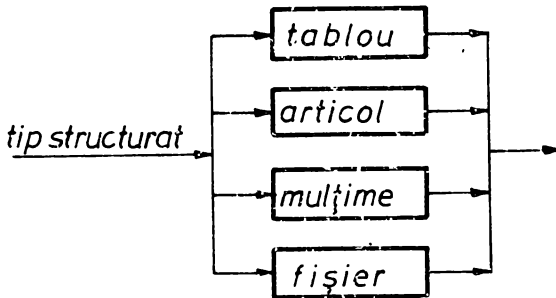


fig. 4.4.

Identificatorul de tip poate fi referit în următoarele moduri:

- a)-într-o altă definiție de tip, obținând astfel structuri complexe;
- b)-într-o declarație de variabile (capitolul II);
- c)-într-o declarație de funcție (capitolul V);
- d)-într-o listă de parametri formali (capitolul VI).

Dacă dorim să introducem una sau mai multe variabile de un anumit tip, declararea acestuia se poate face direct în declarația de variabile, obținând în acest caz *tipuri anonime* de date. Declarația de variabile are forma dată în diagrama din fig.4.5.

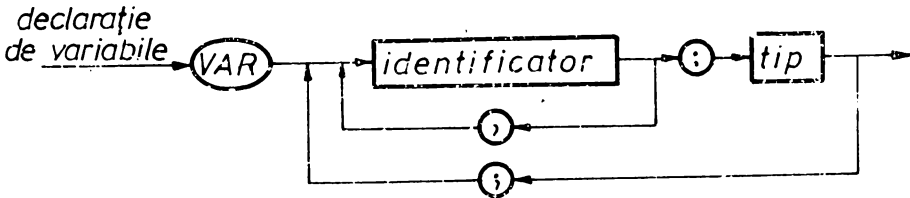


fig. 4.5.

Observații:

1)-prin definirea tipului se realizează descrierea unei mulțimi posibile de valori, nu și alocarea de zone de memorie, rezervarea realizându-se prin declararea unei variabile de tipul respectiv;

2)-definiția de tipuri este situată în program după declarația etichetelor și definiția constantelor și înaintea declarației variabilelor și a declarațiilor subprogramelor;

3)-In PASCAL două specificații de tip diferite introduc tipuri distincte. (Asupra acestei observații se va reveni cu amănunte în acest capitol, pe măsură ce se tratează tipurile de date.)

4.1. TIPURI SIMPLE

În limbajul PASCAL, tipurile simple sînt de două feluri:

- standard (sau predefinite): întreg, real, logic, caracter;
- definite de utilizator: enumerat, subdomeniu.

Toate tipurile simple, cu excepția celui real, se numesc tipuri ordinale.

Pentru a specifica un tip de date, in PASCAL se utilizează 6 modalități:

1) *Asimilarea cu anumite mulțimi de referință (N,R)*. Acestor mulțimi infinite le corespunde în fiecare implementare a limbajului PASCAL cite o submulțime finită de valori. Este cazul tipurilor standard INTEGER și REAL. Pe lângă acestea mai există și tipurile predefinite CHAR reprezentind o mulțime finită și ordonată de caractere și BOOLEAN care reprezintă mulțimea valorilor logice (TRUE și FALSE).

2) *Enumerarea componentelor mulțimii*, obținindu-se tipul enumerat. In cadrul acestor mulțimi componentele sînt ordonate conform enumerării.

3) *Stabilirea unei submulțimi a unui tip ordinal*, in care fiecare element are un loc bine definit. Avînd în vedere că toate tipurile ordinale reprezintă o mulțime cu cardinalitate cunoscută, se poate lua în considerare o submulțime a acestora clar definită de elementul cel mai mic și cel mai mare al submulțimii. Obținem astfel un nou tip finit, numit tip subdomeniu. Tipul inițial devine tipul asociat subdomeniului.

4) *Obținerea unor tipuri structurate pornind de la tipurile simple, prin operațiile:*

-indexarea cu un tip ordinal a unui tip de date deja definit (tablou);

-compunerea mai multor tipuri de date (înregistrare);

-formarea mulțimii submulțimilor de valori ale unui tip ordinal (mulțime);

-înșiruirea unui număr neprecizat de componente de același tip (fișier);

5) *Utilizarea unui tip definit în crearea tipurilor pointer* (capitolul VI.);

6) *Echivalarea cu un tip deja definit*, obținind o nouă denumire a tipului respectiv.

4.1.1. Tipuri simple standard

Cele patru tipuri simple standard sînt desemnate prin identificatorii predefiniți: INTEGER, REAL, BOOLEAN și CHAR.

Tipul întreg. Tipul întreg, desemnat prin identificatorul INTEGER semnifică o submulțime a numerelor întregi, cuprinsă între -MAXINT și MAXINT.

In cazul implementării HP4TM, MAXINT = 32767.

In exemplul următor, variabilele NR1 și NR2 sînt declarate de tip întreg: VAR NR1, NR2: INTEGER;

Operațiile aritmetice definite între valori întregi sînt date de tabelul T.4.1.

Tabel T.4.1.

Operator	Operație
*	înmulțire
/	împărțire (rezultatul este real)
DIV	împărțire întreagă
MOD	restul împărțirii întregi
+	adunare
-	scădere

Observație:

Rezultatele operațiilor asupra valorilor întregi reproduc rezultatele din aritmetica numerelor întregi.

Asupra valorilor întregi se pot aplica operatorii relaționali cuprinși în tabelul T.4.2.

Tabel T.4.2.

Operator	Operație
=	egal
<>	diferit
<	mai mic
>	mai mare
<=	mai mic sau egal
>=	mai mare sau egal

În limbajul PASCAL sînt definite funcții standard care produc rezultate de tip întreg. Aceste funcții sînt date în tabelul T.4.3.

Tabel T.4.3.

Nume funcție	Argument	Semnificația rezultatului
ABS(X)	întreg	valoarea absolută a lui X
SQR(X)	întreg	pătratul lui X
SUCC(X)	întreg	succesorul lui X
PRED(X)	întreg	predecesorul lui X
ORD(X)	întreg	numărul de ordine al lui X
TRUNC(X)	real sau întreg	cel mai mare întreg mai mic sau egal cu X pentru X pozitiv sau cel mai mic întreg mai mare sau egal cu X dacă X este negativ
ROUND(X)	real sau întreg	cel mai apropiat întreg de X
ENTIER(X)	real sau întreg	cel mai apropiat întreg mai mic sau egal cu X

Observații:

1)-funcțiile TRUNC, ROUND și ENTIER se numesc funcții de transfer deoarece realizează transferul de la un tip (cel real) la un alt tip (cel întreg);

2)-dacă în urma operațiilor aritmetice sau a aplicării funcțiilor, rezultatul care este atribuit unei variabile de tip întreg este în afara intervalului [-MAXINT, MAXINT], se semnalează eroare în timpul execuției;

3)-funcția ENTIER este specifică implementării HP4TM.

În programul FUNCINT (P IV.1) sînt cuprinse exemple de utilizare a funcțiilor din tabelul T.4.3.

```

B150 10 PROGRAM FUNCINT;
B150 20 BEGIN
B162 30 WRITELN(CHR(16));
B16C 40 WRITELN('ABS(25)=' ,ABS(25);2);
B191 50 WRITELN('ABS(-25)=' ,ABS(-25));
B1B7 60 WRITELN('SQR(25)=' ,SQR(25));
B1DB 70 WRITELN('SUCC(25)=' ,SUCC(25));
B1FC 80 WRITELN('PRED(25)=' ,PRED(25));
B21D 90 WRITELN('TRUNC(25.6)=' ,TRUNC(25.5));
B246 100 WRITELN('ROUND(25.6)=' ,ROUND(25.6));
B26F 110 WRITELN('ENTIER(25.6)=' ,ENTIER(25.6));
B299 120 WRITELN('TRUNC(-25.6)=' ,TRUNC(-25.6));
B2CB 130 WRITELN('ROUND(-25.6)=' ,ROUND(-25.6));

```

```

B2FD 140 WRITELN('ENTIER(-25.6)', ENTIER(-25.6));
B32F 150 WRITELN('ROUND(-6.5)=' , ROUND(-6.5));
B360 160 WRITELN('ROUND(-6.51)=' , ROUND(-6.61));
B392 170 WRITELN('TRUNC(1.9)=' , TRUNC(1.9));
B3BA 180 WRITELN('ROUND(1.9)=' , ROUND(1.9));
B3E2 190 WRITELN('ENTIER(1.9)=' , ENTIER(1.9));
B40B 200 WRITELN('TRUNC(-1.9)=' , TRUNC(-1.9));
B43C 210 WRITELN('ENTIER(-1.9)=' , ENTIER(-1.9));
B46E 220 WRITELN('ROUND(-1.9)=' , ROUND(-1.9));
B49F 230 WRITELN('TRUNC(-1.1)=' , TRUNC(-1.1));
B4D0 240 WRITELN('ENTIER(-1.1)=' , ENTIER(-1.1));
B502 250 WRITELN('ROUND(-1.1)=' , ROUND(-1.1));
B533 260 WRITELN('TRUNC(1.1)=' , TRUNC(1.1));
B55B 270 WRITELN('ENTIER(1.1)=' , ENTIER(1.1));
B584 280 WRITELN('ROUND(1.1)=' , ROUND(1.1));
B5AC 285 WRITELN(CHR(16));
B586 290 END($P).

```

```

ABS(25)=25
ABS(-25)=25
SQR(25)=625
SUCC(25)=26
PRED(25)=24
TRUNC(25.6)=25
ROUND(25.6)=26
ENTIER(25.6)=25
TRUNC(-25.6)=-25
ROUND(-25.6)=-26
ENTIER(-25.6)=-26
ROUND(-6.5)=-6
ROUND(-6.51)=-7
TRUNC(1.9)=1
ROUND(1.9)=2
ENTIER(1.9)=1
TRUNC(-1.9)=-1
ENTIER(-1.9)=-2
ROUND(-1.9)=-2
TRUNC(-1.1)=-1
ENTIER(-1.1)=-2
ROUND(-1.1)=-1
TRUNC(1.1)=1
ENTIER(1.1)=1
ROUND(1.1)=1

```

P. IV.1.

Programul ALGEUCLID (P. IV.2) determină cel mai mare divizor comun a două numere naturale utilizând algoritmul lui Euclid. In acest program in linia 10 sînt declarate variabilele de tip întreg N, M, C, R. In linia 120 sînt utilizați operatorii -, DIV și *.

```

AE36 10 PROGRAM ALGEUCLID;
AE36 20 VAR N,M,C,R:INTEGER;
AE3F 30 BEGIN
AE48 40 WRITE('N=');
AE55 50 READ(N);
AE58 60 WRITE('M=');
AE68 70 READ(M);
AE6E 80 WRITELN;
AE71 90 WRITELN('N=',N);
AE8A 100 WRITELN('M=',M);
AEA3 110 REPEAT
AEA3 120 C:=N;
AEAC 130 R:=M MOD N;
AEB8 140 M:=N;
AEC0 150 N:=R;
AEC6 160 UNTIL R=0;
AED8 170 WRITELN('CMMDC=',C);
AEF5 180 WRITELN
AEF5 190 END ($P).

```

```
N=6
M=120
CMMDC=6
```

```
N=7121
M=88
CMMDC=1
```

```
N=100
M=1230
CMMDC=10
```

```
N=90
M=65
CMMDC=1
```

P. IV. 2.

In programul IMPARTIRE (P.IV.3) se calculează valoarea citului a două numere întregi cu un număr determinat de cifre în partea fracționară. "Construirea" părții fracționare se realizează prin împărțiri succesive a restului obținut.

```

20 PROGRAM IMPARTIRE;
AF1E 30 ( CONSTRUIREA CITULUI CU UN NUMAR DETERMINAT
AF1E 35 DE CIFRE LA PARTEA ZECIMALA )
AF1E 40 VAR A,B,C,N,I: INTEGER;
AF27 50 BEGIN
AF30 60 WRITE('A='); READ(A);
AF43 70 WRITE('B='); READ(B);
AF56 80 WRITE('N='); READ(N);
AF69 90 WHILE B<>0 DO
AF7E 100 BEGIN
AF7E 110 WRITELN(CHR(13));
AF88 120 WRITE('A=',A:3,' B=',B:3,' A/B=',A DIV B:3,'.:1);
AFF1 130 FOR I:=1 TO N DO
B00F 140 BEGIN
B012 150 A:=(A MOD B)*10;
B026 160 WRITE(A DIV B:1)
B038 170 END;
B03F 175 WRITELN;
B042 180 WRITE(CHR(13));
B049 190 READLN;
B04C 200 WRITE('A='); READ(A);
B05F 210 WRITE('B='); READ(B);
B072 220 WRITE('N='); READ(N);
B085 230 END;
B088 240 END {$P}.

```

```

A= 5 B= 4 A/B= 1.250
A=123 B= 4 A/B= 30.7500000000
A= 32 B= 2 A/B= 16.0
A= 9 B= 65 A/B= 0.138461
A= 4 B= 5 A/B= 0.800000
A= 1 B= 3 A/B= 0.3333
A= 1 B= 10 A/B= 0.10000
A=633 B= 23 A/B= 27.52173913
A=532 B= 21 A/B= 25.333
A= 6 B= 5 A/B= 1.20

```

P. IV. 3.

Tipul real. Tipul real este desemnat prin identificatorul predefinit REAL și este format dintr-o submulțime finită a numerelor raționale, cuprinsă între două limite ce diferă de la o implementare la alta.

Intervalul de valori în care se poate lucra utilizând implementarea HP4TM este $[-3.4E38, +3.4E38]$, însă numerele din vecinătatea lui 0, cuprinse între $(-5.9E-39, 5.9E-39)$ se aproximează cu 0.

Datorită reprezentării în virgulă mobilă, precizia este de 7 cifre semnificative. De aceea nu are sens să utilizăm mai mult decât 7 cifre pentru specificarea mantisei unui număr real, întrucât cifrele în plus vor fi ignorate.

Exemplu:

Numărul 0.000123456 în reprezentarea internă pierde ultimele două cifre, deci este reprezentat mai puțin precis decât $1.23456E-4$.

În timp ce rezultatele operațiilor asupra valorilor întregi sînt exacte, în cazul tipului real în cele mai multe cazuri, rezultatele sînt aproximative, din cauza rotunjirilor datorate folosirii unui număr finit de cifre la reprezentarea în virgulă mobilă.

Operațiile aritmetice definite între valori reale sînt date în tabelul T.4.4.

Tabel T.4.4.

Operator	Operație
*	înmulțire
/	împărțire
+	adunare
-	scădere

Operatorii relaționali menționați la tipul întreg sînt valabili și pentru tipul real. Rezultatul aplicării lor este tot o valoare logică.

Exemplu:

1.5>1.2 are valoarea TRUE;
 1.5>2.0 are valoarea FALSE;
 1.567896307=1.567896399 are valoarea TRUE.

Funcțiile standard care produc rezultate de tip real sînt date în tabelul T.4.5.

Programul FUNCINT (P.IV.1) conține cîteva exemple de utilizare a funcțiilor.

Observații:

- 1)-în cazul argumentelor întregi, la aplicarea funcțiilor SIN, COS, TAN, ARCTAN, LN, EXP, SQRT, se efectuează în prealabil conversia valorii argumentelor din întreg în real;
- 2)-operatorii aritmetici și unele funcții standard (EXP, SQR) pot provoca depășire flotantă;
- 3)-funcția FRAC este specifică implementării HP4TM;

4)-valorile reale nu pot fi folosite în următoarele cazuri:

- argument pentru funcțiile standard SUCC, PRED și ORD
- index pentru tablouri;
- contor pentru instrucțiunea FOR;
- tip de bază pentru o mulțime;
- selector în CASE;
- definire tip subdomeniu.

Tabel T.4.5.

Nume	Tip argument	Semnificația rezultatului funcției
ABS(X)	real	valoarea absolută a argumentului
SQR(X)	real	pătratul valorii argumentului
SIN(X)	real sau întreg	sinusul valorii argumentului exprimat în radiani
COS(X)	real sau întreg	cosinusul valorii argumentului exprimat în radiani
TAN(X)	real sau întreg	tangenta valorii argumentului exprimată în radiani
ARCTAN(X)	real sau întreg	unghiul, exprimat în radiani, a cărui tangentă este X
LN(X)	real sau întreg	logaritm natural din X
EXP(X)	real sau întreg	exponențiala valorii argumentului
SQRT(X)	real sau întreg	radical din X (X)≥0)
FRAC(X)	real	partea fracționară a lui X FRAC(X)=X-ENTIER(X)

Tipul logic. Tipul logic, desemnat prin identificatorul predefinit BOOLEAN este format din mulțimea valorilor logice TRUE și FALSE, doi identificatori predefiniți având valoarea logică fals respectiv adevărat.

Operatorii logici care se aplică asupra valorilor logice sint:

- a) operatorii binari:
 - OR (disjuncție logică);
 - AND (conjunție logică);
- b) operator unar:
 - NOT (negație logică).

În mulțimea formată din elementele TRUE și FALSE se introduce relația de ordine FALSE<TRUE. Asupra tipului logic se pot aplica operatorii relaționali dați în tabelul T.4.6.

Tabel T.4.6.

Operator	Semnificație
=	echivalență
<>	nonechivalență (sau exclusiv)
<=	implicație

Funcțiile standard care admit argumente de tip logic sint date în tabelul T.4.7.

Tabel T.4.7.

Funcția	Tipul valorii	Semnificația
ORD(X)	logic	numărul de ordine al argumentului X
PRED(X)	logic	predecesorul argumentului X
SUCC(X)	logic	succesorul argumentului X

Funcțiile standard care produc rezultate de tip logic sînt cuprinse în tabelul T.4.8.

Tabel T.4.8.

Funcția	Tipul argumentului	Semnificația
ODD(X)	întreg	este TRUE pentru x impar
EOLN	-	este TRUE dacă valoarea care urmează valorii citite este caracterul sfîrșit de linie

Observație:

În implementările limbajului PASCAL în care există tipul fișier, funcția EOLN se poate apela și cu argument *fișier de text*; de asemenea, există funcția EOF de argument *fișier* care va furniza valoarea TRUE dacă următorul caracter de la intrare este marca de sfîrșit de fișier.

Tipul caracter. Tipul caracter este desemnat prin identificatorul standard CHAR și este format, în cazul implementării HP4TM, din setul *extins de caractere ASCII*.

Mulțimea caracterelor și ordinea acestora în mulțime diferă de la un tip de calculator la altul datorită diferențelor din hardware-ul sistemului de calcul. În cazul studiat, ordinea elementelor în mulțime este dată de codurile ASCII ale caracterelor.

Asupra valorilor de tip caracter se pot aplica operatorii relaționali.

Funcțiile standard care produc ca rezultat un caracter sînt date în tabelul T.4.9.

Tabel T.4.9.

Funcția	Argumentul	Semnificația rezultatului
CHR(X)	întreg	caracterul care are codul ASCII X
SUCC(X)	32<X<=164 caracter	succesorul caracterului X în mulțimea caracterelor
PRED(X)	caracter	predecesorul caracterului X în mulțimea caracterelor
INCH	-	execută baleierea tastaturii și dacă a fost apăsată o tastă, returnează caracterul reprezentat de această tastă, iar dacă nu, se returnează CHR(0)

Acestor funcții li se alătură funcția ORD, de argument caracter, care returnează codul ASCII al caracterului căruia i se aplică.

Se observă că au loc relațiile:

ORD(CHR(I))=I I - întreg din [32,164]
CHR(ORD(C))=C C - caracter ASCII

Tabel T.4.10.

Cod ASCII	caracter	Cod ASCII	caracter	Cod ASCII	caracter
32	spațiu	77	M	122	z
33	!	78	N	123	{
34	"	79	O	124	
35	#	80	P	125	}
36	\$	81	Q	126	~
37	%	82	R	127	
38	&	83	S	128	spațiu
39	'	84	T	129	
40	(85	U	130	c
41)	86	V	131	a
42	*	87	W	132	r
43	+	88	X	133	a
44	,	89	Y	134	c
45	-	90	Z	135	t
46	.	91	[136	e
47	/	92	\	137	r
48	0	93]	138	e
49	1	94	^	139	
50	2	95	_	140	g
51	3	96	`	141	r
52	4	97	a	142	a
53	5	98	b	143	f
54	6	99	c	144	i
55	7	100	d	145	c
56	8	101	e	146	e
57	9	102	f	147	
58	:	103	g	148	d
59	;	104	h	149	e
60	<	105	i	150	f
61	=	106	j	151	i
62	>	107	k	152	n
63	?	108	l	153	i
64	@	109	m	154	t
65	A	110	n	155	e
66	B	111	o	156	
67	C	112	p	157	
68	D	113	q	158	
69	E	114	r	159	
70	F	115	s	160	
71	G	116	t	161	
72	H	117	u	162	
73	I	118	v	163	
74	J	119	w	164	
75	K	120	x		
76	L	121	y		

În afară de codurile menționate în tabelul T.4.10., există următoarele posibilități de utilizare a funcției CHR pentru generarea codurilor de control:

- CHR(0) - semnifică șir de caractere vid;
- CHR(8) - mută cursorul înapoi cu un pas pe ecran cu ștergerea caracterului;
- CHR(12) - șterge ecranul sau execută salt la pagină nouă dacă ieșirea este comutată la imprimantă
- CHR(13) - execută retur de car și avans de rând;
- CHR(16) - comută ieșirea între ecran și imprimantă.

In programul **VOCALE** (P.IV.4) avem exemple de utilizare a variabilelor de tip caracter. Acest program numără vocalele aflate într-un șir de caractere introdus de la tastatură.

```

AE21 10 PROGRAM VOCALE;
AE21 20 CONST P=CHR(15);
AE21 30 VAR C:CHAR;
AE2A 40     NA,NE,NI,NO,NU:INTEGER;
AE2A 50 BEGIN
AE33 60     NA:=0;NE:=0;NI:=0;NO:=0;NU:=0;
AE51 70     WRITELN(P,'INTRODU O LINIE CU LITERE',CHR(13),P);
AE39 80     READLN;
AE8C 90     WHILE NOT EOLN DO
AE98 100        BEGIN
AE98 110         READ(C);WRITE(P,C,P);
AEAE 120         CASE C OF
AEB1 130           'A':NA:=NA+1;
AEC0 140           'E':NE:=NE+1;
AECF 150           'I':NI:=NI+1;
AEDE 160           'O':NO:=NO+1;
AEED 170           'U':NU:=NU+1;
AEFC 180           'B','Z','.',',',' ',';',';':BEGIN END;
AF1A 190         END;
AF1A 200         END;
AF1D 210         WRITELN(P);WRITELN;
AF28 220         WRITELN('NUMARUL DE VOCALE DIN LINIE:');
AF52 230         WRITELN(' A=',NA:2);
AF6F 240         WRITELN(' E=',NE:2);
AF8C 250         WRITELN(' I=',NI:2);
AFA9 260         WRITELN(' O=',NO:2);
AFC6 270         WRITELN(' U=',NU:2);
AFE3 280         WRITELN(P);
AFEB 290     END ($P).

```

```

INTRODU O LINIE CU LITERE
ACEASTA ESTE LINIA CU LITERE

```

```

NUMARUL DE VOCALE-DIN LINIE:
A= 4
E= 5
I= 3
O= 0
U= 1

```

P. IV. 4.

4.1.2. Tipuri simple definite de utilizator

Specificarea unui tip de date se poate realiza și în următoarele moduri:

a)-enumerarea componentelor mulțimii, obținând astfel tipul *enumerat*;

b)-stabilirea unei submulțimi a unui tip simplu, diferit de cel real, obținând astfel tipul *subdomeniu*; tipul simplu inițial se numește tipul asociat subdomeniului.

Tipul enumerat. Deseori, în programe se utilizează date numerice în locul unor valori nenumerice deoarece nu există posibilitatea de a le introduce astfel în program.

De exemplu, pentru zilele săptămânii putem stabili următoarea corespondență:

```

luni      - 1
marți     - 2
miercuri  - 3
joi       - 4
vineri    - 5
sâmbătă   - 6
duminică  - 7

```

În acest context 1 reprezintă ziua de luni, 2 reprezintă ziua de marți etc...

Pentru a evita astfel de codificări, în PASCAL s-a introdus un nou tip de date, tipul enumerat, sporind astfel gradul de înțelegere și lizibilitatea programului.

Introducerea unui tip enumerat se realizează conform diagramei din fig.4.6.

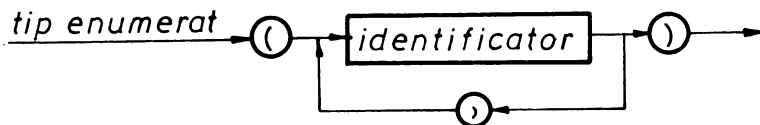


fig.4.6.

Identificatorii prezenți între paranteze precizează valorile noului tip, putând fi utilizate ca și *constante* în program. Noul tip poate rămâne anonim sau poate primi un nume când se utilizează definiția de tipuri. Numărul maxim de identificatori este 256.

Specificarea tipului enumerat introduce o mulțime ordonată de valori, ordine dată de locul identificatorului în cadrul listei. Identificatorii prezenți într-o listă a unui tip enumerat nu pot fi utilizați în definirea unui alt tip enumerat, deoarece s-ar crea ambiguități cu privire la numărul de ordine al identificatorului respectiv.

Deoarece mulțimea valorilor unui tip enumerat este ordonată, între elementele sale se pot aplica operatorii relaționali =, <, <>, <=, >=, >, rezultatul aplicării unui asemenea operator fiind o valoare logică.

Exemplu:

```
TYPE ZILE=(LUNI, MARTI, MIERCURI, JOI, VINERI, SIMBATA, DUMINICA);
VAR ZI: ZILE;
```

În exemplul anterior am definit un tip enumerat cu numele ZILE cu 7 elemente.

Considerind definiția de tipuri precedentă și declarația de variabile, sint permise atribuiri de forma:

```
ZI:=LUNI;
ZI:=DUMINICA;
```

Pentru tipul enumerat, ca și pentru celelalte tipuri ordinale se pot aplica funcțiile standard din tabelul T.4.11.

Tabel T.4.11.

Funcția	Semnificația
SUCC(X)	succesorul argumentului
PRED(X)	predecesorul argumentului
ORD(X)	are valoare întreagă și determină rangul argumentului în enumerare, știind că primul element are rangul 0, iar între două elemente consecutive rangul crește cu 1

În reprezentarea internă mărimile de tip enumerat sint date de întregul corespunzător numărului de ordine. Deci

reprezentările interne sînt niște întregi, însă spre deosebire de mărimile de tip întreg, între ele nu se pot aplica operatorii aritmetici și deci nu pot apare ca și operanzi în expresii aritmetice. Sînt permise doar atribuirii de elemente componente ale unui tip enumerat variabilelor declarate de tipul respectiv.

Observații:

1)-tipul BOOLEAN poate fi considerat ca un tip enumerat predefinit avînd ca și elemente FALSE și TRUE;

2)-valorile tipurilor enumerat nu pot fi nici citite și nici scrise.

În programul TIPENUM (P.IV.5) sînt cîteva exemple de utilizare a tipului enumerat. Deoarece nu putem tipări o valoare de tip enumerat în aceste programe se tipăresc numerele de ordine ale acestor valori.

```

HEB9      5 ( TIPUL ENUMERARE )
AE59      10 PROGRAM TIPENUM;
AE59      20 TYPE ZILE=(LUNI,MARTI,MIERCURI,JOI,VINERI,SIMBATA,DUMINECA);
AE59      30 VAR ZI:ZILE;
AE62      40 BEGIN
AE6B      50 WRITELN(CHR(13));
AE75      60 WRITELN('ORD(LUNI)=' ,ORD(LUNI));
AE97      70 WRITELN('ORD(MARTI)=' ,ORD(MARTI));
AEBB      80 WRITELN('ORD(SUCC(JOI))=' ,ORD(SUCC(JOI)));
AEE4      85 WRITE(CHR(13));
AEEB      90 END($P).

```

```

ORD(LUNI)=0
ORD(MARTI)=1
ORD(SUCC(JOI))=4

```

P. IV.5.

Tipul subdomeniu. Deoarece uneori cunoaștem limitele în care o variabilă de tip ordinal poate lua valori, în PASCAL s-a introdus posibilitatea definirii tipului variabilei ca un interval al tipului ordinal respectiv. Acest nou tip poartă numele de subdomeniu sau interval.

Specificarea noului tip este dată în diagrama din fig.4.7.

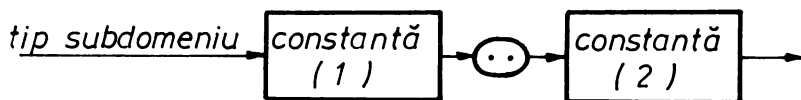


fig.4.7.

unde:

-constantă reprezintă constantă sau identificator de constantă de tip ordinal, reprezentînd limita inferioară și respectiv cea superioară a intervalului din care se alimentează variabilele de tipul subdomeniu.

Evident este necesar ca amîndouă constantele să fie de același tip ordinal și prima constantă să fie strict mai mică decît a doua. Tipul celor două constante se numește *tip de bază* al subdomeniului.

Operatorii relaționali și aritmetici precum și funcțiile standard corespunzătoare unui anumit tip ordinal sînt valabile pentru orice subdomeniu al său.

Se recomandă utilizarea acestui tip de date deoarece:

- pe parcursul rulării, în program se verifică încadrarea mărimii de tip subdomeniu între limitele declarate;

- utilizarea mărimilor de tip subdomeniu contribuie la creșterea clarității programului, prin explicitarea domeniului în care o variabilă ia valori;

- declararea componentelor de tip subdomeniu în cadrul unei structuri are ca efect reducerea spațiului de memorie necesar reprezentării structurii.

Exemplu:

```
VAR A,B: INTEGER;
```

Dacă variabila A va avea în program valoarea minimă 1 și cea maximă 100, iar B are valoarea minimă -10 și cea maximă 1000, folosind tipul subdomeniu putem avea următoarea declarație:

```
VAR A:1..100;
```

```
    B:-10..1000;
```

4.2. TIPURI STRUCTURATE

În limbajul PASCAL pot fi definite structuri complexe pornind de la tipurile simple. Tipurile structurate, implementate în HP4TM sînt:

a)-tipul tablou, rezultat din compunerea unor elemente de același tip, obținînd astfel o structură omogenă;

b)-tipul înregistrare, rezultat din compunerea unor elemente de tipuri diferite, obținînd astfel o structură neomogenă;

c)-tipul mulțime.

4.2.1. Tipul tablou

Tabloul este un ansamblu omogen format dintr-un număr fix de elemente componente, toate fiind de același tip.

Tipul elementelor componente poartă numele de tip de bază al tabloului.

Ordinea elementelor în cadrul acestui tip este dată de asocierea acestora cu valorile unui tip ordinal, numit tipul de indexare al tabloului. Astfel, această structură transpune în programare mulțimile indexate din matematică.

Pentru a preciza o componentă a tabloului se utilizează numele acestuia și o valoare a tipului de indexare numită indice, care indică locul elementului în cadrul ansamblului.

La definirea unui tablou trebuie precizate:

a)-tipul elementelor (tipul de bază al tabloului);

b)-tipul pentru indici (tipul de indexare).

Odată cu precizarea tipului pentru indici se fixează și numărul componentelor tabloului care este dat de cardinalitatea tipului indicelui.

Într-un program PASCAL, pentru a introduce un tablou vom folosi specificarea de tip conform diagramei din fig.4.8.

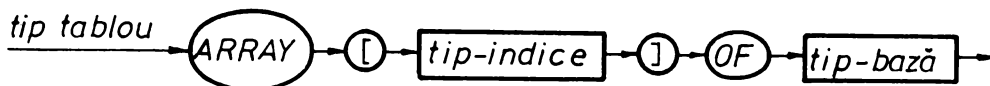


fig. 4.8.

unde:

- *tip-indice* este specificarea tipului de indexare și este un tip ordinal diferit de INTEGER: (enumerat, BOOLEAN, CHAR, subdomenii ale acestora, precum și ale tipului INTEGER);
- *tip-bază* este tipul de bază al tabloului și trebuie să fie cunoscut în momentul definiției acestuia.

O specificație de tip tablou reprezintă mulțimea tuturor tablourilor formate din elemente de tipul *tip-bază* indexate cu valori din mulțimea definită de *tip-indice*. Acest tip poate primi un nume (dacă este prezent în TYPE) sau poate rămâne anonim.

Cardinalitatea unui tip tablou este

$$n = a^b$$

unde: a = cardinalitatea tipului *tip-bază*;

b = cardinalitatea tipului *tip-indice*.

Pentru a putea lucra cu tablouri trebuie declarate variabile de tipul respectiv, operație ce trebuie realizată cu atenție dacă dorim *manipularea tablourilor* în ansamblu. Aceasta constituie o facilitate a limbajului PASCAL față de multe alte limbaje de programare.

Reamintim faptul că *două specificații diferite de tip introduc tipuri distincte*.

Exemplu:

```
TYPE SIR=ARRAY[1..10] OF INTEGER;
```

```
  SUBM=1..10;
```

```
VAR  A,B:SIR;
```

```
  X,Y:ARRAY[SUBM] OF INTEGER;
```

În acest exemplu avem tipul structurat cu numele SIR care definește un tablou de 10 elemente numere întregi. Tipul de bază pentru tablou este INTEGER, iar tipul indicelui este subdomeniu de INTEGER. În declarația de variabile sînt declarate A și B de tip SIR (adică șiruri de cîte 10 elemente numere întregi). În aceeași declarație de variabile sînt declarate X și Y ca șiruri de 10 elemente numere întregi deoarece SUBM este un subdomeniu de întregi cu 10 elemente. Tipul variabilelor X și Y este anonim. Deși aparent A și B au același tip cu X și Y, ținînd cont de observația că două specificații diferite de tip introduc tipuri diferite, rezultă că A și B au un tip, iar X și Y alt tip. În consecință este corectă o atribuire de felul:

A:=B; sau X:=Y;

dar nu este corectă o atribuire de genul:

A:=X; sau Y:=B;

O componentă a tabloului se selectează utilizînd numele variabilei tablou urmat de o expresie numită *expresie indiceală* încadrată de paranteze drepte, așa cum precizează diagrama din fig.4.9.

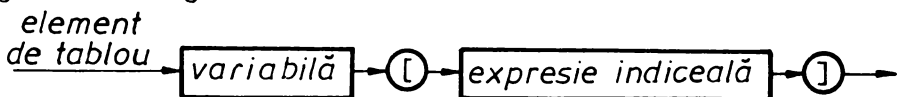


fig.4.9.

unde:

-*expresie indiceală* este o expresie a cărei valoare trebuie să aparțină tipului de indexare.

Deoarece pentru tip-bază nu s-a impus o restricție, el poate fi, la rândul lui, un tip tablou. Astfel, specificarea de tablou din exemplul următor se referă la un tablou cu elemente tablouri, adică la tablouri cu două dimensiuni. Mai mult, când tipul de bază este un tablou cu două dimensiuni obținem un tablou cu trei dimensiuni etc...

Exemplu:

```
VAR A: ARRAY[1..10] OF ARRAY[-1..5] OF INTEGER;
```

introduce variabila A ca tablou cu 10 elemente, fiecare element fiind la rândul său un tablou cu 7 elemente numere întregi, adică un tablou bidimensional.

Pentru a simplifica scrierea în cazul tablourilor cu mai multe dimensiuni, se recomandă ca specificația tipului tablou să se facă conform diagramei din fig.4.10.

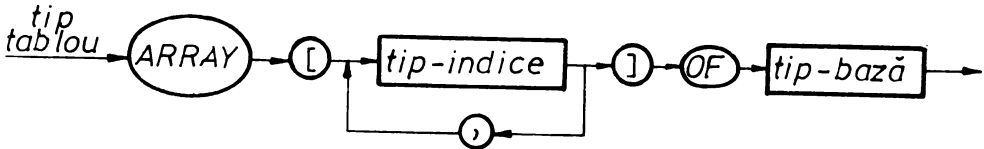


fig. 4. 10.

Referirea unui element se face cu ajutorul unei variabile indexate a cărei sintaxă este dată de diagrama din fig.4.11.

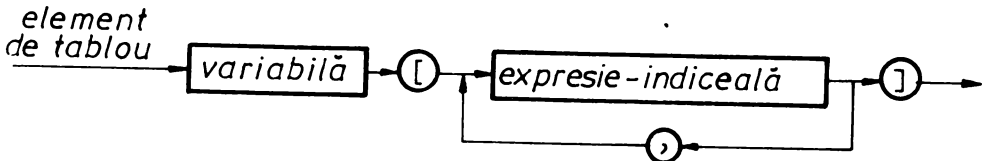


fig. 4. 11.

Pentru a referi un element, în HP4TM este permisă atât scrierea:

```
id-tabloul[ind-1,ind-2,...,ind-n]
```

cât și

```
id-tabloul[ind-1][ind-2]...[ind-n]
```

Programul TABLCHAR (P.IV.6) rotește cu 90° o figură generată din caractere.

```
AF91 10 PROGRAM TABLCHAR;
AF91 20 CONST N=8;
AF91 30 VAR I,J:INTEGER;
AF9A 40 M:ARRAY [1..64] OF CHAR;
AF9A 50 A,B:ARRAY [1..N,1..N] OF CHAR;
AF9A 60 BEGIN
AF9A 70 M:= ** ** * * ***** ** ** * * **
AFF3 80 FOR I:=1 TO 8 DO
B00D 90 FOR J:=1 TO 8 DO
B02A 100 A[I,J]:=M[(I-1)*8+J];
B08F 110 ( FOR I:=0 TO N DO
B08F 120 FOR J:=0 TO N DO READ(A[I,J]))
B08F 130
B08F 140 (TIPARIRE A[I,J])
B08F 150
B08F 160 PAGE;
B094 170 FOR I:=1 TO N DO
B0AE 180 BEGIN
B0B1 190 FOR J:=1 TO N DO WRITE(A[I,J]);
B106 200 WRITELN;
B109 210 END;
B10C 220
B10C 230 (ROTIRE CU 90 GRADE)
B10C 240
```



```

B10C 250 FOR I:=1 TO N DO
B126 260   FOR J:=1 TO N DO B[I,J]:=A[J,I];
B182 270
B182 280   WRITELN;WRITELN;
B188 290 (TIPARIRE B[I,J])
B188 300
B188 310   FOR I:=1 TO N DO
B1D2 320     BEGIN
B1D5 330       FOR J:=1 TO N DO WRITE(B[I,J]);
B22A 340     WRITELN;
B22D 350     END;
B230 360 END {$P}.

```

```

** **
* *
*****
*****
** ** **
*****
* *
** **

```

```

**
* **** *
**** ***
****
****
*** ***
* **** *
**

```

P. IV. 6.

Un concept important folosit în PASCAL este *împachetarea* structurilor de date, introdusă pentru economisirea memoriei. Împachetarea presupune rezervarea de către compilator a unui spațiu de memorie minim structurii de date respective. Avantajul împachetării apare în cazul calculatoarelor cu lungime mare a cuvintului (16, 32 biți). Având în vedere că implementarea HP4TM este concepută pentru calculatoare personale și acestea funcționează cu microprocesor pe 8 biți, împachetarea este inefectivă.

Din punct de vedere sintactic, o structură împachetată se introduce punind cuvântul rezervat PACKED înaintea declarației.

Exemplu:

```

TYPE A=PACKED ARRAY[1..100] OF CHAR;
VAR Y:A;
X:PACKED ARRAY[1..10] OF CHAR;

```

În acest exemplu sînt declarate două variabile tablou: Y de 100, respectiv X de 10 elemente caractere, amîndouă împachetate.

În programul FIBONACCI (P.IV.7) se generează șirul lui FIBONACCI.

```

AD76 10
AD76 20 ( RULAREA PE IMPRIMANTA )
AD76 30
AD76 40 PROGRAM FIBONACCI;
AD76 50 VAR N:INTEGER;
AD7F 60 A:ARRAY[1..100] OF INTEGER;
AD7F 70 BEGIN
AD88 80 WRITELN(CHR(13));
AD92 90 WRITELN('GENERAREA PRIMILOR 20 DE TERMENI DIN SIRUL ');
ADC8 100 WRITELN(' F I B O N A C C I ');
ADF5 110 A[1]:=1;A[2]:=2;
AE3B 120 WRITELN('A ( 1)=' ,A[1];5);WRITELN('A ( 2)=' ,A[2];5);

```

```

AEB1 130   FOR N:=3 TO 20 DO
AECB 140     BEGIN
AECE 150     ACN1:=ACN-2]+ACN-1];
AF35 160     WRITELN('A(' ,N:2,')=' ,A[N]:5);
AF85 170     END;
AF88 180 END {$P}.

```

GENERAREA PRIMILOR 20 DE TERMENI DIN SIRUL
FIBONACCI

```

A( 1) = 1
A( 2) = 2
A( 3) = 3
A( 4) = 5
A( 5) = 8
A( 6) = 13
A( 7) = 21
A( 8) = 34
A( 9) = 55
A(10) = 89
A(11) = 144
A(12) = 233
A(13) = 377
A(14) = 610
A(15) = 987
A(16) = 1597
A(17) = 2584
A(18) = 4181
A(19) = 6765
A(20) = 10946

```

P. IV. 7.

Programul **ADUNSIR** (P. IV. 8) realizează suma a două șiruri.

```

AD53 10 PROGRAM ADUNSIR;
AD53 20 TYPE SIR=ARRAY[1..10] OF INTEGER;
AD53 30 VAR SIR1,SIR2,SUMA: SIR;
AD5C 40 I: INTEGER;
AD5C 50 BEGIN
AD65 60 WRITELN('INTRODU SIRURILE');
AD83 70 FOR I:=1 TO 10 DO
AD9D 80   BEGIN
ADA0 90     READ(SIR1[I],SIR2[I]);
ADE6 100    SUMA[I]:=SIR1[I]+SIR2[I];
AE4A 110    WRITELN(CHR(16));
AE54 120    WRITE('SIR1(' ,I:2,')=' ,SIR1[I]:6, ' SIR2(' ,I:2,')=' );
AECE 130    WRITELN(SIR2[I]:6, ' S(' ,I:2,')=' ,SUMA[I]:6);
AF48 140    WRITELN(CHR(16));
AF52 150    END
AF52 160 END {$P}.

```

```

SIR1( 1) = 1 SIR2( 1) = 1 S( 1) = 2
SIR1( 2) = 0 SIR2( 2) = 0 S( 2) = 0
SIR1( 3) = -1000 SIR2( 3) = 1000 S( 3) = 0
SIR1( 4) = 500 SIR2( 4) = 500 S( 4) = 1000
SIR1( 5) = 0 SIR2( 5) = 0 S( 5) = 0
SIR1( 6) = 12345 SIR2( 6) = 12345 S( 6) = 24690
SIR1( 7) = -12345 SIR2( 7) = 1000 S( 7) = -11345
SIR1( 8) = 9 SIR2( 8) = 0 S( 8) = 9
SIR1( 9) = 9 SIR2( 9) = 9 S( 9) = 18
SIR1(10) = 9 SIR2(10) = -9 S(10) = 0

```

P. IV. 8.

Programul **CONTORIZARE** (P.IV.9) este un exemplu de utilizare a indicilor de tip caracter.

```

AE88 10 ( EXEMPLU CU INDICI LITERE)
AE88 20 PROGRAM CONTORIZARE;
AE82 30 VAR A:ARRAY['A'..'I'] OF REAL;
AE91 40   POZ:INTEGER;
AE91 50   I:CHAR;
AE91 60 BEGIN
AE9A 70   WRITELN('INTRODU SIRUL');
AE85 80   POZ:=0;
AE88 90   FOR I:='A' TO 'I' DO
AE8B 100  BEGIN
AECE 110     READ(A[I]);
AF02 120     WRITELN('A(',I,')=' ,A[I]:7:2);
AF60 130     IF A[I] > 0
AF98 140       THEN
AFAD 150         POZ:=POZ+1;
AFB4 160     END;
AFC0 170   WRITELN;
AFC3 180   WRITELN('NUMARUL ELEMENTELOR POZITIVE=' ,POZ);
AFF7 190 END ($P).

```

INTRODU SIRUL

```

A(A)= 3.66
A(B)= 8.23
A(C)= 6.00
A(D)= -7.67
A(E)= 6.66
A(F)= -2.21
A(G)= 1.23
A(H)= -45.67
A(I)= 111.11

```

NUMARUL ELEMENTELOR POZITIVE=6

P. IV. 9.

Programul **INMAT** (P.IV.10) calculează produsul a două matrici cu elemente întregi.

```

AFF2 10 PROGRAM INMAT;
AFF2 20 VAR A:ARRAY [1..2,1..4] OF INTEGER;
AFFB 30   B:ARRAY [1..4,1..3] OF INTEGER;
AFFB 40   C:ARRAY [1..2,1..3] OF INTEGER;
AFFB 50   I,J,K:INTEGER;
AFFB 60
AFFB 70 BEGIN
B004 80   WRITE(CHR(13));
B00B 90   PAGE;WRITELN;
B013 100  WRITE('INTRODUCETI MATRICEA A');
B034 110  WRITELN;
B037 120  FOR I:=1 TO 2 DO
B051 130    FOR J:=1 TO 4 DO
B06E 140      BEGIN
B071 150        WRITE('A(',I:1,',' ,J:1,')=' );READ(A[I,J]);
B0E4 160      END;
B0EA 170  WRITELN;
B0ED 180  WRITE('INTRODUCETI MATRICEA B');
B10E 190  WRITELN;
B111 200  FOR I:=1 TO 4 DO
B12B 210    FOR J:=1 TO 3 DO
B148 220      BEGIN
B14B 230        WRITE('B(',I:1,',' ,J:1,')=' );READ(B[I,J]);
B1C0 240      END;
B1C6 250  FOR I:=1 TO 2 DO
B1E0 260    FOR K:=1 TO 3 DO
B1FD 270      BEGIN
B200 280        C[I,K]:=0;
B23E 290        FOR J:=1 TO 4 DO
B258 300          C[I,K]:=C[I,K]+A[I,J]*B[J,K];
B34F 310        END;
B355 320
B355 330  WRITELN;
B358 340  WRITE('MATRICEA C');

```

```

B36D 350 FOR I:=1 TO 2 DO
B387 360 BEGIN
B38A 370 WRITELN;
B38D 380 FOR J:=1 TO 3 DO WRITE(C[I,J]:4);
B3EF 390 END;
B3F2 400 WRITELN;
B3F5 410 END {$P}.

```

INTRODUCETI MATRICEA A

```

A[1,1]=5
A[1,2]=3
A[1,3]=4
A[1,4]=7
A[2,1]=8
A[2,2]=1
A[2,3]=2
A[2,4]=3

```

INTRODUCETI MATRICEA B

```

B[1,1]=7
B[1,2]=5
B[1,3]=4
B[2,1]=3
B[2,2]=2
B[2,3]=1
B[3,1]=1
B[3,2]=2
B[3,3]=3
B[4,1]=5
B[4,2]=8
B[4,3]=5

```

MATRICEA C

```

MATRICEA C
 83 95 70
 76 70 54

```

P. IV.10.

Programul TRIPASCAL (P.IV.11) afișează primele 13 nivele ale triunghiului Pascal.

```

AE24 10 PROGRAM TRIPASCAL;
AE24 20 CONST N=13;
AE24 30 VAR I,J:INTEGER;
AE2D 40 A:ARRAY [1..N] OF INTEGER;
AE2D 50 BEGIN
AE36 60 FOR I:=1 TO N DO
AE50 70 BEGIN
AE53 80 A[I]:=1;
AE76 90 FOR J:=1-1 DOWNT0 2 DO
AE90 100 A[J]:=A[J-1]+A[J];
AEFB 110 WRITE(' ',2*(N-I)+1);
AF1F 120 FOR J:=1 TO I DO
AF3D 130 WRITE(A[J]:4);
AF6B 140 WRITELN
AF6B 150 END
AF6E 160 END {$P}.

```

```

          1
         1 1
        1 2 1
       1 3 3 1
      1 4 6 4 1
     1 5 10 10 5 1
    1 6 15 20 15 6 1
   1 7 21 35 35 21 7 1
  1 8 28 56 70 56 28 8 1
 1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
1 11 55 165 330 462 462 330 165 55 11 1
1 12 66 220 495 792 924 792 495 220 66 12 1

```

P. IV.11.

4.2.2. Tipul înregistrare

Tipul înregistrare desemnează o structură formată dintr-un număr fix de componente (nu neapărat de același tip), numite *cîmpuri*.

Pentru a avea acces la elementele componente ale unei structuri de tip înregistrare, se folosesc *selectorii*, (denumiri ale componentelor tipului respectiv).

Declarația unui tip structurat ca înregistrare este dată de diagrama din fig.4.12.

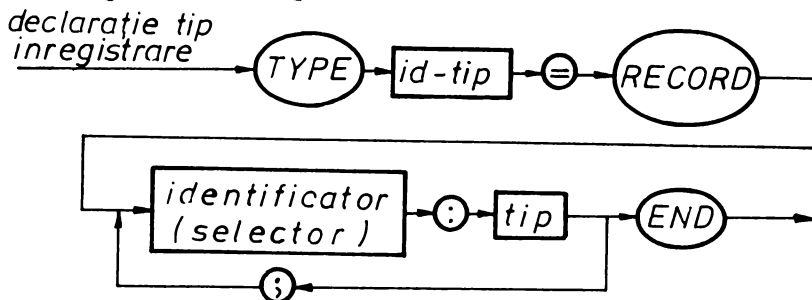


fig.4.12.

caz în care noul tip primește și un nume. Dacă nu se dorește identificarea tipului printr-un nume, vom specifica tipul structurat ca înregistrare în declarația de variabile așa cum rezultă din diagrama de sintaxă din fig.4.13.

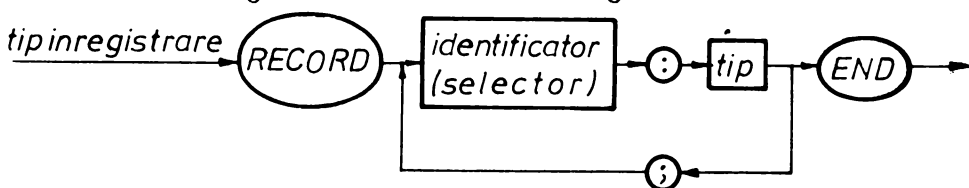


fig.4.13.

caz în care noul tip rămâne anonim.

Observații:

1) - în cazul existenței mai multor selectori, aceștia trebuie să fie *distincți* pentru a identifica cîmpurile din cadrul structurii respective;

2) - selectorii nu sînt calculabili și nu li se pot atribui valori.

Ca și în cazul celorlalte tipuri, pentru a putea lucra cu o înregistrare trebuie declarată o variabilă de tipul respectiv.

Exemplu:

```

TYPE INREGISTRARE=RECORD
    NUME: ARRAY[1..20] OF CHAR;
    MATR: 1..999
END;
  
```

Componentele unui tip înregistrare pot fi, la rîndul lor, structurate ca înregistrare.

Exemplu:

```

TYPE ELEV=RECORD
    NUME: ARRAY[1..20] OF CHAR;
    DATA: RECORD
        AN: 1900..2000;
        LU: 1..12;
        ZI: 1..31
    END;
    ADRESA: ARRAY[1..30] OF CHAR
END;

```

Dacă două sau mai multe componente succesive sînt de același tip, tipul lor poate fi specificat o singură dată, precedat de lista selectorilor cîmpurilor identice, așa cum rezultă din diagrama de sintaxă din fig.4.14.

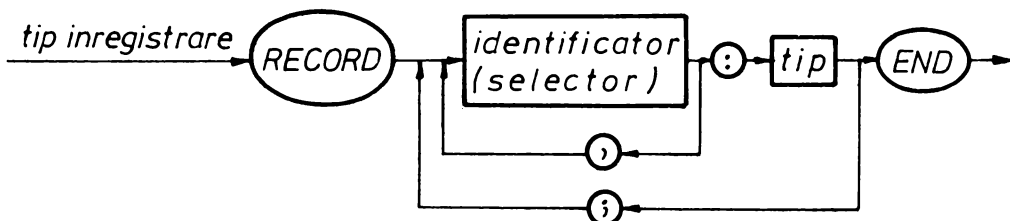


fig. 4. 14.

Exemplu:

Pentru următoarea structură de tip înregistrare

```

TYPE ARTICOL=RECORD

```

```

    NUME: ARRAY[1..10] OF CHAR;
    PRENUME: ARRAY[1..10] OF CHAR;
    MARCA: INTEGER;
    VIRSTA: INTEGER
END;

```

putem cumula selectorii consecutivi de același tip astfel:

```

TYPE ARTICOL=RECORD
    NUME,PRENUME: ARRAY[1..10] OF CHAR;
    MARCA,VIRSTA: INTEGER
END;

```

Accesul la o anumită componentă a unei variabile înregistrare se face prin intermediul *specificatorului de cîmp* care este o expresie formată din numele variabilei și numele selectorului, după modelul din diagrama din fig.4.15.

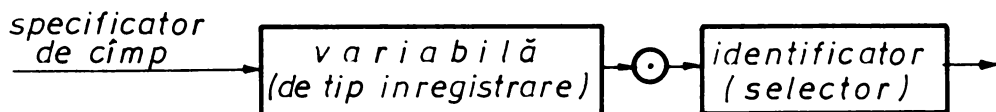


fig. 4. 15.

Pentru tipul înregistrare ELEV definit mai sus, cu declarația de variabile:

```

VAR E: ELEV;

```

putem avea următoarele variabile selectate:

```

E. NUME
E. DATA. AN
E. ADRESA

```

Observăm că în cazul în care o componentă a unui tip structurat ca înregistrare este la rîndul ei structurată ca înregistrare, selectorii și expresiile indiceale se adaugă unul după celălalt pe măsură ce se pătrunde în structură.

În cazul utilizării înregistrărilor în care anumite cimpuri sînt tot înregistrări, deci structurate pe mai multe nivele, accesul la o componentă încarcă foarte mult textul programului. Aceste cazuri, în PASCAL se rezolvă utilizînd instrucțiunea WITH. Sintaxa acestei instrucțiuni este dată de diagrama din fig.4.16.

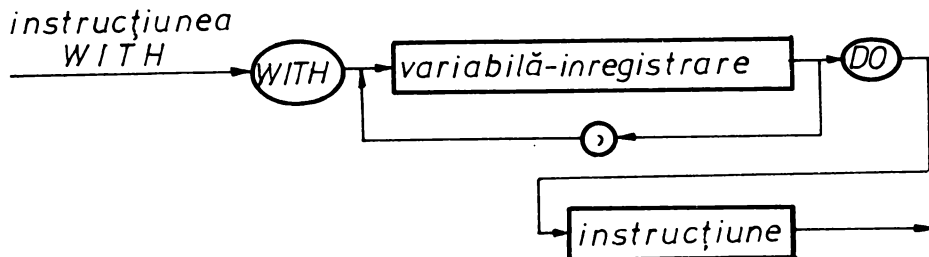


fig.4.16.

unde:

-instrucțiune este orice instrucțiune PASCAL.

Efectul instrucțiunii WITH se manifestă în faza de compilare, deci se referă la textul programului și nu la execuția lui. Instrucțiunea WITH delimitează în textul programului o zonă ce cuprinde textul lui *instrucțiune*. Peste această zonă variabilele din lista de variabile a instrucțiunii WITH acționează astfel: în toate referințele la componentele unei variabile *V* cuprinsă în listă, se poate omite prefixarea lor cu *V*, utilizînd chiar selectorii cimpurilor respective. Aceștia pot apare, deci, în cadrul instrucțiunii sau blocului din cadrul lui WITH ca identificatori de variabile.

Exemplu:

```
VAR ELEV: RECORD
    NUME: ARRAY[1..20] OF CHAR;
    DATA: ARRAY[1..3] OF INTEGER;
    NOTA: REAL
END;
```

Considerînd declarația de variabile anterioară, secvența:

```
READ(ELEV.NUME);
READ(ELEV.NOTA);
```

poate fi înlocuită, utilizînd instrucțiunea WITH, cu:

```
WITH ELEV DO
BEGIN
    READ(NUME);
    READ(NOTA)
END;
```

Observații:

1)-instrucțiunea

```
WITH A,B DO instrucțiune
```

este echivalentă cu grupul de instrucțiuni:

```
WITH A DO
```

```
WITH B DO instrucțiune
```

2)-un selector dintr-o înregistrare poate avea același nume cu o variabilă oarecare din program;

Exemplu:

```

VAR A: CHAR;
    INREG: RECORD
        A: INTEGER;
        B: REAL
    END;
.....
WITH INREG DO
    BEGIN
        A:=10;
        B:=5.2
    END;

```

Atribuirea A:=10 se referă la specificatorul de cîmp INREG.A deoarece este sub controlul lui WITH;

3)-variabila de tip înregistrare nu poate fi schimbată în timpul execuției instrucțiunii WITH; în cazul în care avem un tablou cu elemente de tip înregistrare

```

VAR CLASA: ARRAY[1..40] OF RECORD
    NUME: ARRAY[1..10] OF CHAR;
    NUMAR: INTEGER
END;

```

următoarea secvență este nepermisă:

```

WITH CLASA[I] DO
    BEGIN
        .....
        I:=I+1;
        :.....
    END;

```

deoarece mărirea variabilei I înseamnă schimbarea variabilei înregistrare.

În programul RECOR (P.IV.12) se declară o structură de tip înregistrare și se exemplifică modul de lucru și accesul la componentele variabilei. Tabloul ale cărui elemente sînt structurate ca înregistrare, reprezintă elevii unei clase. Sînt afișați acei elevi care au media peste 7. Informațiile despre fiecare elev constau în nume, adresă, număr matricol și medie.

```

B09F 10 PROGRAM RECOR;
B09F 20 TYPE DATE=RECORD
B09F 30     NUME,ADR:ARRAY [1..10] OF CHAR;
B09F 40     NR:INTEGER;
B09F 50     NOTA:REAL
B09F 60     END;
B09F 70 VAR ELEV:ARRAY[1..10] OF DATE;
B0A8 80     I,J:INTEGER;
B0A8 90
B0A8 100 BEGIN
B0B1 110 FOR I:=1 TO 4 DO
B0CB 120     BEGIN
B0CE 130         WRITELN('NUMELE: ');READLN;READ(ELEV[I].NUME);
B10A 140         WRITELN('ADR: ');READLN;READ(ELEV[I].ADR);
B147 150         FOR J:=1 TO 10 DO
B161 160             BEGIN
B164 170                 IF ELEV[I].NUME[J]=CHR(0) THEN ELEV[I].NUME[J]:=' ';
B1E2 180                 IF ELEV[I].ADR[J]=CHR(0) THEN ELEV[I].ADR[J]:=' ';
B268 190             END;
B268 200         WRITE('NR: ');READ(ELEV[I].NR);
B2A5 210         WRITE('NOTA: ');READ(ELEV[I].NOTA);
B2E6 220         WRITELN;
B2E9 230     END;
B2EC 240

```



```

B2EC 250 FOR I:=1 TO 4 DO
B306 260 WITH ELEV[I] DO
B330 270 BEGIN
B330 280 WRITE(I,NUME,ADR:11,NR:3,NOTA:6:2);
B387 290 WRITELN
B387 300 END;
B38D 310 WRITELN;
B390 320 WRITELN('ELEVII CU NOTA MAI MARE DE 7');
B3BA 330 FOR I:=1 TO 4 DO
B3D4 340 IF ELEV[I].NOTA>=7.00 THEN
B41B 350 BEGIN
B41B 360 WRITELN('ELEVUL:',I:2);
B43C 370 WRITELN;
B43F 380 WITH ELEV[I] DO
B466 390 BEGIN
B466 400 WRITELN('NUME:',NUME);
B481 410 WRITELN('ADR:',ADR);
B49F 420 WRITELN('NR:',NR);
B4C1 430 WRITELN('NOTA:',NOTA:5:2);
B4F0 440 WRITELN;
B4F3 450 END;
B4F3 460 END;
B4F6 470 END{#P}.

```

```

1 CORICI MARASTI 12 6.99
2 MANZ RESITA 59 9.33
3 SERBAN HEBE 38 7.01
4 SIMULESCU PLAVAT 98 6.13

```

```

ELEVII CU NOTA MAI MARE DE 7
ELEVUL: 2

```

```

NUME:MANZ
ADR:RESITA
NR:59
NOTA: 9.33

```

```

ELEVUL: 3

```

```

NUME:SERBAN
ADR:HEBE
NR:38
NOTA: 7.01

```

P. IV.12.

4.2.3. Tipul mulțime

Tipul *mulțime* este un tip structurat care se definește în raport cu un tip de bază care este un tip *ordinal*. Dându-se tipul de bază, tipul mulțime se definește ca fiind mulțimea tuturor *submulțimilor* tipului de bază, inclusiv mulțimea vidă.

Fiecare valoare a tipului mulțimii este o mulțime ale cărei componente sînt valori distincte ale tipului de bază. Numărul maxim de valori pentru tipul de bază diferă de la o implementare la alta. În cazul lui HP4TM, acest număr este 256.

Specificarea unui tip mulțime se face, conform diagramei din fig.4.17.

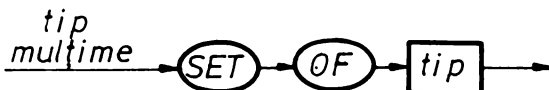


fig. 4.17.

unde:

-*tip* este tipul de bază.

Ca și în cazul celorlalte tipuri, acest tip nou definit poate rămâne anonim sau poate primi un nume într-o declarație de tip conform diagramei de sintaxă din fig.4.18.

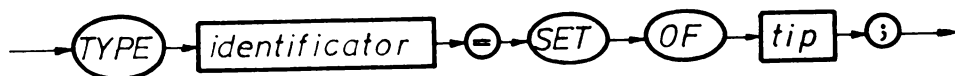


fig. 4. 18.

Mulțimile cu care lucrăm în program se pot alcătui din componentele lor folosind *constructori*. Un constructor are sintaxa din fig. 4.19.

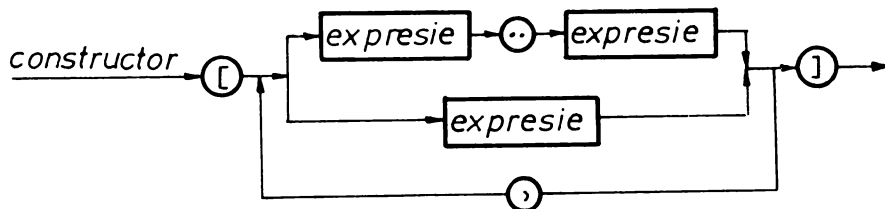


fig. 4. 19.

Cu alte cuvinte, un constructor determină o enumerare a elementelor unei mulțimi, cuprinsă între paranteze drepte.

Exemplu:

```

VAR A: SET OF 0..20;
    B: SET OF 'A'..'Z';
  
```

Variabilelor de tip mulțime declarate li se pot atribui mulțimi prin următorii constructori:

- 1) A:=[1,5..10];
B:=['C'..'J'];
- 2) A:=[1,5,10];
B:=['A', 'C', 'D'];
- 3) A:=[];
B:=['C'..'A'];

În exemplul 1) A este o mulțime formată din elementele 1,5,6,7,9,10 iar B o mulțime formată din caracterele C,D,E,F,G,H,I,J. În exemplul 3) mulțimile A și B desemnează mulțimea vidă deoarece atât constructorul [] cât și un constructor de tipul [a..b] cu b>a desemnează o mulțime fără elemente.

Tipul de bază al unei mulțimi desemnate de un constructor este considerat cel mai "sărac" tip căruia îi aparțin valorile elementelor cuprinse între parantezele drepte. De exemplu, pentru constructorul [1,2,3] tipul de bază este considerat 1..3, pentru [1..3] este tot 1..3 etc.

Cu variabilele de tip mulțime se pot efectua calcule. Operatorii folosiți sint dați în tabelul T.4.12.

Observație:

Operanzii trebuie să fie de același tip de bază ordinal, un tip de bază subdomeniu al celuiilalt sau ambele tipuri de bază subdomenii ale aceluiași tip.

Tabel T.4.12.

Operator	Tip operanzi	Tip rezultat	Semnificație
+	Mulțimi	Mulțime	Reuniune
*	Mulțimi	Mulțime	Intersecție
-	Mulțimi	Mulțime	Diferență
=	Mulțimi	Boolean	...este egală cu ...
<>	Mulțimi	Boolean	...este diferită de...
<=	Mulțimi	Boolean	...este inclusă în...
>=	Mulțimi	Boolean	...include pe...
IN	Ordinal, mulțime	Boolean	...aparține lui...

Programul REUNIUNE (P.IV.13) determină reuniunea a două mulțimi de numere întregi.

```

AF2C 10 PROGRAM REUNIUNE;
AF2C 20 (EXEMPLU PENTRU TIPUL MULTIME)
AF2C 30 TYPE NUMERE=0..20;
AF2C 40 VAR E,N,M,J,I:INTEGER;
AF35 50   A,B,C:SET OF NUMERE;
AF35 60 BEGIN
AF3E 70   A:=[];
AF3E 80   B:=[];
AF57 90   WRITELN('NUMARUL DE ELEMENTE DIN MULTIMEA A:');
AFA1 100  READ(J);
AFA7 110  WRITELN('NUMARUL DE ELEMENTE AL MULTIMII B:');
AFD7 120  READ(M);
AFDD 130  WRITELN('ELEMENTELE DIN A');
AFFB 140  FOR I:=1 TO J DO
B019 150  BEGIN
B01C 160  READ(E);
B022 170  A:=A+[E];
B061 180  END;
B065 190  WRITELN('ELEMENTELE DIN B');
B083 200  FOR I:=1 TO M DO
B0A1 210  BEGIN
B0A4 220  READ(E);
B0AA 230  B:=B+[E];
B0E9 240  END;
B0ED 250  C:=A + B;
B12F 260  FOR I:=0 TO 20 DO
B149 270  IF I IN C THEN WRITE(I:3);
B188 280  END ($P).

```

```

ELEMENTELE DIN A
( 5 4 7 8 3 )

```

```

ELEMENTELE DIN B
( 2 3 10 6 11 14 7 )

```

```

ELEMENTELE MULTIMII C
( 2 3 4 5 6 7 8 10 11 14 )

```

P. IV.13.

Programul SIRURI (P.IV.14) construiește, pornind de la un șir de numere întregi două șiruri: unul cu elementele pare, celălalt cu cele impare. În acest scop am construit o mulțime ale cărei elemente sînt cifre pare. În cadrul programului se localizează ultima cifră a unui element din șir și se cercetează apartenența sa la mulțimea cifrelor pare. În caz afirmativ elementul se depune în șirul cu numerele pare iar în caz contrar în cel cu numerele impare.

```

AEBC 10 PROGRAM SIRULI;
AEBC 20 TYPE TIF=SET OF 0..9;
AEBC 30   SIR=ARRAY[1..100] OF INTEGER;
AEBC 40 VAR NUMERE,SPAR,SIMPAR,SIRUL:SIR;
AECS 50   I,J,K,N: INTEGER;
AECS 60   PARE,IMPARE:TIF;
AECS 70 BEGIN
AECE 80   PARE:=[0,2,4,6,8];
AF14 90   IMPARE:=[1,3,5,7,9];
AF5A 100  WRITELN('NUMARUL DE ELEMENTE AL SIRULUI: ');
AF88 110  READ(N);
AF8E 120  WRITELN;WRITELN('ELEMENTELE SIRULUI:');WRITELN;
AF85 130  K:=0;J:=0;
AFC1 140  FOR I:=1 TO N DO
AFDF 150    BEGIN
AFE2 160      READ(SIRUL[I]);
B005 170      IF SIRUL[I] MOD 10 IN PARE THEN
B054 180        BEGIN
B054 190          J:=J+1;
B058 200          SPAR[J]:=SIRUL[I]
B091 210        END
B099 220      ELSE
B09C 230        BEGIN
B09C 240          K:=K+1;
B0A3 250          SIMPAR[K]:=SIRUL[I]
B0D9 260        END
B0E1 270      END;
B0E5 280    WRITELN;
B0E8 290    WRITELN('SIRUL CU ELEMENTELE PARE:',CHR(13));
B116 300    FOR I:=1 TO J DO
B134 310      WRITELN('SPAR(',I,')=',SPAR[I]);
B188 320    WRITELN;
B188 330    WRITELN('SIRUL CU ELEMENTELE IMPARE:',CHR(13));
B188 340    FOR I:=1 TO K DO
B1D9 350      WRITELN('SIMPAR(',I,')=',SIMPAR[I]);
B22F 360  END ($P).

```

ELEMENTELE SIRULUI:

5 6 3 9 1 2 4 6 4 3 8 2

SIRUL CU ELEMENTELE PARE:

```

SPAR(1 )=6
SPAR(2 )=2
SPAR(3 )=4
SPAR(4 )=6
SPAR(5 )=4
SPAR(6 )=8
SPAR(7 )=2

```

SIRUL CU ELEMENTELE IMPARE:

```

SIMPAR(1 )=5
SIMPAR(2 )=3
SIMPAR(3 )=9
SIMPAR(4 )=1
SIMPAR(5 )=3

```

P. IV. 14.

PROBLEME PROPUSE

1. Se dă un tablou unidimensional A cu elemente numere reale. Să se scrie un program PASCAL pentru interschimbarea elementelor egal depărtate între ele.

2. Fiind dat un tablou bidimensional cu elemente numere reale, să se scrie programul PASCAL care calculează media aritmetică a elementelor strict pozitive aflate deasupra diagonalei principale.

3. Fiind date tablourile unidimensionale A și B, fiecare având 20 de elemente caractere, să se scrie programul PASCAL care construiește tabloul C obținut din tablourile A și B astfel: primul element din A, al doilea element din B etc...

4. Fiind date tablourile unidimensionale de la problema precedentă să se construiască mulțimea C având ca elemente caracterele din tablourile A și B.

5. Să se scrie programul PASCAL care citește un articol cu următoarea structură: nume (20 caractere), adresa, nr. telefon, data nașterii: an, lună, zi. Să se afișeze persoanele care sînt născute în luna martie între anii 1950 și 1980.

6. Să se scrie programul PASCAL de înmulțire a două polinoame. Coeficienții celor două polinoame sînt dați sub formă de tablou unidimensional.

7. Dîndu-se un tablou bidimensional de numere reale, să se reordoneze liniile tabloului în ordine descrescătoare a elementelor de pe prima coloană.

8. Dîndu-se un tablou unidimensional de elemente numere reale, să se scrie programul PASCAL care elimină elementele nule din tablou.

9. Dîndu-se patru mulțimi de elemente caractere, să se scrie programul PASCAL care calculează și afișează următoarele rezultate:

```
A U B
A - B
A U (C - D)
```

10. Dîndu-se o matrice pătrată cu elemente caractere, să se scrie un program PASCAL care rotește această matrice cu 90 respectiv 180° spre stînga și spre dreapta și obține imaginea ei în oglindă.

V. SUBPROGRAME

Scrierea programului se face mult mai ușor dacă împărțim problema în subprobleme relativ independente, pentru fiecare din ele scriindu-se programe mult mai simple. De altfel, realizarea unui program de complexitate mare impune aproape ca o necesitate organizarea unor date și a acțiunilor la care acestea trebuie supuse, sub formă de subprograme. În PASCAL există un mecanism evoluat de declarare și utilizare a subprogramelor.

Subprogramele limbajului PASCAL sînt de două tipuri: *proceduri și funcții*. Diferența dintre ele constă în numărul valorilor calculate și returnate programului apelant: procedura transmite oricîte astfel de valori, pe cînd funcția transmite o singură valoare, acest lucru permițînd ca apelul ei să se facă chiar din expresia care conține valoarea calculată.

Atît procedurile, cît și funcțiile, pot fi de două tipuri: *standard* (predefinite) și *nestandard* (declarate în program). Procedurile și funcțiile nestandard trebuie în mod obligatoriu declarate înainte de a fi apelate. În cazul general, un program PASCAL este format dintr-un program principal și dintr-un număr oarecare de proceduri și funcții, apelabile din programul principal, sau unele din altele.

5.1. DOMENIUL DE VALABILITATE AL IDENTIFICATORILOR SI ETICHETELOR

Declararea procedurilor în cadrul altor proceduri, fiecare din acestea cu propriile declarații și definiții ridică o problemă deosebită: domeniul de valabilitate al identificatorilor și etichetelor.

Prin *domeniu de valabilitate* înțelegem zona din program în care este cunoscută declarația sau definiția unui identificator sau a unei etichete.

În PASCAL, la baza stabilirii domeniului de valabilitate stă conceptul de *bloc*. Conform diagramei din cap I, fig.1.3, blocul este corpul unui subprogram format din declarații, definiții și o instrucțiune compusă. Blocul are o proprietate importantă: toți identificatorii și etichetele definite în el sînt locale blocului și cunoscute doar în interiorul blocului. Se definește astfel *structura de bloc*. Într-un limbaj de programare cu structură de bloc, diferitele entități definite într-un bloc se "nasc" la intrarea în execuția (activarea) blocului, apoi "dispar" la ieșirea din execuția (dezactivarea) blocului respectiv. În acest fel se

obține o folosire mai bună a spațiului de memorare a datelor. De asemenea, există în acest fel posibilitatea activării recursive a unui bloc.

Atunci când fluxul de control intră într-un bloc, toate entitățile declarate în blocul respectiv sînt alocate pe stiva de execuție; la părăsirea unui bloc zona de pe stivă alocată blocului este eliberată astfel încît valorile aflate acolo nu mai pot fi referite.

O consecință a structurii de bloc este că procedurile și funcțiile pot fi recursive, adică se pot apela pe ele însele, fie direct, fie indirect printr-un lanț de apeluri. Fiecare invocare a unei proceduri determină alocarea de spațiu pe stivă pentru entitățile sale locale care sînt astfel distincte de entitățile corespunzătoare alocate în timpul altor apeluri ale aceleiași proceduri. Acest spațiu este eliberat la terminarea invocării respective.

In concluzie:

Domeniul de valabilitate al unui identificator sau etichete este tot textul blocului în care aceștia au fost definiți, inclusiv textele blocurilor subprogramelor declarate în bloc, cu excepția celor care redefinesc identificatorul sau eticheta. Într-un punct al programului situat în blocul B sînt cunoscuți toți identificatorii definiți în B și în blocurile care înconjoară textual pe B. Pentru oricare dintre acești identificatori, definiția valabilă în punctul considerat este cea aflată în blocul cel mai apropiat de B. Același lucru este valabil și pentru etichete.

5.2. DEZVOLTAREA PROGRAMELOR PASCAL

În PASCAL există două posibilități de dezvoltare a programelor.

A) În programarea ascendentă subprogramele se declară în secțiunea de declarații a programului unul după altul, respectînd o ordine anume impusă de eventualele apeluri dintre acestea. Restricția se referă la regula generală privind referirea identificatorilor numai după declararea lor. În acest context este clar că apelul unei proceduri p într-o linie program este acceptat de compilator numai dacă procedura p a fost declarată într-o linie anterioară.

Un program scris în stilul ascendent va avea structura din fig. 5.1.

Observații:

1) - prin antet procedură înțelegem linia program în care se declară numele subprogramului; sintaxa ei se va prezenta detaliat în 5.3.1.

2) - într-un astfel de program, blocurile sînt independente;

3) - apelurile posibile într-un program cu structura de mai sus sînt:

* din programul principal se pot apela subprogramele notate procedură- i , pentru $i=1, n$ în orice ordine;

* din procedură- i se pot apela subprogramele procedură- k , pentru $k=1, i-1$ în orice ordine.

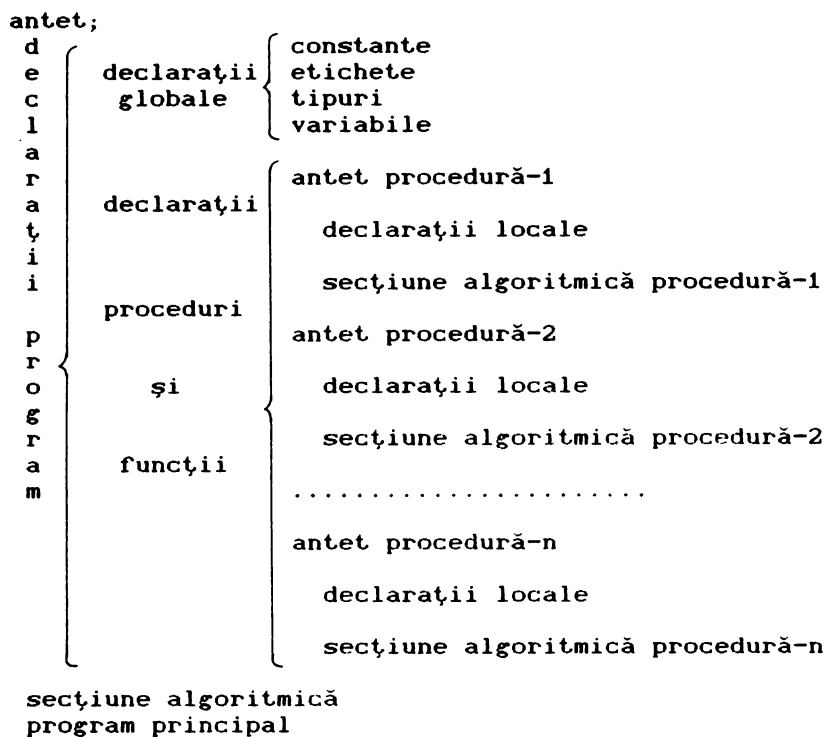


fig. 5.1.

Simplificînd schema din fig.5.1., structura programului este reprezentată în fig.5.2.

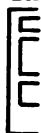


fig. 5.2.

B) În programarea *descendentă* subprogramele se declară *imbricate* unul în celălalt. Avînd în vedere că dintr-un bloc sînt vizibile (deci apelabile) numai acele subprograme care sînt declarate în secțiunea de declarații' atașate blocului respectiv, nu orice subprogram va fi apelabil dintr-un anume bloc, chiar dacă acesta este declarat într-o linie anterioară.

Un program scris în stilul *descendent* va avea structura din fig.5.3.

Observații:

1)- apelurile posibile într-un program cu structura din fig.5.3. sînt:

* din programul principal se poate apela subprogramul *procedură-1*;

* din *procedură-i* se poate apela *procedură-i+1*;

2)- comparînd cele două structuri, observăm că vom prefera dezvoltarea programului *ascendent* în cazul în care dorim să apelăm din programul principal orice subprogram declarat;

3)- în cazul programării *descendente*, avînd în vedere regulile după care se stabilește domeniul de valabilitate al

identificatorilor, in general, se reduce numărul parametrilor și al declarațiilor locale;

4)- evident, în practică se pot combina cele două stiluri (*ascendent* și *descendent*).

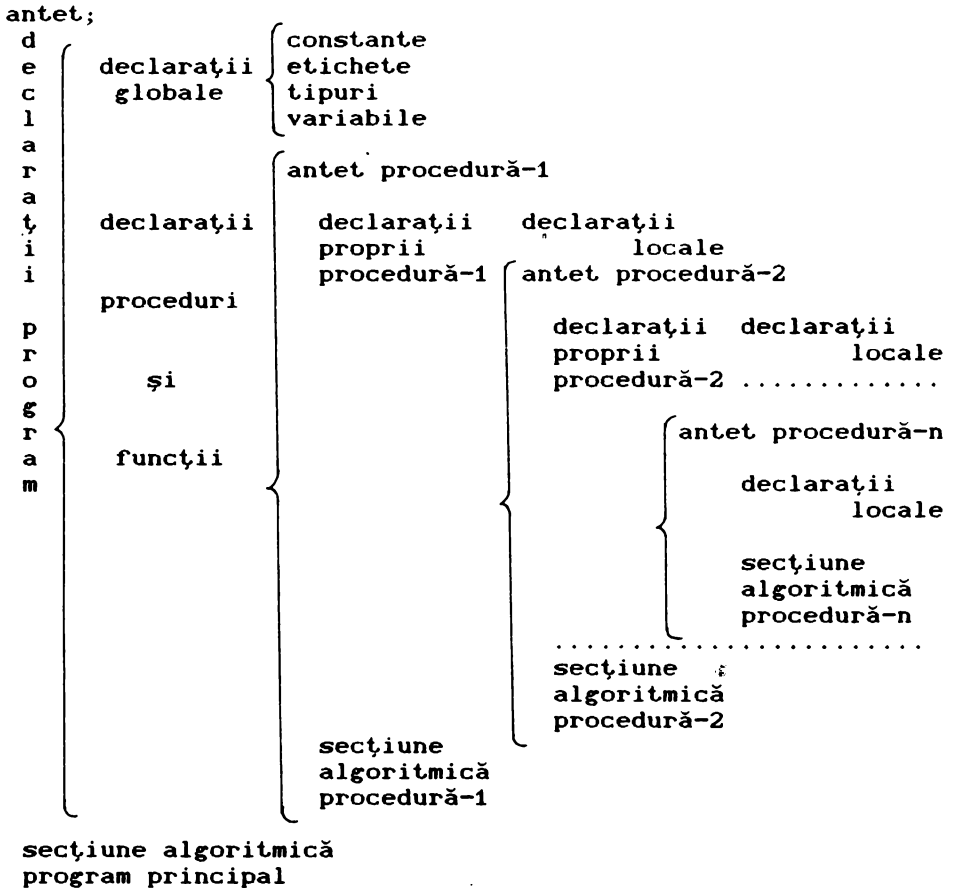


fig.5.3.

fig.5.4. Schematic, structura programului este reprezentată în

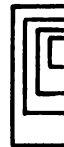


fig.5.4.

5.3. PROCEDURI

5.3.1. Declararea procedurilor.

Parametri formali, parametri efectivi

Declarațiile de proceduri reprezintă părți distincte ale programului PASCAL. Aceste părți pot fi identificate în program conform declarației de sintaxă din fig.5.5.

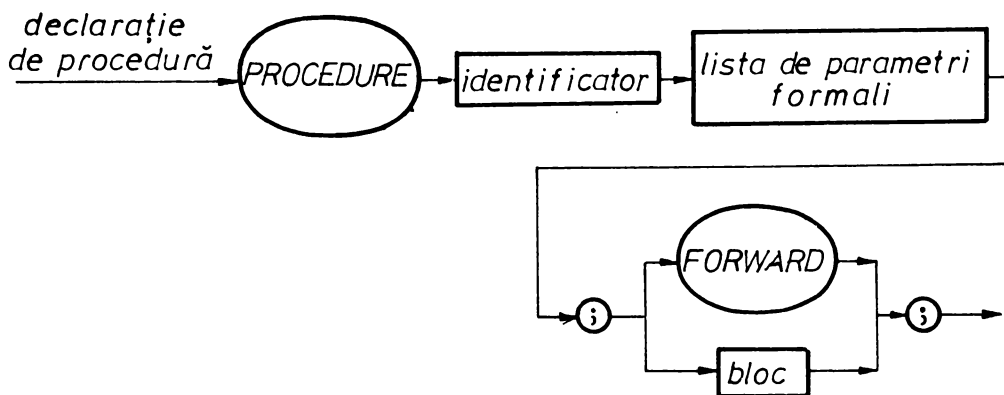


fig.5.5.

Lista de *parametri formali* este formată din mai multe subliste conform diagramei din fig.5.6.

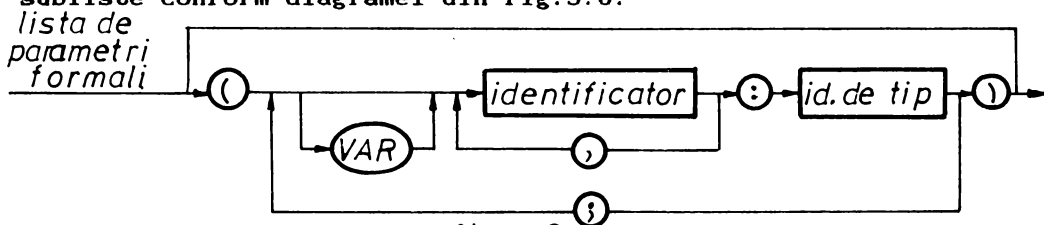


fig.5.6.

Se observă că sublistele se separă între ele prin ';', iar parametri în cadrul sublistei se despart prin ',' In PASCAL există următoarele reguli de utilizare a parametrilor formali:

- un parametru formal apare ca un identificator; acestuia i se asociază un tip, specificat prin *id. de tip*;
- parametri formali declarați într-o procedură sînt cunoscuți în întreg textul procedurii și necunoscuți în afară;
- în cadrul procedurii, parametri formali se manifestă ca niște variabile de tipul respectiv;
- dacă lista parametrilor formali este vidă, atunci sau nu există schimb de informații cu restul programului sau schimbul se execută prin intermediul variabilelor globale.

In limbajul PASCAL se pot defini trei mari categorii de parametri formali:

a) *-parametri transmiși prin valoare*: sublista respectivă nu este precedată de nici un cuvînt cheie (VAR, FUNCTION, PROCEDURE); acești parametri pot fi modificați în corpul procedurii dar valorile noi nu se transmit către blocul apelant, ei sînt doar *parametri de intrare*;

b) *-parametri transmiși prin referință*: sublista respectivă este precedată de cuvîntul cheie VAR, iar parametri actuali sînt *variabile*; acești parametri pot fi modificați în corpul procedurii, ei sînt *parametri de ieșire*, nu sînt transmiși prin valoare ci se transmite adresa lor, deci în blocul apelant valorile parametrilor actuali vor fi modificate;

c) *-parametri funcții sau proceduri*: sublista respectivă este precedată de cuvintele cheie FUNCTION sau PROCEDURE; avînd în vedere că în implementarea aleasă nu pot exista parametri funcții sau proceduri, nu vom trata acest caz.

Activarea unei proceduri se realizează prin *instrucțiunea de apel* (*instrucțiune procedurală*) care constă în identificatorul procedurii, urmat eventual de o listă de *parametri actuali* (*efectivi*).

Referitor la utilizarea listelor de parametri (formali și efectivi), în PASCAL există următoarele reguli:

a)-între lista parametrilor formali și lista parametrilor efectivi există o corespondență unu-la-unu determinată de ordinea parametrilor în listă;

b)-în cazul transmiterii prin valoare, parametrii efectivi sînt expresii; tipul unei astfel de expresii trebuie să fie compatibil cu tipul parametrului formal corespunzător; din punct de vedere al compatibilității trebuie respectate regulile stabilite la atribuire între expresie și variabilă;

c)-în cazul transmiterii prin referință, parametrul efectiv asociat cu un parametru-variabilă trebuie să fie o variabilă de același tip cu parametrul formal; orice operație efectuată asupra unui parametru-variabilă va fi realizată în fond asupra parametrului efectiv asociat.

5.3.2.Proceduri standard de intrare/ieșire

În PASCAL există o serie de proceduri standard, care nu trebuie declarate, ele asigurînd o serie de facilități necesare în orice program.

Procedura WRITE

-este folosită pentru a transmite date către monitor sau către imprimantă.

Forma generală a apelului procedurii este:

```
WRITE(p1,p2,...,pn);
```

și este echivalentă cu:

```
BEGIN
  WRITE(p1);
  WRITE(p2);
  .....
  WRITE(pn)
END;
```

Parametrii p_1, p_2, \dots, p_n pot avea una din următoarele forme:

```
e
e:m
e:m:n
e:m:H
```

unde e este expresia de tipărit, m , n sînt expresii întregi, iar H o constantă literală indicînd tipărirea în hexa.

Avînd în vedere forma pe care o pot avea parametrii, precum și tipurile pe care le pot avea expresiile, semnalăm 5 cazuri:

1)- e este de tip INTEGER și se folosește una din formele e sau $e:m$;

În acest caz valoarea expresiei este convertită într-un șir de caractere, urmat de un spațiu. Lungimea șirului

poate fi mărită (cu mai multe spații în față) prin folosirea lui m , care specifică numărul total de caractere transmise spre ieșire. Dacă m este mai mic decît numărul de cifre al expresiei e , atunci m nu se ia în considerare (se consideră absent) și se transmite spre ieșire valoarea expresiei e , fără a se tipări și spațiul la sfîrșit.

2)- e este de tip **INTEGER** și se folosește forma $e:m:H$;

În acest caz, înainte de transmiterea valorii lui e se realizează o conversie în hexazecimal:

-dacă $m=1$ sau $m=2$ atunci este transmisă valoarea ultimelor m cifre hexa ale expresiei pe m poziții;

-dacă $m=3$ sau $m=4$ este transmisă valoarea completă a lui e , în hexazecimal, pe 4 poziții;

-dacă $m>4$, vor fi inserate spații în fața valorii complete a lui e , în hexazecimal, pînă la completarea celor m caractere. Dacă este cazul, sînt inserate zerouri la stînga, iar numerele negative sînt complementate. Ilustrarea celor spuse mai sus se poate urmări în programele **PV1** și **TEST (P.V.1)**.

```
AC80 10 PROGRAM PV1:
AC80 20 VAR M,N:INTEGER;
AC87 30 E:INTEGER;
AC87 40 BEGIN
AC92 50 E:=1025;
AC98 60 FOR M:= 1 TO 10 DO
AC92 70 BEGIN
AC95 80 WRITE('m=',M:2,' E=',E:M:H);
ACE7 90 WRITE(CHR(13))
ACEE 100 END;
ACF1 110 END {$P}.
```

```
m= 1 E=1
m= 2 E=01
m= 3 E=0401
m= 4 E=0401
m= 5 E= 0401
m= 6 E= 0401
m= 7 E= 0401
m= 8 E= 0401
m= 9 E= 0401
m=10 E= 0401
```

```
AC82 10 PROGRAM TEST:
AC82 20 VAR M,N:INTEGER;
AC88 30 E:INTEGER;
AC88 40 BEGIN
AC94 50 E:=-1025;
AC9D 60 FOR M:= 1 TO 10 DO
AC87 70 BEGIN
AC9A 80 WRITE('m=',M:2,' E=',E:M:H);
ACEC 90 WRITE(CHR(13))
ACF3 100 END;
ACF6 110 END {$P}.
```

```
m= 1 E=F
m= 2 E=FF
m= 3 E=FBFF
m= 4 E=FBFF
m= 5 E= FBFF
m= 6 E= FBFF
m= 7 E= FBFF
m= 8 E= FBFF
m= 9 E= FBFF
m=10 E= FBFF
```

3)-e este de tip REAL; se poate utiliza una din formele e, e:m, e:m:n;

Valoarea lui e este convertită într-un șir de caractere care reprezintă un număr real. Formatul reprezentării este determinat de n:

-dacă n nu este prezent, atunci numărul este transmis în notație cu exponent. Dacă numărul este negativ, se transmite un semn - (minus) în fața mantisei; în caz contrar este transmis un spațiu; numărul este transmis întotdeauna cu cel puțin una și cel mult 5 zecimale, iar exponentul este transmis întotdeauna cu semn (plus sau minus); aceasta înseamnă că lungimea maximă a acestei reprezentări este de 8 caractere;

-dacă m este mai mic decât 8 (ex. m=7), atunci va fi transmisă întotdeauna lungimea completă de 12 caractere;

-dacă m>=8, vor fi transmise una sau mai multe zecimale (pină la 5 zecimale, cind m=12);

-dacă m>12, în fața numărului vor fi inserate spații.

Ilustrarea celor prezentate mai sus se poate urmări în programul TEST (P.V.2), pe următoarele exemple:

```
e = -1.025E10
e = 100000/3
e = 1/333
```

```
AC81 10 PROGRAM TEST;
AC81 20 VAR M,N:INTEGER;
AC8A 30 E:REAL;
AC8A 40 BEGIN
AC93 50 E:=-1.025E10;
ACA8 60 FOR M:= 7 TO 15 DO
ACC2 70 BEGIN
ACC5 80 WRITE('m=',M:2,' E=',E:M);
ACFE 90 WRITE(CHR(13))
AD05 100 END;
AD08 110 END {$P}.
```

```
m= 7 E=-1.02500E+10
m= 8 E=-1.0E+10
m= 9 E=-1.02E+10
m=10 E=-1.025E+10
m=11 E=-1.0250E+10
m=12 E=-1.02500E+10
m=13 E= -1.02500E+10
m=14 E= -1.02500E+10
m=15 E= -1.02500E+10
```

```
m= 7 E= 3.33333E+03
m= 8 E= 3.3E+03
m= 9 E= 3.33E+03
m=10 E= 3.333E+03
m=11 E= 3.3333E+03
m=12 E= 3.33333E+03
m=13 E= 3.33333E+03
m=14 E= 3.33333E+03
m=15 E= 3.33333E+03
```

```
m= 7 E= 3.00300E-03
m= 8 E= 3.0E-03
m= 9 E= 3.00E-03
m=10 E= 3.003E-03
m=11 E= 3.0030E-03
m=12 E= 3.00300E-03
m=13 E= 3.00300E-03
m=14 E= 3.00300E-03
m=15 E= 3.00300E-03
```

Dacă se folosește forma $e:m:n$, atunci expresia e va fi scrisă în reprezentare cu punct fix, cu n zecimale. Când lungimea m a cimpului este suficient de mare, vor fi transmise în față spații. Dacă $n=0$, e va fi transmis ca întreg. Dacă e este prea mare pentru a fi scris în lungimea m specificată de cimp, atunci el va fi transmis în formatul cu exponent, pe un cimp de lungime m , ca în cazul anterior. Marca zecimală '.' se ia în considerare la stabilirea numărului de caractere ce trebuie transmise. Exemplele din programul TEST (P.V.3) ilustrează cele prezentate mai sus, pentru:

```
e = 23.455
e = -23.455
```

```
ACB6 10 PROGRAM TEST;
ACB6 20 VAR M,N:INTEGER;
ACBF 30 E:REAL;
ACBF 40 BEGIN
ACC8 50 E:=23.455;
ACD5 60 FOR M:= 4 TO 8 DO
ACEF 70 BEGIN
ACF2 80 FOR N:=0 TO 4 DO
AD0C 90 BEGIN
AD0F 100 WRITE('m=',M:1,' n=',N:1,' E=',E:M:N);
AD64 110 WRITE(CHR(13))
AD6B 120 END;
AD6E 130 WRITE(CHR(13))
AD75 140 END;
AD78 150 END (*P).
```

```
m=4 n=0 E= 23
m=4 n=1 E=23.5
m=4 n=2 E= 2.34550E+01
m=4 n=3 E= 2.34550E+01
m=4 n=4 E= 2.34550E+01
```

```
m=5 n=0 E= 23
m=5 n=1 E= 23.5
m=5 n=2 E=23.45
m=5 n=3 E= 2.34550E+01
m=5 n=4 E= 2.34550E+01
```

```
m=6 n=0 E= 23
m=6 n=1 E= 23.5
m=6 n=2 E= 23.45
m=6 n=3 E=23.455
m=6 n=4 E= 2.34550E+01
```

```
m=7 n=0 E= 23
m=7 n=1 E= 23.5
m=7 n=2 E= 23.45
m=7 n=3 E= 23.455
m=7 n=4 E=23.4550
```

```
m=8 n=0 E= 23
m=8 n=1 E= 23.5
m=8 n=2 E= 23.45
m=8 n=3 E= 23.455
m=8 n=4 E= 23.4550
```

```
m=4 n=0 E= -23
m=4 n=1 E=-2.34550E+01
m=4 n=2 E=-2.34550E+01
m=4 n=3 E=-2.34550E+01
m=4 n=4 E=-2.34550E+01
```

```
m=5 n=0 E= -23
m=5 n=1 E=-23.5
m=5 n=2 E=-2.34550E+01
m=5 n=3 E=-2.34550E+01
m=5 n=4 E=-2.34550E+01
```

```

m=6 n=0 E= -23
m=6 n=1 E= -23,5
m=6 n=2 E=-23,45
m=6 n=3 E=-2,34550E+01
m=6 n=4 E=-2,34550E+01

m=7 n=0 E= -23
m=7 n=1 E= -23,5
m=7 n=2 E= -23,45
m=7 n=3 E=-23,455
m=7 n=4 E=-2,34550E+01

m=8 n=0 E= -23
m=8 n=1 E= -23,5
m=8 n=2 E= -23,45
m=8 n=3 E= -23,455
m=8 n=4 E=-23,4550

```

P. V. 3.

4)-e este de tip CHAR sau de tip șir de caractere;

Se poate folosi una din formele e sau e:m. Caracterul sau șirul de caractere va fi transmis pe o lungime minimă de cimp, egală cu 1 (pentru caractere), sau egală cu lungimea șirului (în cazul șirurilor). Dacă m este suficient de mare vor fi inserate spații la stînga.

Cînd expresia care urmează să fie scrisă este simplă, de tip caracter, atunci WRITE(e) transmite către monitor sau imprimantă, după caz, valoarea pe 8 biți a expresiei e. În PASCAL HP4TM există cîteva coduri care într-o procedură WRITE au un rol aparte, fiind considerate coduri de control. Aceste coduri sînt:

```

CHR(8) -mută cursorul înapoi cu un pas pe ecran, cu
ștergere (BACK SPACE);
CHR(12)-șterge ecranul sau execută salt la pagină nouă,
dacă este transmis la imprimantă;
CHR(13)-execută <CR> și <LF> (Carriage Return și Line
Feed) adică termină linia și execută salt la linie
nouă;
CHR(16)-comută ieșirea de la monitor la imprimantă și
invers.

```

Observații:

1)-înainte de terminarea programului trebuie să ne asigurăm că ieșirea este pe monitor, pentru a putea continua lucrul;

2)-cînd se dorește lucrul alternativ cu monitorul și imprimanta, citirea datelor se face de la tastatură prin vizualizare pe monitor iar scrierea se face la imprimantă, deci trebuie să avem grijă ca în momentul citirii datelor canalul de ieșire spre monitor să fie deschis, iar în momentul scrierii la imprimantă canalul de ieșire spre imprimantă să fie deschis;

3)-la pornire, HP4TM are conectată ieșirea spre monitor.

```

AD29 10 PROGRAM TEST;
AD29 20 VAR M:INTEGER;
AD32 30 E:ARRAY[1..8] OF CHAR;
AD32 40 CAR:CHAR;
AD32 50 BEGIN
AD38 60 CAR:= '*';
AD40 70 E:= 'EX EMPLU';
AD58 90 WRITE('CAR=', CAR, CHR(13), 'E=', E, CHR(13), CHR(13));
AD97 90 FOR M:= 0 TO 4 DO
ADB1 100 BEGIN
ADB4 110 WRITE('m=', M:1, ' CAR=', CAR:M);
ADED 120 WRITE(CHR(13))
ADF4 130 END;
ADF7 140 FOR M:= 6 TO 12 DO
AE11 150 BEGIN
AE14 160 WRITE('m=', M:2, ' E=', E:M);

```

```

AE4D 170 WRITE(CHR(13))
AE54 180 END;
AE57 190 END {$P}.

```

```

CAR=*
E=EX EMPLU

```

```

m=0 CAR=*
m=1 CAR=*
m=2 CAR= *
m=3 CAR= *
m=4 CAR= *
m= 6 E=EX EMPLU
m= 7 E=EX EMPLU
m= 8 E=EX EMPLU
m= 9 E= EX EMPLU
m=10 E= EX EMPLU
m=11 E= EX EMPLU
m=12 E= EX EMPLU

```

P. V. 4.

5)-e este de tip **BOOLEAN**; pot fi utilizate formele e sau e:m;

În ambele cazuri se transmite 'TRUE' sau 'FALSE', în funcție de valoarea lui e, folosind o lungime minimă de cimp de 4 sau 5 caractere. Programele **NPRIM** (P.V.5a și P.V.5b) stabilesc dacă un număr natural introdus de la tastatură este sau nu prim, răspunsul la întrebare -'TRUE' sau 'FALSE'- fiind atribuit variabilei booleene PRIM.

Se observă (în P.V.5a) modul de scriere a celor două valori 'TRUE' și 'FALSE' pe 4/5 caractere și faptul că listarea este "urită". Prin simpla scriere cu format (liniile 170 și 180 în P.V.5b) listarea se face ordonat, tabulat.

```

AD10 10 PROGRAM NPRIM:
AD10 20 VAR I,N:INTEGER;
AD19 30 PRIM:BOOLEAN;
AD19 40 BEGIN
AD22 50 REPEAT
AD22 60 WRITE ('N=');
AD32 70 READ(N);
AD38 80 I:=2;
AD3E 90 WHILE (I<N) AND ((N MOD I) <>0) DO I:=I+1;
AD78 100 PRIM:=I=N;
AD89 110 CASE PRIM OF
AD8C 120 TRUE:Writeln(PRIM, ' NUMARUL ',N:6, ' ESTE PRIM');
ADD2 130 FALSE:Writeln(PRIM, ' NUMARUL ',N:6, ' NU ESTE PRIM')
AE15 140 END;
AE18 150 UNTIL FALSE
AE1A 160 END {$P}.

```

```

TRUE NUMARUL      67 ESTE PRIM
TRUE NUMARUL     4567 ESTE PRIM
FALSE NUMARUL    7653 NU ESTE PRIM
TRUE NUMARUL      2 ESTE PRIM
TRUE NUMARUL      3 ESTE PRIM
FALSE NUMARUL     4 NU ESTE PRIM
FALSE NUMARUL   32767 NU ESTE PRIM
TRUE NUMARUL    15737 ESTE PRIM
FALSE NUMARUL    987 NU ESTE PRIM

```

P. V. 5a.


```

AD14 10 PROGRAM NPRIM;
AD14 20 VAR I,N:INTEGER;
AD1D 30 PRIM:BOOLEAN;
AD1D 40 BEGIN
AD26 50 REPEAT
AD26 60 WRITE ('N=');
AD36 70 READ(N);
AD3C 80 I:=2;
AD42 90 WHILE (I<N) AND ((N MOD I) <>0) DO I:=I+1;
AD7C 100 PRIM:=I=N;
AD8D 110 CASE PRIM OF
AD90 120 TRUE:WRITELN(PRIM:6,' NUMARUL ',N:6,' ESTE PRIM');
ADE3 130 FALSE:WRITELN(PRIM:6,' NUMARUL ',N:6,' NU ESTE PRIM')
AE33 140 END;
AE36 150 UNTIL FALSE
AE38 160 END {#P}.

```

```

TRUE NUMARUL      67 ESTE PRIM
TRUE NUMARUL     4567 ESTE PRIM
FALSE NUMARUL    7653 NU ESTE PRIM
TRUE NUMARUL       2 ESTE PRIM
TRUE NUMARUL       3 ESTE PRIM
FALSE NUMARUL     4 NU ESTE PRIM
FALSE NUMARUL   32767 NU ESTE PRIM
TRUE NUMARUL    15737 ESTE PRIM
FALSE NUMARUL   987 NU ESTE PRIM

```

P. V. 5b.

Procedura WRITELN

-asemenea procedurii WRITE, este folosită pentru a transmite date către monitor sau către imprimantă.

Forma generală a apelului procedurii este:

```
WRITELN (p1,p2,...,pn);
```

Avind același rol ca și procedura WRITE, sint valabile toate cazurile tratate anterior; singura deosebire constă în faptul că după terminarea afișării valorilor tuturor parametrilor se execută automat salt la linie nouă, deci:

```
WRITELN (p1,p2,...,pn);
```

este echivalent cu

```
WRITE (p1,p2,...,pn,CHR(13));
```

precum și cu

```

BEGIN
WRITE (p1);
WRITE (p2);
.....
WRITE (pn);
WRITE(CHR(13))
END;

```

WRITELN se poate apela și fără parametri (WRITELN;), caz în care are ca efect doar salt la linie nouă.

Exemple se pot urmări în programul WRITELNEX (P. V. 6).

```

AC5E 10 PROGRAM WRITELNEX:
AC5E 20 VAR I:INTEGER;
AC67 30 BEGIN
AC70 40 WRITELN:
AC73 50 FOR I:=1 TO 10 DO
AC8D 60 WRITELN(I:3,' ',I*I:4,' ',I*I*I:5)
ACD6 70 END {$P}.

```

1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

P.V.6.

Procedura PAGE

Forma generală a apelului procedurii este:

```
PAGE;
```

Această procedură este echivalentă cu WRITE (CHR(12)) și are ca efect ștergerea ecranului dacă ieșirea este direcționată spre monitor, sau saltul la pagină nouă dacă ieșirea este direcționată spre imprimantă.

Observație:

Pentru scrierea din prima linie a ecranului se procedează în felul următor:

```
PAGE;
WRITE (CHR(22),CHR(0),CHR(0));
```

Remarcăm că deoarece CHR(22) reprezintă codul instrucțiunii BASIC "PRINT AT", se așteaptă încă doi parametri: linia și coloana (în cazul nostru 0, 0 deci CHR(0), CHR(0)).

Utilizarea procedurii PAGE este ilustrată în programul WRITELNEX (P.V.7).

```

AC88 10 PROGRAM WRITELNEX:
AC88 20 VAR I:INTEGER;
AC91 30 BEGIN
AC9A 40 PAGE;
AC9F 50 WRITE(CHR(22),CHR(0),CHR(0));
ACB4 60 WRITELN;
ACB7 70 FOR I:=1 TO 10 DO
ACD1 80 WRITELN(I:3,' ',I*I:4,' ',I*I*I:5)
AD1A 90 END {$P}.

```

P.V.7.

Procedura READ

-este folosită pentru a transmite date de la tastatură sau din fișierele utilizatorului (în cazul altor implementări decât HP4TM).

Forma generală a apelului procedurii este:

```
READ (v1,v2,...,vn);
```

Introducerea datelor se realizează prin intermediul unei zone tampon, deschisă în timpul executării programului,

zonă care inițial este goală, cu excepția (în cazul implementării HP4TM) a unui caracter sfârșit de linie (EOL).

Se poate considera că accesul la această zonă are loc printr-o fereastră de text care permite vizualizarea în zonă a unui singur caracter odată. Dacă această fereastră de text este poziționată în dreptul unui caracter EOL și operația de citire este încheiată, va fi citită în zona tampon o nouă linie de text de la tastatură. Când se citește această linie, sint recunoscute diferite coduri de control specifice implementării HP4TM. Aceste coduri sint:

```

RETURN (ENTER)      -sfârșit de linie;
CC (CAPS SHIFT + 1) -revenire in editor (EDIT);
CH (CAPS SHIFT + 0) -șterge ultimul caracter introdus
                    (DELETE);
CI (CAPS SHIFT + 8) -deplasarea la următoarea poziție TAB;
CP (CAPS SHIFT + 3) -schimbă ieșirea spre imprimantă
                    (dacă există), iar dacă ieșirea este
                    la imprimantă, se revine la monitor;
CX (CAPS SHIFT + 5) -șterge întreaga linie.

```

Forma generală a apelului procedurii READ este echivalentă cu:

```

BEGIN
  READ(v1);
  READ(v2);
  .....
  READ(vn)
END;

```

Variabilele v_1, v_2, \dots, v_n pot fi de tip caracter, șir de caractere, întreg sau real.

Ținând cont de tipul variabilei V , READ(V) are efecte diferite; în acest sens trebuie considerate 4 cazuri:

1)- v este de tip caracter;

În acest caz, READ(V) citește un caracter din zona tampon de intrare și îl atribuie lui V . Dacă fereastra din tampon este poziționată pe un caracter sfârșit-de-linie (CHR(13)), atunci funcția standard EOLN va primi valoarea TRUE și va fi citită o nouă linie de la tastatură. Când operația de citire este ulterior executată, fereastra de text va fi poziționată la începutul unei linii noi.

Este important de reținut că EOLN are valoarea TRUE la pornirea programului. Aceasta înseamnă că dacă la primul READ se citește o variabilă de tip caracter, se returnează o valoare CHR(13), urmată de citirea unei noi linii și deci următoarea citire de tip caracter va returna primul caracter din noua linie.

În programul TESTREAD (P.V.8), s-au introdus pe rînd caracterele 'TEST READ CU INLOCUIRE CHR(13)' și au fost scrise la imprimantă (s-a utilizat constanta CHR(16) pentru scrierea la imprimantă și revenirea imediată la monitor pentru continuarea citirii). Se observă că primul caracter transmis spre imprimantă a fost '#', deci primul caracter citit a fost CHR(13).

Programul conține un ciclu infinit din cauza liniei 100 în care condiția niciodată nu va fi îndeplinită (FALSE nu poate să devină TRUE). Pe de altă parte, citind astfel, nu avem obligația să precizăm numărul caracterelor, nici să semnalăm (cu CHR(13)) sau un caracter anume) sfârșitul citirii. Execuția se intrerupe cu codul de control CC.

```
AC79 10 PROGRAM TESTREAD;
AC79 20 CONST P=CHR(16);
AC79 30 VAR CAR:CHAR;
AC82 40 BEGIN
AC88 50 REPEAT
AC88 60 READ(CAR);
AC94 70 IF CAR=CHR(13) THEN
ACA7 80 CAR:='#';
ACAC 90 WRITE(P,CAR,P)
ACBC 100 UNTIL FALSE
ACBE 110 END {$P}.
```

#TEST READ CU INLOCUIRE CHR(13)#

P. V. 8.

2)-v este de tip șir de caractere;

Un șir de caractere poate fi citit cu READ și, în acest caz, vor fi citite o serie de caractere, ori până se atinge numărul de caractere rezultat din definiția șirului, ori până când EOLN=TRUE. Dacă șirul nu este completat prin citire (adică dacă sfârșit-de-linie este atins înainte ca întregul șir să fie atribuit), atunci până la sfârșitul șirului se completează cu caractere nule (CHR(0)); astfel, programatorul are posibilitatea să evalueze lungimea șirului care a fost citit. Observațiile făcute la cazul 1) sint valabile.

În programul TESTREAD (P.V.9) se citește cu READ(CHR13) caracterul inițial sfârșit-de-linie (CHR(13)), apoi cu READ(A) șirul A de tip caracter. La tastarea lui ENTER (CHR(13)), se intrerupe introducerea caracterelor, se tipărește șirul A (cu spații la sfârșit - deoarece CHR(0) nu este tipăribil - pe monitor apare!) apoi parcurgându-se șirul se înlocuiește CHR(0) cu '#'. La o nouă imprimare a lui A, în locul spațiilor vor apărea 7 caractere '#'. Se remarcă folosirea constantelor P și L pentru comutarea ieșirii spre imprimantă (și invers) respectiv pentru saltul la linie nouă.

```
ACCC 10 PROGRAM TESTREAD;
ACCC 20 CONST P=CHR(16);
ACCC 30 L=CHR(13);
ACCC 40 VAR A:ARRAY[1..15] OF CHAR;
ACD5 50 I:INTEGER;
ACD5 60 CHR13:CHAR;
ACD5 70 BEGIN
ACDE 80 READ(CHR13);
ACE4 90 READ(A);
ACEC 100 WRITE(P,A,L);
ACFE 110 FOR I:=1 TO 15 DO
AD18 120 IF A[I]=CHR(0) THEN
AD44 130 A[I]:='#';
AD64 140 WRITE(A,L,P)
AD76 150 END {$P}.
```

TEST SIR
TEST SIR#####

P. V. 9.

Observație:

Citirea șirurilor de caractere diferă de la o implementare la alta.

3)-v este de tip întreg;

În acest caz, READ(V) citește un șir de caractere care reprezintă un întreg, așa cum a fost acesta definit. Toate spațiile și caracterele sfârșit-de-linie din stînga numărului sînt ignorate.

Dacă numărul citit nu aparține intervalului [-MAXINT,MAXINT] va fi emis mesajul de eroare 'Number too large', iar execuția programului va fi încheiată.

Dacă, după ce toate spațiile și caracterele sfârșit-de-linie au fost ignorate, primul caracter nu este o cifră sau un semn ('+' sau '-'), va fi emis mesajul de eroare 'Number expected', iar execuția programului va fi oprită.

4)-v este de tip real;

În acest caz vor fi citite o serie de caractere reprezentînd un număr real, în acord cu sintaxa descrisă în capitolul II.

Ca și în cazul anterior, toate spațiile și caracterele sfârșit-de-linie din fața primului caracter sînt ignorate și, în mod analog, primul caracter trebuie să fie o cifră sau un semn. În caz contrar va fi emis mesajul de eroare 'Number expected'; dacă 'E' este prezent dar nu este urmat de o cifră sau un semn, este emis mesajul 'Exponent expected' și execuția programului este oprită.

Dacă numărul citit este prea mare sau prea mic, adică modulul lui nu aparține intervalului [5.9E-39,3.4E38], va apare mesajul 'Overflow' și execuția programului este oprită.

Avînd în vedere că în PASCAL HP4TM cel mai mare întreg este MAXINT=32767, orice număr mai mare trebuie tratat ca număr real.

Lungimea mantisei unui număr real este de 23 de biți. Din această cauză precizia obținută prin utilizarea numerelor reale este de aproximativ 6 cifre semnificative. Remarcăm că precizia scade dacă rezultatul unui calcul este mult mai mic decît valorile absolute ale argumentelor sale.

Exemplu:

2.00002 - 2 NU va fi 0.00002 ci 0.0!

Din cauza modului în care se reprezintă numerele reale în memorie, nu are sens să se utilizeze mai mult de 7 cifre semnificative cînd se specifică mantisa unui număr real, deoarece cifrele în plus sînt ignorate. Cînd precizia este importantă, este bine să fie evitate zerourile semnificative de după marca zecimală, deoarece ele sînt numărate ca cifre semnificative.

Exemplu:

0.000123456 este mai puțin precis decît 1.23456E-4.

Dacă în program este necesară citirea mai multor valori, de diferite tipuri, acestea pot fi tastate în continuare (fără a tasta ENTER - sfârșit de linie), dar separate între ele prin spații.

În programul READ (P.V.10), presupunem că în zona tampon avem șirul de caractere:

EXEMPLU*~~***~~ 13.1 33 -10 0314.9 -15E-1 99.04EOL

Prima procedură READ(CAR) atribuie variabilei CAR, de tip caracter, caracterul sfârșit-de-linie (EOL) existent în tampon în momentul lansării în execuție; variabila SIR primește valoarea 'EXEMPLU***'; următoarele două spații sînt ignorate, deoarece urmează a fi citită o variabilă reală A, căreia i se atribuie valoarea 13,1. Celor trei elemente din șirul de întregi (SIRDEINTREGI) li se atribuie pe rînd valorile 33,-10 și 314 (ultimul element este luat în considerare pînă la întîlnirea unui caracter care nu este cifră); variabilei CAR i se atribuie caracterul '.', iar celor două variabile reale B și C valorile 9 respectiv -15E-1. Dacă următorul READ citește o variabilă reală, acestea i se va atribui valoarea 99,04.

Rezultatele execuției programului READ (P.V.10) sînt separate prin caracterul '/' pentru o mai bună vizualizare a atribuirilor făcute prin citire.

```

ADS1 10 PROGRAM READ;
ADS1 20 VAR A,B,C:REAL;
ADS1 30 SIR:ARRAY[1..10] OF CHAR;
ADS1 40 I,INTREG:INTEGER;
ADS1 50 SIRDEINTREGI:ARRAY[1..3] OF INTEGER;
ADS1 60 CAR:CHAR;
ADS1 70 BEGIN
AD63 80 READ(CAR);
AD69 90 READ(SIR,A);
AD7B 100 FOR I:=1 TO 3 DO READ(SIRDEINTREGI[I]);
AD8E 110 READ(CAR,B,C);
ADDB 120 WRITE(CHR(16));
ADDF 130 WRITE(SIR,'/',A,'/');
ADFB 140 FOR I:=1 TO 3 DO WRITE(SIRDEINTREGI[I]);
AE3F 150 WRITE('/',CAR,'/',B,'/',C);
AE68 160 WRITE(CHR(16));
AE6F. 170 END {$P}.

```

EXEMPLU***/ 1.31000E+01/33 -10 314 ./ 9.00000E+00/-1.50000E+00

P. V. 10.

Procedura READLN

- asemenea procedurii READ, este folosită pentru a transmite date de la tastatură sau din fișierele utilizatorului;

Forma generală a apelului procedurii este:

READLN(v1,v2,...,vn);

Efectul este similar cu cel al procedurii standard READ, cu singura deosebire că după citirea valorilor variabilelor v1,v2,...,vn restul caracterelor din zona tampon pînă la caracterul CHR(13) se vor ignora, apoi se trece automat la începutul liniei următoare.

Efectul apelului procedurii standard READLN fără parametri (READLN;) este poziționarea citirii la primul caracter ce urmează după caracterul CHR(13) din zona tampon.

READLN poate fi folosită pentru "a sări" peste linia vidă prezentă la începutul execuției programului, deci are efectul citirii într-o nouă zonă tampon. Acest lucru este util atunci cînd prima variabilă citită este de tip caracter dar nu are nici un efect în cazul în care variabila este de tip REAL sau INTEGER (deoarece caracterele EOL sînt ignorate).

5.3.3. Reprezentarea datelor în memorie

Este foarte important pentru orice programator să cunoască modul de reprezentare a fiecărui tip de date în memorie precum și necesarul de memorie în fiecare caz.

1) Intregi

Intregii ocupă fiecare cite 2 octeți în memorie, fiind reprezentați în cod complementar.

Exemple:

1 = #0001
256 = #0100
-256 = #FF00

RegistruL Z80 standard pentru manipularea întregilor este registruL HL.

2) Caractere, valori logice și alte valori scalare

Acestea ocupă fiecare cite un octet, în format binar, fără semn. Pentru caractere este folosit codul ASCII extins, pe 8 biți.

Exemple:

'E' = #45
'I' = #5B

Pentru valori logice, deoarece:

ORD(TRUE) = 1
ORD(FALSE) = 0

TRUE se va reprezenta ca 1, iar FALSE prin 0. În mod analog, reprezentarea unei valori ordinale este dată de poziția ei în cadrul tipului respectiv.

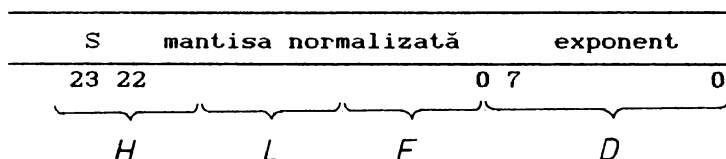
3) Numere reale

Se folosește formatul cu mantisă și exponent, similar celui folosit în notația științifică standard, însă cu reprezentare binară în locul celei zecimale.

Exemple:

$2 = 2 * 10^0$ sau $1.0_2 * 2^1$
 $1 = 1 * 10^0$ sau $1.0_2 * 2^0$
 $-12.5 = -1.25 * 10^1$ sau $-25 * 2^{-1} = -11001_2 * 2^{-1} = -1.1001_2 * 2^3$
 $0.1 = 1.0 * 10^{-1}$ sau $\frac{1}{10} = \frac{1}{1010_2} = \frac{0.1_2}{101_2} = 1.1001100_2 * 2^{-4}$

Un număr real este reprezentat pe 4 octeți în felul următor:



unde:

S - semnul mantisei: 1 = negativ; 0 = pozitiv;
mantisa normalizată - mantisa adusă la forma 1.XXXXXX, cu bitul cel mai semnificativ (bitul 22) egal totdeauna cu 1, cu excepția cazului cînd se reprezintă 0 (HL=0, DE=0);
exponent - exponentul este în binar, în cod complementar.

În memorie numerele reale sînt memorate sub forma EDLH.

Exemple:

Semn	Mantisa	Exponent
2= 01000000	00000000	00000000
└──────────┘	└──────────┘	└──────────┘
#40	#00	#01

deci LD HL, #4000
 LD DE, #0100

4) Inregistrări și tablouri

Inregistrările ocupă o cantitate de memorie egală cu totalul cerut pentru componentele sale.

În cazul *tablourilor*, dacă *n* este numărul de elemente din tablou și *s* este numărul de octeți ocupat de un element, atunci numărul de octeți ocupat de tablou este *n*s*.

Exemple:

```
ARRAY [1..10] OF INTEGER
    ocupă 10*2=20 octeți;
ARRAY [2..12,1..10] OF CHAR
    are 11*10=110 elemente și ocupă 110*1=110 octeți.
```

5) Mulțimi

Mulțimile sînt stocate ca șiruri de biți și deci dacă tipul de bază are *n* elemente, atunci numărul de octeți folosit este $(n-1) \text{ DIV } 8 + 1$.

Exemple:

```
SET OF CHAR necesită (256-1) DIV 8 +1=32 octeți,
SET OF (blue,green,yellow) necesită (3-1) DIV 8 +1=1 octet.
```

6) Pointeri - (tip referință)

Pointerii ocupă 2 octeți, care conțin adresa variabilei dinamice în format Z80, adică cu octetul cel mai puțin semnificativ în față.

5.3.4. Accesul direct al memoriei

HP4TM dispune de câteva proceduri standard prin intermediul cărora este permis accesul direct la anumite zone de memorie, fie pentru a insera o rutină în cod-mașină, fie pentru a apela o rutină în cod-mașină existentă; de asemenea, execuția programului poate fi oprită în anumite condiții.

Procedura INLINE

Forma generală a apelului procedurii este:

```
INLINE(C1,C2,...,Cn);
```

Această procedură permite ca în programul PASCAL să fie inserată o secvență cod-mașină Z80. Valorile (C1 MOD 256, C2 MOD 256,...) sînt inserate în program direct la adresa curentă dată de contorul de locații din cadrul compilatorului. C1,C2,... sînt constante întregi sau hexa, numărul lor putînd fi oarecare.

Programul EXINLINE (P.V.11) exemplifică modul de folosire a procedurii INLINE. Rutina prezentată realizează transferul informației din zona [23296;8] pe ecran, și anume dacă în zona (23296,8) sînt scrise caracterele '*INLINE*' atunci rutina va scrie în linia 10, coloana 12, cu FLASH, '*INLINE*'.

Remarcăm cîteva operații în plus:

a) Prima operație este încărcarea registrului IY cu adresa 23610. În această locație, care face parte din zona variabilelor sistem, va fi pus codul de eroare în caz că există.

b) Următoarea operație o reprezintă deschiderea canalului pentru ecran. Acest lucru se realizează apelînd rutina din ROM de la adresa #1601 cu valoarea 2 în registrul A. Transmiterea caracterelor se face folosind tot o rutină din ROM de la adresa #10 cu instrucțiunea RST și cu codul caracterului în registrul A.

Linia 320 introduce o nouă funcție INCH care returnează caracterul cu codul ASCII al tastei care este apăsată în acel moment. Dacă nu este apăsată nici o tastă, funcția ia valoarea CHR(0).

```
AFAC 10 PROGRAM EXINLINE;
AFAC 20 BEGIN
AFBE 30 PAGE;
AFCC 40 INLINE(#FD,#21,#3A,#5C,#3E,2,#CD,1,#16,#3E,#16,#D7);
AFCF 50 { linia 40 realizeaza
AFCF 60 LD IY,#5C3A ;in IY adresa cod eroare
AFCF 70 LD A,2 ;deschidere canal
AFCF 80 CALL #1601 ;spre ecran
AFCF 90 LD A,22 ;PRINT
AFCF 100 RST #10 ; AT }
AFCC 110 INLINE(#3A,10,#D7,#3E,12,#D7,#21,0,#5B,17,#4C,#59);
AFDB 120 { linia 110 realizeaza
AFDB 130 LD A,10 ;linia 10
AFDB 140 RST #10
AFDB 150 LD A,12 ;coloana 12
AFDB 160 RST #10
AFDB 170 LD HL,23296 ;in HL adresa de unde se transfera
AFDB 180 LD DE,22860 ;in DE adresa zona atribute }
AFDB 190 INLINE(6,8,#7E,#D7,#3E,#87,#12,#13,#23,5,#20,#F6,#3E,13,#D7);
AFEA 200 { linia 190 realizeaza
AFEA 210 LD B,8
AFEA 220 CICLU LD A,(HL) ;transfer caracter
AFEA 230 RST #10
AFEA 240 LD A,135
AFEA 250 LD (DE),A ;memorare atribute
AFEA 260 INC DE ;atribut urmator
AFEA 270 INC HL ;caracter urmator
AFEA 280 DEC B ;B<-- B-1
AFEA 290 JR NZ,CICLU ;repetă pina cind B=0
AFEA 300 LD A,13 ;linie noua
AFEA 310 RST #10 }
AFEA 320 REPEAT UNTIL INCH(<>CHR(0))
AFFS 330 END {$P}.
```

Procedura USER

Forma generală a apelului procedurii este:

```
USER(V);
```

Este o procedură cu un singur argument *V*, întreg, care reprezintă o adresă din memorie. Procedura determină un apel (CALL) la adresa *V* din memorie. Deoarece HP4TM memorează întregii în cod complementar, pentru referiri mai mari decît 32767 (#7FFF), trebuie să folosim valori negative pentru *V*.

Exemplu:

#C000 este -16384, deci USER (-16384) reprezintă un apel la adresa #C000. Atunci cînd ne referim la o adresă din memorie, este mai convenabil să folosim numere hexazecimale.

Rutina apelată trebuie să se incheie cu o instrucțiune Z80 RET (#C9) și trebuie să conserve registrul IX.

Programul EXUSER (P.V.12) realizează exact același lucru ca și programul READ (P.V.10) cu singura deosebire că el nu face parte din programul PASCAL, ci este apelat ca o rutină separată în cod-mașină. Remarcăm că RAMTOP-ul este la 59999, rutina la 60000 (#EA60). Și în acest caz rutina trebuie să conțină operația a) de la procedura anterioară - în caz contrar sistemul se blochează.

```
AC35 10 PROGRAM EXUSER;
AC35 20 BEGIN
AC47 30 PAGE;
AC4C 40 USER(#EA60);
AC52 50 REPEAT UNTIL INCH<>CHR(0)
AC5D 60 END {$P}.
```

P.V.12. ;

Procedura POKE

Forma generală a apelului procedurii este:

```
POKE(X,V);
```

unde:

X-expresie de tip întreg reprezentînd o adresă din memorie;

V-expresie de orice tip cu excepția tipului mulțime.

Procedura POKE stochează expresia *V* în memorie, începînd de la adresa *X*.

Exemple:

```
POKE(#6000,'A')   încarcă #41 în locația #6000;
POKE(-16384,3.6F3) memorează succesiv valorile #00, #0B,
                  #80, #70, în locațiile #C000, #C001,
                  #C002, #C003;
```

Programul EXUSER (P.V.13) exemplifică un mod de stocare în zona [23296,8] a unui set oarecare de caractere, care apoi poate fi vizualizat așa cum s-a arătat în programul EXINLINE (P.V.11), folosind procedura USER(#EA60). Procedura READLN utilizată la început permite citirea într-o nouă zonă tampon, apoi sînt citite pe rînd 8 caractere și memorate în zona indicată.

```

AC9B 10 PROGRAM EXUSER;
AC9B 20 VAR I:INTEGER;
ACA4 30   CAR:CHAR;
ACA4 40 BEGIN
ACAD 50 PAGE;
ACB2 60 READLN;
ACB5 70 FOR I:=0 TO 7 DO
ACCF 80   BEGIN
ACD2 90     READ(CAR);
ACD8 100    POKE(23296+I,CAR)
ACEB 110   END;
ACEF 120 USER(#EA60);
ACF5 130 REPEAT UNTIL INCH<>CHR(0)
AD00 140 END {$P}.

```

P. V. 13.

Programul **EXPOKESIUSER** (P.V.14) realizează același lucru, dar exemplifică modul de lucru al procedurii **POKE** pentru un tablou. Se observă că parametrii procedurii **POKE** sînt doar adresa primului octet (23296) și numele tabloului A. Memorarea întregului tablou este realizată de procedura **POKE**. Remarcăm că pot fi memorati și întregi reprezentați hexa, dar în acest caz ei sînt memorati în format Z80, adică octetul mai puțin semnificativ la prima adresă, urmat de octetul mai semnificativ.

```

AC7B 10 PROGRAM EXPOKESIUSER;
AC7B 20 VAR A:ARRAY[1..8] OF CHAR;
AC84 30 BEGIN
AC8D 40 PAGE;
AC92 50 READLN;
AC95 60 READ(A);
AC9D 70 POKE(23296,A);
ACAA 80 USER(#EA60);
ACB0 90 REPEAT UNTIL INCH<>CHR(0)
ACB8 100 END {$P}.

```

P. V. 14.

În urma rulării programului **EXPOKESIUSER** (P.V.15) pe ecran va apare scris:

```
'BABA/*/*'
```

iar memorarea s-a făcut cu:

```
POKE(23296,#4142) #41='A'
```

```
POKE(23298,#4142) #42='B'
```

adică:

```
(23296)=#42 'B'
```

```
(23297)=#41 'A'
```

```
(23298)=#42 'B'
```

```
(23299)=#41 'A'
```

```

AC7E 10 PROGRAM EXPOKESIUSER;
AC7E 20 BEGIN
AC90 30 PAGE;
AC95 40 POKE(23298,#4142);
ACA1 50 POKE(23300,'/*/*');
ACB5 60 POKE(23296,#4142);
ACC1 70 USER(#EA60);
ACC7 80 REPEAT UNTIL INCH<>CHR(0)
ACD2 90 END {$P}.

```

P. V. 15.

Observație:

Pentru ca programele P.V.12 - P.V.15 să poată lucra corect este necesar ca la adresa #EA60 (60000) să existe în momentul apelului **USER(#EA60)** rutina în cod-mașină, dată ca și

comentarii în programul EXINLINE (P.V.11), care trebuie să mai conțină la sfârșit o instrucțiune Z80 de revenire în programul PASCAL, RET (cod #C9).

Procedura OUT

Forma generală a apelului procedurii este:

```
OUT(P,C);
```

unde:

p-este un parametru de tip întreg, valoarea lui este încărcată în registrul BC;

c-este un parametru de tip caracter; valoarea lui este încărcată în registrul A, după care se execută instrucțiunea Z80: OUT A, (C).

Această procedură este folosită pentru a avea acces direct la porturile de ieșire ale microprocesorului Z80, fără a fi nevoie să folosim procedura INLINE.

Exemplu:

```
OUT(1,'A')-transmite caracterul 'A' la portul 1 al
microprocesorului Z80.
```

În programul EXOUT (P.V.16) se pot urmări diferite efecte sonore și efecte pe border-ul ecranului, mărinđ sau micșorinđ ciclul care simulează instrucțiunea PAUSE, sau schimbînd valorile transmise la portul 254.

```
ACAE 10 PROGRAM EXOUT;
ACAE 20 VAR I,J:INTEGER;
ACB7 30
ACB7 40 BEGIN
ACC0 50 FOR I:= 1 TO 3200 DO
ACDA 60 BEGIN
ACDD 70 OUT(254,CHR(123));
ACE8 80 FOR J:=1 TO 25 DO J:=J+1;
AD0F 90 OUT(254,CHR(97));
AD1A 100 FOR J:=1 TO 25 DO J:=J+1;
AD41 110 OUT(254,CHR(255))
AD4A 120 END;
AD4F 130 END {$P}.
```

P. V. 16.

Procedura HALT

Forma generală a apelului procedurii este:

```
HALT;
```

În urma execuției procedurii programul se va opri cu mesajul:

```
'HALT at PC=XXXX'
```

unde XXXX este adresa hexazecimală a locației de memorie de unde a fost emis HALT. Instrucțiunea HALT se folosește în depanarea programelor. Punînd instrucțiunea HALT pe fiecare din ramurile unui program se va putea determina ramura pe care se execută programul.

5.3.5.Proceduri HP4TM pentru lucrul cu caseta

Există două proceduri standard specifice implementării HP4TM pentru memorarea, respectiv încărcarea datelor de pe caseta magnetică.

Procedura TOUT

Forma generală a apelului procedurii este:

```
TOUT(NUME, ADRESA, LUNGIME);
```

Procedura TOUT (Tape OUTput) este folosită pentru a salva pe bandă variabile, zone de memorie etc. Primul parametru (NUME) este de tip ARRAY[1..8] OF CHAR și reprezintă denumirea fișierului care urmează să fie salvat. Vor fi salvați pe casetă LUNGIME octeți, începând de la adresa ADRESA. Ambii parametri (ADRESA și LUNGIME) trebuie să fie de tip întreg.

Exemplu:

```
    pentru a salva un SCREEN$ se poate folosi:
    TOUT('ECRAN ', 16384, 6912);
```

Bineînțeles, în același mod pot fi salvate variabile sau tablouri, cu condiția să fie cunoscută adresa și lungimea variabilei respective. Deoarece în PASCAL HP4TM există două funcții standard care permit aflarea adresei și lungimii unei variabile (vor fi studiate în paragraful 5.4), se poate folosi procedura TOUT și în felul următor:

```
TOUT('VAR V ', ADDR(V), SIZE(V));
```

care realizează salvarea pe casetă a variabilei V sub numele 'VAR V ', de la adresa ADDR(V) și având lungimea SIZE(V).

Programul EXTOUTTIN (P.V.17) constituie un exemplu de utilizare a procedurilor PAGE, WRITE, WRITELN, POKE pentru realizarea unui tabel pe ecran care conține valorile întregi I, ale pătratului variabilei, precum și ale radicalului variabilei cu 6 zecimale exacte. Pentru calculul radicalului s-a folosit funcția standard aritmetică SQRT.

```
ADC1  10 PROGRAM EXTOUTTIN;
ADC1  20 VAR I:INTEGER;
ADCA  30 BEGIN
ADD3  40 PAGE;
ADD8  50 WRITE(CHR(22),CHR(1),CHR(3));
ADED  60 WRITELN(' I      I*I      I^(1/2)');
AE16  70 WRITE(CHR(22),CHR(4),CHR(1));
AE28  80 WRITELN;
AE2E  90 FOR I:=1 TO 17 DO
AE48  100 WRITELN(I:5,I*I:10,SQRT(I):15:6);
AE87  110 I:=16384;
AE8D  120 REPEAT
AE8D  130 POKE(I,128);
AE9C  140 POKE(I+31,1);
AEB1  150 I:=I+32;
AEC0  160 UNTIL I>22495;
AED3  170 I:=0;
AED9  180 REPEAT
AED9  190 POKE(16384+I,#FFFF);
AEF3  200 POKE(16480+I,#FFFF);
AF0A  210 POKE(22496+I,#FFFF);
AF21  220 I:=I+2;
AF29  230 UNTIL I>30;
AF3C  240 REPEAT UNTIL INCH(<)CHR(0);
AF51  250 TOUT('ECRAN ',16384,6912)
AF6E  260 END {$P}.
```

```
AC42  10 PROGRAM EXTOUTTIN;
AC42  20 BEGIN
AC54  30 PAGE;
AC59  40 TIN('ECRAN ',16384);
AC6F  50 REPEAT UNTIL INCH(<)CHR(0)
AC7A  60 END {$P}.
```

I	I*I	I↑ (1/2)
1	1	1.000000
2	4	1.414213
3	9	1.732050
4	16	2.000000
5	25	2.236067
6	36	2.449489
7	49	2.645751
8	64	2.828427
9	81	2.984472
10	100	3.162277
11	121	3.316624
12	144	3.464101
13	169	3.605551
14	196	3.741657
15	225	3.872983
16	256	4.000000
17	289	4.123106

P. V. 17.

Procedura TIN

Forma generală a apelului procedurii este:

TIN(NUME, ADRESA);

Procedura TIN (Tape INput) este folosită pentru a încărca de pe bandă variabile sau zone de memorie care au fost salvate cu TOUT. Parametrul *NUME* este de tip ARRAY[1..8] OF CHAR, iar *ADRESA* este de tip întreg. Se caută pe bandă fișierul cu denumirea *NUME*, care este apoi încărcat în memorie de la adresa *ADRESA*. Numărul de octeți care se încarcă este luat din header-ul fișierului (salvat pe casetă cu TOUT).

Exemplu:

pentru a încărca variabila *V* salvată anterior cu TOUT, se folosește:

TIN('VAR V ', ADDR(V));

Observație:

Din cauză că fișierele sursă sînt înregistrate de editor în același format cu cel folosit de procedurile TOUT și TIN, cu TIN pot fi încărcate fișierele de text în variabile de tip tablou de caractere pentru prelucrare ulterioară.

5.3.6. Alocarea dinamică a memoriei

Alocarea dinamică a memoriei va fi tratată în detaliu în capitolul VI unde vor fi prezentate în amănunt și procedurile aferente: NEW, MARK, RELEASE.

5.3.7. Proceduri definite în program

Utilizarea procedurilor standard oferă posibilități largi programatorilor în rezolvarea de probleme, dar puterea limbajului PASCAL constă tocmai în posibilitățile nelimitate de concepere a procedurilor *proprie* fiecărui utilizator. Aceste proceduri se definesc în program, iar *după* ce au fost definite conform diagramelor din paragraful 5.3.1 pot fi folosite ca și procedurile standard obișnuite.

Programul **MEDIEINMATRICE** (P.V.18) calculează media elementelor maxime de pe fiecare linie a unei matrici. Programul conține două proceduri definite de utilizator:

ELMAX - liniile 130 - 190
 MEDIE - liniile 210 - 280

Prima procedură, **ELMAX**, are trei parametri formali: primul parametru, **I** - este transmis prin valoare; el are rolul de a indica linia în care se calculează maximul; ceilalți doi parametri sînt **A** (reprezentînd matricea în care se lucrează), respectiv **VALMAX** (care va transmite la ieșire elementul maxim din linia respectivă). Se observă că ultimii doi parametri sînt precedați de **VAR**, deci sînt parametri transmiși prin referință, lucru obligatoriu pentru **VALMAX**, pentru a putea transmite la ieșire valoarea calculată.

A doua procedură, **MEDIE**, are doi parametri formali, ambii transmiși prin referință - vectorul **V**, (care conține maximele de pe fiecare linie), respectiv **VALMED**, (care va conține la ieșire valoarea medie a maximelor). Transmiterea parametrului **VALMED** prin referință este obligatorie, acesta fiind un parametru de ieșire.

Este de remarcat, de asemenea, faptul că în afara parametrilor formali, în descrierea unei proceduri pot să apară și alte nume, reprezentînd diferite variabile. În programul **MEDIEINMATRICE** (P.V.18), procedura **MEDIE** folosește variabilele locale **I** și **X** și constanta globală **N**, definite în secțiunea de declarații a blocului program. În același program identificatorul **I** se utilizează atît ca variabilă globală în blocul programului principal, cît și ca variabilă locală în procedura **MEDIE**. Remarcăm că în momentul declarării variabilei locale **I**, variabila globală cu același nume din blocul apelant se conservă și orice modificare a valorii variabilei locale **I** afectează doar variabila din procedură.

```

B025 10 { PROGRAMUL CALCULEAZA MEDIA ELEMENTELOR MAXIME
B025 20 DE PE FIECARE LINIE A UNEI MATRICI }
B025 30
B025 40 PROGRAM MEDIEINMATRICE;
B025 50 CONST N=5;
B025 60 TYPE LINIE=ARRAY[1..N] OF REAL;
B025 70 MATRICE=ARRAY[1..N,1..N] OF REAL;
B025 80 VAR X:REAL;
B02E 90 VECTOR:LINIE;
B02E 100 MAT:MATRICE;
B02E 110 I,J:1..N;
B02E 120
B02E 130 PROCEDURE ELMAX(I:INTEGER;VAR A:MATRICE;VAR VALMAX:REAL);
B031 140 VAR J:1..N;
B031 150 BEGIN
B049 160 VALMAX:=A[I,1];
B09F 170 FOR J:=2 TO N DO
B0C2 180 IF VALMAX<A[I,J] THEN VALMAX:=A[I,J];
B188 190 END;
B193 200
B193 210 PROCEDURE MEDIE(VAR V:LINIE;VAR VALMED:REAL);
B196 220 VAR I:1..N;
B196 230 X:REAL;
B196 240 BEGIN
B1AE 250 X:=0;
B1C0 260 FOR I:=1 TO N DO X:=X+V[I];
B22F 270 VALMED:=X/N;
B24C 280 END;
B266 290
B266 300 BEGIN
B26F 310 FOR I:=1 TO N DO
B289 320 BEGIN
B28C 330 FOR J:=1 TO N DO

```

```

B2A6 340 BEGIN
B2A9 350 READ(X);
B2B3 360 MAT[I,J]:=X
B2E8 370 END;
B2FF 380 WRITELN;
B302 390
B302 400 ( *** apel procedura ELMAX *** )
B302 410
B302 420 ELMAX(I,MAT,X);
B313 430 VECTOR[IJ]:=X
B32F 440 END;
B343 450 PAGE;
B348 460 WRITE(CHR(16));
B34F 470 FOR I:=1 TO N DO
B369 480 BEGIN
B36C 490 FOR J:=1 TO N DO
B386 500 WRITE(MAT[I,J]:6:2);
B3D8 510 WRITELN
B3D8 520 END;
B3DE 530
B3DE 540 ( *** apel procedura MEDIE *** )
B3DE 550
B3DE 560 MEDIE(VECTOR,X);
B3EB 570 WRITELN;
B3EE 580 WRITELN('ELEMENTELE MAXIME ALE LINIILOR');
B41A 590 FOR J:=1 TO N DO WRITE(VECTOR[J]:6:2);
B46A 600 WRITELN;WRITELN;
B470 610 WRITELN('MEDIA ELEMENTELOR MAXIME: ',X:6:2);
B4AC 620 WRITELN;WRITE(CHR(16))
B4B6 630 END {$P}.

```

```

4.00 5.00 7.00 -8.00 0.00
2.00 -7.00 11.00 23.00 9.00
-9.00 5.00 23.00 -1.00 5.00
22.00 -8.00 0.00 -7.00 1.00
-6.00 34.00 -7.00 0.00 3.00

```

```

ELEMENTELE MAXIME ALE LINIILOR
7.00 23.00 23.00 22.00 34.00

```

```

MEDIA ELEMENTELOR MAXIME: 21.80

```

P. V. 18..

Observații:

1) O procedură poate avea un număr oarecare de parametri, eventual nici unul - de exemplu dacă în procedură doar se afișează mesaje. De asemenea vom avea proceduri definite fără parametri în acele cazuri când variabilele tratate în procedură sint globale.

Exemplu:

```

PROGRAM PP;
VAR A,B:REAL;
.....
PROCEDURE P;
.....
END;{P}
BEGIN {PP}
.....
P;
.....
END.

```

În procedura P se pot trata variabilele A și B ale căror valori noi se regăsesc în blocul apelant după fiecare apel al procedurii P. Remarcăm însă că de obicei procedurile vor avea parametri, ele prelucrând diferite valori corespunzător fiecărui apel. Astfel dacă procedura P va prelucra, corespunzător unui apel variabila A, iar pentru alt apel variabila B, structura programului va fi:


```

PROGRAM PP;
VAR A, B: REAL;
.....
PROCEDURE P (VAR X: REAL);
.....
END; {P}
BEGIN {PP}
.....
  P(A);
.....
  P(B)
.....
END.

```

În acest exemplu, parametrul formal X ar putea fi orice identificator, chiar și A sau B, acesta fiind folosit doar pentru descrierea operațiilor asupra valorilor transmise din blocul apelant.

2) În afara declarației variabilelor, într-o procedură pot să apară și alte definiții și declarații (de etichete, constante, tipuri; proceduri și funcții). Deci structura declarației unei proceduri este similară cu cea a programului (procedurii principale), descrisă în capitolul I. Deosebirea constă în faptul că etichetele, constantele, tipurile, variabilele, procedurile și funcțiile definite sau declarate într-o procedură sînt locale procedurii respective. Domeniul de valabilitate a fiecărei entități se stabilește conform regulilor prezentate în paragraful 5.1.

Exemplu:

```

10 PROGRAM PP;
20 TYPE T=-5..5;
30 VAR A: INTEGER;
40   B: T;
50
60 PROCEDURE P1(T: REAL);
70 LABEL 11;
80
90 PROCEDURE P2;
100 VAR B: T;
110 BEGIN {P2}
120   A:=B
130 END; {P2}
140
150 BEGIN {P1}
160   B:=T*A
170 END; {P1}
180
190 BEGIN {PP}
200   READ(A,B);
210   IF A<B THEN GOTO 11;
220   P2;
230 11: P1(A/B)
240 END. {PP}

```

Observații:

1) linia 100 va produce eroare - tipul T nu este tip pentru P1, deoarece identificatorul T este redeclarat în procedura P1 ca o variabilă reală;

2) în linia 120 A este din PP, iar B este din P2 dar este

nedefinită;

3) în linia 160 A și B sînt din PP, iar T este din P1 (parametru formal);

4) linia 210 va produce eroare - eticheta 11 este definită în P1, ea nu există în PP, este nedefinită;

5) linia 220 va produce eroare - procedura P2 este nedefinită în afara procedurii P1;

6) în linia 230 procedura P1, variabilele A și B sînt din PP, iar eticheta 11 este nedefinită.

Directiva FORWARD

Să considerăm un program în care două proceduri se apelează reciproc:

```

10 PROGRAM APELRECIPROC;
   {declarații}
100 PROCEDURE P(X:REAL);
110 {declarații P}
120 BEGIN
   ...
150 Q(X);
   ...
190 END; {sfîrșit P}
200 PROCEDURE Q(Y:REAL);
210 {declarații Q}
220 BEGIN
   ...
250 P(Y);
   ...
290 END; {sfîrșit Q}
300 BEGIN
   ...
900 END.

```

Un program cu structura de mai sus nu este acceptat de compilatorul PASCAL care cere ca la apelul unei proceduri aceasta să fie declarată cu întregul său bloc. Pentru a putea rezolva această situație, care apare relativ frecvent în practica programării în PASCAL, se utilizează directiva FORWARD, așa cum se poate vedea în diagrama de sintaxă din paragraful 5.3.1. Rolul acestei directive este de a declara un identificator ca nume de procedură împreună cu antetul său permițînd astfel apelul său înainte ca să fie dezvoltată întregul bloc al său.

În concluzie, structura variantei corecte a programului din exemplul anterior va fi următoarea:

```

10 PROGRAM APELRECIPROC;
   {declarații}
90 PROCEDURE Q(Y:REAL);FORWARD;
100 PROCEDURE P(X:REAL);
110 {declarații}
120 BEGIN
   ...
150 Q(X);
   ...
190 END; {sfîrșit P}
200 PROCEDURE Q;
210 {declarații Q}
220 BEGIN

```

```

...
250 P(Y);
...
290 END; (sfirsit Q)
300 BEGIN
...
900 END.

```

Se observă că procedura apelată înainte ca blocul său să fie dezvoltat este doar declarată prin antetul său, urmat de directiva FORWARD (linia 90). Pe de altă parte, declarația efectivă a procedurii, care a fost anterior "semnalată" cu FORWARD, nu conține lista parametrilor formali. În blocul procedurii se vor folosi parametrii formali din declarația de procedură cu FORWARD.

Programul SPATII (P.V.39) oferă un exemplu care necesită folosirea declarației anticipate a unei proceduri.

În finalul acestui paragraf prezentăm un program mai complex care calculează și afișează în ordine alfabetică și în ordinea descrescătoare a mediilor situația trimestrială a unei clase în care sînt cel mult 36 elevi, numărul de materii fiind maximum 16.

În programul MEDIEINCLASA (P.V.19) sînt utilizate 5 proceduri, fiecare îndeplinind o funcție indicată sugestiv prin numele procedurii:

CITNUM	- liniile	140	-	210
CITNOTE	- liniile	230	-	360
ORDONNUME	- liniile	380	-	570
ORDONNOTA	- liniile	590	-	780
LISTARE	- liniile	800	-	850

```

B1D4 20 PROGRAM MEDIEINCLASA;
B1D4 30 TYPE NUME=ARRAY[1..16] OF CHAR;
B1D4 40     MEDII=ARRAY[1..16] OF INTEGER;
B1D4 50     ELEV=RECORD
B1D4 60         NUM:NUME;
B1D4 70         MEDIA:REAL
B1D4 80     END;
B1D4 90     CLASA=ARRAY[1..36] OF ELEV;
B1D4 100 VAR CL:CLASA;
B1DD 110     N:INTEGER;
B1DD 120     I:1..36;
B1DD 130
B1DD 140 PROCEDURE CITNUM(VAR SIR:NUME);
B1E0 150 VAR I:INTEGER;
B1E0 160 BEGIN
B1F8 170     READLN;
B1FB 180     READ(SIR);
B206 190     FOR I:=1 TO 16 DO
B229 200         IF SIR[I]=CHR(0) THEN SIR[I]:=' ';
B281 210 END;
B289 220
B289 230 PROCEDURE CITNOTE(VAR MEDIE:REAL);
B28C 240 VAR S,I:INTEGER;
B28C 250     NOTE:MEDII;
B28C 260 BEGIN
B2A4 270     S:=0;
B2AD 280     FOR I:=1 TO 16 DO
B2D0 290         BEGIN
B2D3 300             WRITE('MARK NO. ',I:2,' ');
B2FA 310             READ(NOTE[I]);
B324 320             S:=S+NOTE[I];
B35D 330         END;
B360 340     IF NOTE[11]=0 THEN MEDIE:=S/15
B3AB 350     ELSE MEDIE:=S/16
B3D3 360 END;
B3EA 370

```

```

B3EA 380 PROCEDURE ORDONNUME(VAR C:CLASA);
B3ED 390 VAR I:1..35;
B3ED 400 SW:BOOLEAN;
B3ED 410 AUX:ELEV;
B3ED 420 BEGIN
B405 430 REPEAT
B405 440 SW:=FALSE;
B40C 450 I:=1;
B415 460 REPEAT
B415 470 IF C[I].NUM>C[I+1].NUM THEN
B47A 480 BEGIN
B47A 490 AUX:=C[I];
B4AD 500 C[I]:=C[I+1];
B4FF 510 C[I+1]:=AUX;
B533 520 SW:=TRUE;
B538 530 END;
B538 540 I:=I+1;
B545 550 UNTIL I>N-1;
B55C 560 UNTIL NOT SW;
B565 570 END;
B571 580
B571 590 PROCEDURE ORDONNOTA(VAR C:CLASA);
B574 600 VAR I:1..35;
B574 610 SW:BOOLEAN;
B574 620 AUX:ELEV;
B574 630 BEGIN
B58C 640 REPEAT
B58C 650 SW:=FALSE;
B593 660 I:=1;
B59C 670 REPEAT
B59C 680 IF C[I].MEDIA<C[I+1].MEDIA THEN
B613 690 BEGIN
B613 700 AUX:=C[I];
B646 710 C[I]:=C[I+1];
B698 720 C[I+1]:=AUX;
B6CC 730 SW:=TRUE;
B6D1 740 END;
B6D1 750 I:=I+1;
B6DE 760 UNTIL I>N-1;
B6F5 770 UNTIL NOT SW;
B6FE 780 END;
B70A 790
B70A 800 PROCEDURE LISTARE(C:CLASA);
B70D 810 VAR I:INTEGER;
B70D 820 BEGIN
B725 830 FOR I:=1 TO N DO
B74C 840 WRITELN(I:2,' ',C[I].NUM,' ',C[I].MEDIA:5:2)
B7E2 850 END;
B7F4 860
B7F4 870 BEGIN
B7FD 880 READ(N);
B803 890 FOR I:=1 TO N DO
B821 900 BEGIN
B824 910 WRITELN('NAME PLEASE:');
B83E 920 CITNUM(CL[I].NUM);
B863 930 WRITELN('MARK PLEASE:');
B87D 940 CITNOTE(CL[I].MEDIA)
B8A1 950 END;
B8AA 960 ORDONNUME(CL);
B8B3 970 PAGE;
B8B8 980 WRITELN(' *** ORDONARE ALFABETICA ***');WRITELN;
B8E5 990 LISTARE(CL);
B8FB 1000 ORDONNOTA(CL);
B904 1010 WRITELN;WRITELN(' *** ORDONARE DUPA MEDIE ***');WRITELN;
B934 1020 LISTARE(CL)
B945 1030 END {$P}.

```

5.4. FUNCȚII

5.4.1. Declaraarea funcțiilor

Funcțiile sînt subprograme care calculează și returnează subprogramului apelant o singură valoare. Pentru limbajul PASCAL, această valoare este în mod obligatoriu de tip simplu sau referință (capitolul VI). Această valoare se transmite în locul apelului funcției care astfel apare ca un operand într-o expresie oarecare.

O declarație de funcție poate fi identificată într-un program conform diagramei de sintaxă din fig.5.7.

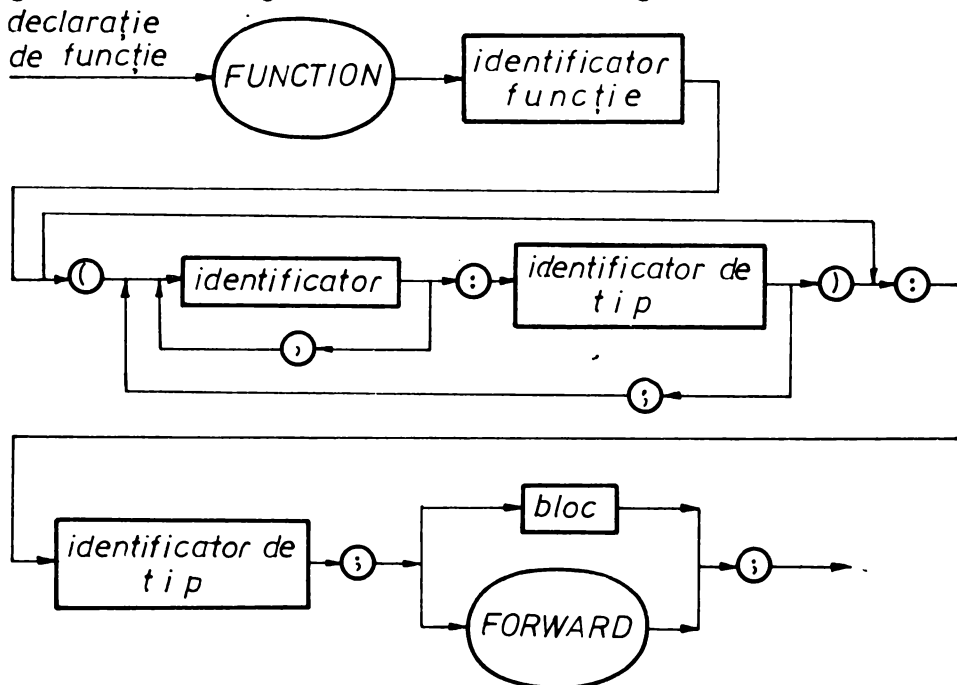


fig.5.7

Din diagramă se observă că există două forme ale declarației de funcție:

- funcție fără parametri;
- funcție cu parametri.

În cazul funcției cu parametri, lista parametrilor formali este identică cu cea de la proceduri.

Sintaxa declarației funcției seamănă cu sintaxa declarației procedurii. Rezultă deci că observațiile privind parametrii formali și efectivi sau domeniul de valabilitate este aplicabil și în cazul funcției.

Rezultatul calculat de o funcție este asociat numelui funcției, căruia conform diagramei de sintaxă, îi este precizat tipul. Numele funcției se va referi în blocul funcției ca orice variabilă. Nu se vor specifica parametrii decât în caz de apel recursiv. Din această cauză este obligatoriu ca numele funcției să apară cel puțin o dată în partea stîngă a unei instrucțiuni de atribuire de forma:

identificator-funcție:=expresie;

Ultima valoare atribuită lui *identificator-funcție* va fi rezultatul funcției.

În momentul apelului unei funcții, numele funcției apare ca *operand într-o expresie*, urmat de parametrii actuali.

5.4.2. Funcții standard specifice limbajului PASCAL HP4TM

Asemenea procedurilor standard, în limbajul PASCAL există o serie de funcții standard, (care evident nu trebuie declarate), ele asigurând o mulțime de facilități necesare oricărui program. Aceste funcții se pot grupa astfel:

- a) funcții de intrare;
- b) funcții de transfer;
- c) funcții aritmetice;
- d) alte funcții (existente în orice implementare);
- e) funcții specifice implementării HP4TM.

a) Funcții de intrare

Funcția EOLN

Forma generală a apelului funcției este: EOLN.

Este o funcție booleană care returnează valoarea TRUE când caracterul ce urmează să fie citit este un caracter *sfârșit-de-linie* (CHR(13)). În caz contrar funcția returnează valoarea FALSE. În cazul implementării HP4TM funcția nu are parametri.

Funcția INCH

Forma generală a apelului este: INCH.

Această funcție execută baleierea tastaturii calculatorului, iar în cazul în care una din taste este apăsată, returnează caracterul reprezentat de această tastă. Dacă nu a fost apăsată nici o tastă, este returnat caracterul CHR(0). Deci funcția este de tip caracter (CHAR). Nu are parametri.

Programul TESTEOLNINCH (P.V.20a) exemplifică un mod de utilizare a funcțiilor EOLN și INCH. Execuția se oprește când este apăsată o anumită tastă (în acest caz tasta C).

Se observă că introducerea șirului de caractere 'EOLN' urmat de <CR> va produce scrierea următorului rezultat:

```
FALSE E FALSE O FALSE L FALSE N TRUE
```

```

4CCC 10 PROGRAM TESTEOLNINCH:
ACCC 20 CONST C=CHR(13);
ACCC 30 P=CHR(16);
ACCC 40 VAR CAR:CHAR;
ACD5 50 BEGIN
ACDE 60 REPEAT
ACDE 70 READLN;
ACE4 80 REPEAT
ACE4 90 WRITE(P,EOLN:6,P);
AD04 100 READ(CAR);
AD0A 110 WRITE(P,CAR:2,P);
AD24 120 UNTIL EOLN;
AD2B 130 WRITE(P,EOLN:6,C,P);
AD4D 140 REPEAT UNTIL INCH IN ['C'];
AD77 150 UNTIL FALSE
AD79 160 END {$P}.

```

```
FALSE E FALSE O FALSE L FALSE N TRUE
FALSE T FALSE E FALSE S FALSE T FALSE A FALSE R FALSE E TRUE
```

P.V.20a

Dacă eliminăm linia 70, care are rol de a citi într-o nouă zonă tampon (vezi procedura READ, paragraful 5.3.2), atunci înainte de a introduce șirul 'EOLN' va apare scris TRUE și programul așteaptă introducerea șirului, deoarece zona tampon inițială are primul caracter CHR(13), deci acesta va fi primul citit (TESTEOLNINCH - P.V.20b).

```
ACD0 10 PROGRAM TESTEOLNINCH:
ACD0 20 CONST C=CHR(13);
ACD0 30 P=CHR(16);
ACD0 40 VAR CAR:CHAR;
ACD9 50 BEGIN
ACE2 60 REPEAT
ACE2 70 ( READLN; )
ACE2 80 REPEAT
ACE5 90 WRITE(P,EOLN:6,P);
AD05 100 READ(CAR);
AD08 110 WRITE(P,CAR:2,P);
AD25 120 UNTIL EOLN;
AD2C 130 WRITE(P,EOLN:6,C,P);
AD4E 140 REPEAT UNTIL INCH IN ['C'];
AD78 150 UNTIL FALSE
AD7A 160 END {$P}.
```

```
TRUE
FALSE E FALSE O FALSE L FALSE N TRUE
TRUE
FALSE T FALSE E FALSE S FALSE T FALSE A FALSE R FALSE E TRUE
TRUE
```

P.V.20b

b) Funcții de transfer

Funcția TRUNC

Forma generală a apelului funcției este: TRUNC(X).

Funcția are un parametru de tip real sau întreg, returnează cel mai mare întreg, mai mic sau egal cu X, pentru X pozitiv, sau cel mai mic întreg, mai mare sau egal cu X, dacă X este negativ. Deci funcția este de tip întreg.

Funcția ROUND

Forma generală a apelului este: ROUND(X).

Funcția are un parametru de tip real sau întreg. Ea returnează un rezultat de tip întreg și anume cel mai apropiat întreg față de X, în conformitate cu regulile standard de rotunjire.

Funcția ENTIER

Forma generală a apelului funcției este: ENTIER(X).

Funcția are un parametru de tip real sau întreg și returnează un rezultat de tip întreg - partea întreagă a lui X. Observație:

Funcția ENTIER este specifică implementării HP4TM. Este echivalentă cu INT din BASIC și este utilă când se scriu rutine rapide, pentru aplicații în matematică.

Funcția ORD

Forma generală a apelului funcției este: ORD(X).

Are un parametru de tip ordinal. Funcția returnează un rezultat de tip întreg care reprezintă numărul de ordine a valorii lui X în cadrul setului care definește tipul lui X. Dacă X este de tip întreg, atunci ORD(X)=X.

Funcția CHR

Forma generală a apelului funcției este: CHR(X).

Funcția are un parametru de tip întreg și returnează caracterul al cărui cod ASCII este X.

Din definiția funcțiilor de transfer ORD și CHR se observă că există relațiile:

ORD(CHR(i))=i și

CHR(ORD(c))=c

Observație:

În cazul funcțiilor cu parametru (prezentate anterior), parametrul actual poate fi o expresie de tipul cerut de funcție.

Programul FUNCTIITRANSFER (P.V.21) ilustrează modul de lucru cu funcțiile prezentate.

```

AF68 10 PROGRAM FUNCTIITRANSFER;
AF68 20 TYPE TRANSFER=(TRUN,ROUN,ENTIE,ORDIN,CAHR);
AF68 30 VAR X,PAS:REAL;
AF71 40     I:INTEGER;
AF71 50     F:TRANSFER;
AF71 60
AF71 70 PROCEDURE TRUNCROUNDENTIER;
AF74 80 BEGIN
AF9C 90   X:=-2.1;
AFA1 100  PAS:=0.1;
AFAE 110  WHILE X<6 DO
AFCF 120  BEGIN
AFCF 130  WRITE(' TRUNC(,X:4:1,)=',TRUNC(X):2);
B016 140  WRITE(' ROUND(,X:4:1,)=',ROUND(X):2);
B05D 150  WRITELN(' ENTIER(,X:4:1,)=',ENTIER(X):2);
B0A8 160  PAS:=PAS+0.1;
B0C1 170  X:=X+PAS;
B0CA 180  END;
B0DB 190  END;
B0E4 200
B0E4 210 PROCEDURE FUNCTIAORD;
B0E7 220 BEGIN
B0FF 230 WRITELN(' ORD(FALSE)=' ,ORD(FALSE):1);
B127 240 WRITELN(' ORD(TRUE)=' ,ORD(TRUE):1);
B14F 250 WRITELN(' ORD(ORDIN)=' ,ORD(ORDIN):1);
B178 260 I:=-1;
B181 270 WRITELN(' ORD(' ,I:2,')=' ,ORD(I):1);
B1B7 280 END;
B1C0 290
B1C0 300 PROCEDURE FUNCTIACHR;
B1C3 310 BEGIN
B1DB 320 FOR I:=32 TO 63 DO
B1F5 330 BEGIN
B1F8 340 WRITE(' CHR(' ,I:3,')=' ,CHR(I):1);
B231 350 WRITE(' CHR(' ,I+32:3,')=' ,CHR(I+32):1);
B27D 360 WRITELN(' CHR(' ,I+64:3,')=' ,CHR(I+64):1);
B2C9 370 END;
B2CC 380 END;
B2D5 390
B2D5 400 BEGIN
B2DE 410 WRITELN(CHR(16));WRITELN;
B2EB 420 TRUNCROUNDENTIER;WRITELN;
B2F3 430 FUNCTIAORD;WRITELN;
B2FB 440 FUNCTIACHR;WRITELN;
B303 450 WRITELN(CHR(16))
B30A 460 END {$P}.

```



```

TRUNC(-2.1)=-2  ROUND(-2.1)=-2  ENTIER(-2.1)=-2
TRUNC(-1.9)=-1  ROUND(-1.9)=-2  ENTIER(-1.9)=-2
TRUNC(-1.6)=-1  ROUND(-1.6)=-2  ENTIER(-1.6)=-2
TRUNC(-1.2)=-1  ROUND(-1.2)=-1  ENTIER(-1.2)=-2
TRUNC(-0.7)=0   ROUND(-0.7)=-1  ENTIER(-0.7)=-1
TRUNC(-0.1)=0   ROUND(-0.1)=0   ENTIER(-0.1)=-1
TRUNC(0.6)=0    ROUND(0.6)=1   ENTIER(0.6)=0
TRUNC(1.4)=1    ROUND(1.4)=1   ENTIER(1.4)=1
TRUNC(2.3)=2    ROUND(2.3)=2   ENTIER(2.3)=2
TRUNC(3.3)=3    ROUND(3.3)=3   ENTIER(3.3)=3
TRUNC(4.4)=4    ROUND(4.4)=4   ENTIER(4.4)=4
TRUNC(5.6)=5    ROUND(5.6)=6   ENTIER(5.6)=5

```

```

ORD(FALSE)=0
ORD(TRUE)=1
ORD(ORDIN)=3
ORD(-1)=-1

```

```

CHR(32)=CHR(64)=@  CHR(96)=`
CHR(33)=!  CHR(65)=A  CHR(97)=a
CHR(34)=\"  CHR(66)=B  CHR(98)=b
CHR(35)=#  CHR(67)=C  CHR(99)=c
CHR(36)=$  CHR(68)=D  CHR(100)=d
CHR(37)=%  CHR(69)=E  CHR(101)=e
CHR(38)=&  CHR(70)=F  CHR(102)=f
CHR(39)=\"  CHR(71)=G  CHR(103)=g
CHR(40)=(  CHR(72)=H  CHR(104)=h
CHR(41)=)  CHR(73)=I  CHR(105)=i
CHR(42)=*  CHR(74)=J  CHR(106)=j
CHR(43)=+  CHR(75)=K  CHR(107)=k
CHR(44)=,  CHR(76)=L  CHR(108)=l
CHR(45)=-  CHR(77)=M  CHR(109)=m
CHR(46)=.  CHR(78)=N  CHR(110)=n
CHR(47)=/  CHR(79)=O  CHR(111)=o
CHR(48)=0  CHR(80)=P  CHR(112)=p
CHR(49)=1  CHR(81)=Q  CHR(113)=q
CHR(50)=2  CHR(82)=R  CHR(114)=r
CHR(51)=3  CHR(83)=S  CHR(115)=s
CHR(52)=4  CHR(84)=T  CHR(116)=t
CHR(53)=5  CHR(85)=U  CHR(117)=u
CHR(54)=6  CHR(86)=V  CHR(118)=v
CHR(55)=7  CHR(87)=W  CHR(119)=w
CHR(56)=8  CHR(88)=X  CHR(120)=x
CHR(57)=9  CHR(89)=Y  CHR(121)=y
CHR(58)=:  CHR(90)=Z  CHR(122)=z
CHR(59)=;  CHR(91)=[  CHR(123)={
CHR(60)=<  CHR(92)=\  CHR(124)=|
CHR(61)==  CHR(93)=]  CHR(125)=}
CHR(62)=>  CHR(94)=^  CHR(126)=~
CHR(63)=?  CHR(95)=_  CHR(127)

```

P. V. 21.

c) Funcții aritmetice

Toate funcțiile aritmetice au un singur parametru, care trebuie să fie de tip real sau întreg.

Funcția ABS

Forma generală a apelului funcției este: ABS(X).

Funcția returnează un rezultat de același tip cu X, și anume valoarea absolută a lui X.

Funcția SQR

Forma generală a apelului este: SQR(X).

Funcția returnează pătratul lui X, având același tip ca variabila X.

Funcția SQRT

Forma generală a apelului funcției este: $SQRT(X)$.

Funcția returnează rădăcina pătrată a lui X , rezultatul fiind întotdeauna de tip real. În cazul în care parametrul X este negativ, se generează mesajul de eroare 'Math call error'.

Funcția FRAC

Forma generală a apelului funcției este: $FRAC(X)$.

Funcția returnează o valoare de tip real, egală cu partea fracționară a lui X , deci avem relația:

$$FRAC(X) = X - ENTIER(X).$$

Funcțiile trigonometrice: SIN, COS și TAN

Forma generală a apelurilor este: $SIN(X)$, $COS(X)$, $TAN(X)$.

Funcțiile returnează valorile funcțiilor trigonometrice corespunzătoare. Parametrul X se dă în radiani. Rezultatul funcțiilor este întotdeauna de tip real.

Funcția ARCTAN

Forma generală a apelului funcției este: $ARCTAN(X)$.

Funcția, de tip real returnează, în radiani, unghiul a cărui tangentă este egală cu X .

Funcția EXP

Forma generală a apelului funcției este: $EXP(X)$.

Funcția returnează valoarea lui e^x (e este baza logaritmului natural). Rezultatul este întotdeauna de tip real.

Funcția LN

Forma generală a apelului funcției este: $LN(X)$.

Funcția returnează logaritmul natural (în baza e) al lui X . Rezultatul este întotdeauna de tip real. Dacă $X \leq 0$, se generează mesajul 'Math call error'.

Programul **FUNCTIIARITMETICE** (P.V.22) ilustrează modul de lucru cu funcțiile aritmetice.

```

AFBB 10 PROGRAM FUNCTIIARITMETICE;
AFBB 20 VAR X,PAS:REAL;
AFC4 30 I:INTEGER;
AFC4 40
AFC4 50 PROCEDURE ARITMETICE;
AFC7 60 BEGIN
AFDF 70 X:=-2.1;
AFF4 80 PAS:=0.1;
B001 90 WHILE X<6 DO
B022 100 BEGIN
B022 110 WRITE(' ABS(',X:4:1,')= ',ABS(X):3:1);
B06A 120 WRITE(' SQR(',X:4:1,')= ',SQR(X):5:2);
B0B3 130 WRITE(' FRAC(',X:4:1,')= ',FRAC(X):3:1);
B0FD 140 WRITELN(' EXP(',X:4:1,')= ',EXP(X):7:3);

```

```

B149 150 PAS:=PAS+0.1;
B162 160 X:=X+PAS
B168 170 END
B17C 180 END:
B185 190
B185 200 PROCEDURE PARAMETRUPOZITIV;
B188 210 BEGIN
B1A0 220 X:=0.2;
B1AD 230 PAS:=0.1;
B1BA 240 WHILE X<9 DO
B1DB 250 BEGIN
B1DB 260 WRITE(' SQRT(' ,X:3:1,')=' ,SQRT(X):8:6);
B225 270 WRITELN(' LN(' ,X*10:2:0,')=' ,LN(X*10):8:6);
B2B6 280 PAS:=PAS+0.1;
B29F 290 X:=X+PAS
B2A8 300 END
B2B9 310 END:
B2C2 320
B2C2 330 PROCEDURE TRIGONOMETRICE;
B2C5 340 BEGIN
B2DD 350 X:=0.2;
B2EA 360 PAS:=0.1;
B2F7 370 WHILE X<9 DO
B318 380 BEGIN
B318 390 WRITE(' SIN(' ,X:3:1,')=' ,SIN(X):9:6);
B361 400 WRITE(' COS(' ,X:3:1,')=' ,COS(X):9:6);
B3AA 410 WRITE(' TAN(' ,X:3:1,')=' ,TAN(X):10:6);
B3F3 420 WRITELN(' ARCTAN(' ,X:3:1,')=' ,180*ARCTAN(X)/3.1415:2:0);
B459 430 PAS:=PAS+0.1;
B472 440 X:=X+PAS
B47B 450 END
B48C 460 END:
B495 470
B495 480 BEGIN
B49E 490 WRITELN(CHR(16));
B4A8 500 ARITMETICE;WRITELN;
B4B0 510 PARAMETRUPOZITIV;WRITELN;
B4B8 520 TRIGONOMETRICE;WRITELN;
B4C0 530 WRITELN(CHR(16))
B4C7 540 END {$P};

```

ABS(-2.1)=2.1	SQR(-2.1)= 4.41	FRAC(-2.1)=0.9	EXP(-2.1)= 0.122
ABS(-1.9)=1.9	SQR(-1.9)= 3.61	FRAC(-1.9)=0.1	EXP(-1.9)= 0.150
ABS(-1.6)=1.6	SQR(-1.6)= 2.56	FRAC(-1.6)=0.4	EXP(-1.6)= 0.202
ABS(-1.2)=1.2	SQR(-1.2)= 1.44	FRAC(-1.2)=0.8	EXP(-1.2)= 0.301
ABS(-0.7)=0.7	SQR(-0.7)= 0.49	FRAC(-0.7)=0.3	EXP(-0.7)= 0.497
ABS(-0.1)=0.1	SQR(-0.1)= 0.01	FRAC(-0.1)=0.9	EXP(-0.1)= 0.905
ABS(0.6)=0.6	SQR(0.6)= 0.36	FRAC(0.6)=0.6	EXP(0.6)= 1.822
ABS(1.4)=1.4	SQR(1.4)= 1.96	FRAC(1.4)=0.4	EXP(1.4)= 4.055
ABS(2.3)=2.3	SQR(2.3)= 5.29	FRAC(2.3)=0.3	EXP(2.3)= 9.974
ABS(3.3)=3.3	SQR(3.3)=10.89	FRAC(3.3)=0.3	EXP(3.3)= 27.112
ABS(4.4)=4.4	SQR(4.4)=19.36	FRAC(4.4)=0.4	EXP(4.4)= 81.450
ABS(5.6)=5.6	SQR(5.6)=31.36	FRAC(5.6)=0.6	EXP(5.6)=270.425

SQRT(0.2)=0.447213	LN(2)=0.693147
SQRT(0.4)=0.632455	LN(4)=1.386294
SQRT(0.7)=0.836660	LN(7)=1.945909
SQRT(1.1)=1.048808	LN(11)=2.397894
SQRT(1.6)=1.264911	LN(16)=2.772588
SQRT(2.2)=1.483239	LN(22)=3.091041
SQRT(2.9)=1.702938	LN(29)=3.367295
SQRT(3.7)=1.923537	LN(37)=3.610916
SQRT(4.6)=2.144760	LN(46)=3.828641
SQRT(5.6)=2.366431	LN(56)=4.025349
SQRT(6.7)=2.588435	LN(67)=4.204691
SQRT(7.9)=2.810692	LN(79)=4.369447

SIN(0.2)= 0.198669	COS(0.2)= 0.980066	TAN(0.2)= 0.202707	ARCTAN(0.2)=11
SIN(0.4)= 0.389419	COS(0.4)= 0.921060	TAN(0.4)= 0.422793	ARCTAN(0.4)=22
SIN(0.7)= 0.644217	COS(0.7)= 0.764842	TAN(0.7)= 0.842286	ARCTAN(0.7)=35
SIN(1.1)= 0.891206	COS(1.1)= 0.453597	TAN(1.1)= 1.964756	ARCTAN(1.1)=48
SIN(1.6)= 0.999574	COS(1.6)=-0.029198	TAN(1.6)=-34.234137	ARCTAN(1.6)=58
SIN(2.2)= 0.808498	COS(2.2)=-0.588499	TAN(2.2)= -1.373830	ARCTAN(2.2)=66
SIN(2.9)= 0.239253	COS(2.9)=-0.970957	TAN(2.9)= -0.246407	ARCTAN(2.9)=71
SIN(3.7)= -0.529832	COS(3.7)=-0.848102	TAN(3.7)= 0.624728	ARCTAN(3.7)=75
SIN(4.6)= -0.993690	COS(4.6)=-0.112160	TAN(4.6)= 8.859647	ARCTAN(4.6)=78
SIN(5.6)= -0.631273	COS(5.6)= 0.775560	TAN(5.6)= -0.813955	ARCTAN(5.6)=80
SIN(6.7)= 0.408442	COS(6.7)= 0.914386	TAN(6.7)= 0.442748	ARCTAN(6.7)=82
SIN(7.9)= 0.998942	COS(7.9)=-0.045990	TAN(7.9)=-21.720232	ARCTAN(7.9)=83

d) Alte funcții (existente în orice implementare)

În limbajul PASCAL există și alte funcții standard, unele regăsindu-se în toate implementările, altele fiind proprii lui HP4TM.

Funcția SUCC

Forma generală a apelului funcției este: SUCC(X).

Parametrul X poate fi de orice tip ordinal și returnează succesorul lui X în codul setului care definește tipul lui X.

Funcția PRED

Forma generală a apelului funcției este: PRED(X).

Parametrul X poate fi de orice tip ordinal și returnează predecesorul lui X în setul care definește tipul lui X.

Observație:

În cazul celor două funcții, SUCC și PRED, rezultatul funcției nu este definit dacă funcțiile se aplică ultimului, respectiv primului element din set. În implementarea HP4TM însă

PRED(primul element) = ultimul element,

SUCC(ultimul element) = primul element.

Exemplificarea funcțiilor PRED și SUCC se poate urmări în programul TESTPREDSUCC (P.V.23).

```

AF7B 10 PROGRAM TESTPREDSUCC;
AF7B 20 CONST P=CHR(16);
AF7B 30 TYPE ENUM=(INTRARE,TRANSFER,ARITMETICE,ALTE);
AF7B 40 VAR C:CHAR;
AF84 50 LOGIC:BOOLEAN;
AF84 60 FUNCTIE:ENUM;
AF84 70
AF84 80 PROCEDURE SCRIECHAR;
AF87 90 BEGIN
AF9F 100 READLN;
AFA2 110 REPEAT
AFA2 120 READ(C);
AFAE 130 WRITE(P,'C=',C,' PRED(',C,')=',PRED(C));
AFEE 140 WRITELN('SUCC(',C,')=',SUCC(C));
B01C 150 WRITELN(P);
B024 160 UNTIL EOLN;
B02B 170 WRITELN;
B02E 180 END;
B034 190
B034 200 PROCEDURE SCRUI;
B037 210 BEGIN
B04F 220 WRITE('PRED(',LOGIC,')=',PRED(LOGIC),CHR(13));
B080 230 WRITELN('SUCC(',LOGIC,')=',SUCC(LOGIC));
B0AD 240 END;
B0B3 250
B0B3 260 PROCEDURE SCRIELOGIC;
B0B6 270 BEGIN
B0CE 280 LOGIC:=FALSE;
B0D2 290 SCRUI;
B0DB 300 LOGIC:=TRUE;
B0E0 310 SCRUI;
B0E9 320 WRITELN;
B0EC 330 END;
B0F2 340
B0F2 350 PROCEDURE SCRIENUMERARE;
B0F5 360 BEGIN
B10D 370 FOR FUNCTIE:=INTRARE TO ALTE DO
B11C 380 BEGIN
B11F 390 WRITELN(ORD(SUCC(FUNCTIE)), ' ',ORD(PRED(FUNCTIE)));
B147 400 END;
B156 410 WRITELN;
B159 420 END;

```

```

B15F 430
B15F 440 PROCEDURE SCRIEMAXINT:
B162 450 BEGIN
B17A 460 WRITE(' PRED(',-MAXINT-1:6,')=' ,PRED(-MAXINT-1));
B185 470 WRITELN(' SUCC(' ,MAXINT:5,')=' ,SUCC(MAXINT))
B1E8 480 END:
B1F1 490
B1F1 500 BEGIN
B1FA 510 SCRIECHAR:
B1FF 520 WRITE(P):
B204 530 SCRIELOGIC:
B209 540 SCRIENUMERARE:
B20E 550 SCRIEMAXINT:
B213 560 WRITE(P)
B218 570 END {#P}.

```

```

C=A PRED(A)=0 SUCC(A)=B
C=X PRED(X)=1 SUCC(X)=2
C=- PRED(-)=* SUCC(-)=.
C+ PRED(+)=* SUCC(+)=.
C' PRED(')=& SUCC(')=&
C! PRED(!)=& SUCC(!)=&
C^ PRED(^)=^ SUCC(^)=^
C# PRED(#)=A SUCC(#)=C
C# PRED(a)=b SUCC(a)=b
C# PRED(t)=s SUCC(t)=u
PRED(FALSE)=TRUE
SUCC(FALSE)=TRUE
PRED(TRUE)=FALSE
SUCC(TRUE)=TRUE

```

```

1 255
2 0
3 1
4 2

```

```
PRED(-32769)=32767 SUCC(32767)=-32769
```

P. V. 23.

Funcția ODD

Forma generală a apelului funcției este: ODD(X).

Parametrul X trebuie să fie de tip întreg. Funcția este de tip logic și returnează TRUE dacă parametrul X este impar și FALSE dacă parametrul X este par. Programul TESTODD (P.V.24) exemplifică utilizarea acestei funcții.

```

ADCB 10 PROGRAM TESTODD:
ADCB 22 VAR INTREG:INTEGER;
ADCB 470
ADCB 40 PROCEDURE IMPARPAR:
ADCB 50 BEGIN
ADCF 60 FOR INTREG=-3 TO 5 DO
ADDC 70 WRITELN(' ODD(' ,INTREG:2,')=' ,ODD(INTREG):5)
AD4D 80 END:
AD59 90
AD59 100 BEGIN
AD62 110 WRITELN(CHR(16));WRITELN:
AD6F 120 IMPARPAR:
AD74 130 WRITELN(CHR(16))
AD7E 140 END {#P}.

```

```

ODD(-3)= TRUE
ODD(-2)=FALSE
ODD(-1)= TRUE
ODD( 0)=FALSE
ODD( 1)= TRUE
ODD( 2)=FALSE
ODD( 3)= TRUE
ODD( 4)=FALSE
ODD( 5)= TRUE

```

P. V. 24.

E) Funcții specifice implementării HP4TM

Funcția RANDOM

Forma generală a apelului funcției este: RANDOM.

Funcția returnează un număr pseudoaleator cuprins între 0 și 255 inclusiv.

În programul TESTRANDOM (P.V.25), procedura VERIFICA realizează compararea celor 100 de numere generate prin metoda propusă în procedura SCRIERANDOM indicând, dacă este cazul, egalitatea a două valori.

```

AES3 10 PROGRAM TESTRANDOM:
AES3 20 VAR I:INTEGER:
AES3 30 A:REAL:
AES3 40 B:ARRAY[1..100]OF REAL:
AES3 50 SW:BOOLEAN:
AES3 60
AES3 70 PROCEDURE SCRIERANDOM:
AES3F 90 BEGIN
AE77 90 FOR I:=1 TO 100 DO
AE91 100 BEGIN
AE94 110 A:=RANDOM/10E3+RANDOM/10E3/(RANDOM/10E3)*10:
AF01 120 IF A>1 THEN A:=A/10
AF38 130 ELSE
AF43 140 IF A<0.1 THEN
AF51 150 A:=A*10:
AF7A 160 B[I]:=A:
AF7A 170 WRITE(A:10:6):
AF8B 190 IF (I MOD 5)=0 THEN WRITELN:
AFD7 190 END:
AFDA 200 WRITELN
AFDA 210 END:
AFE3 220
AFE3 230 PROCEDURE VERIFICA:
AFE6 240 VAR J:INTEGER:
AFE6 250 BEGIN
AFEE 260 SW:=FALSE:
B002 270 FOR I:=2 TO 100 DO
B01C 280 FOR J:=1 TO I-1 DO
B047 290 IF B[J]=B[I]
B08E 300 THEN
B0A7 310 BEGIN
B0A7 320 SW:=TRUE:
B0AC 330 WRITELN(I,J,B[I])
B0E7 340 END
B0EA 350 END:
B0F8 360
B0F8 370 BEGIN
E101 380 WRITELN(CHR(16)):WRITELN:
B10E 390 SCRIERANDOM:
E113 400 VERIFICA:
B118 410 IF NOT SW THEN WRITELN('TOATE DIFERITE!!!'):
B140 420 WRITELN:WRITELN(CHR(16))
E14A 430 END {$P}.

```

0.427737	0.943255	0.542762	0.941071	0.124069
0.230459	0.221089	0.414285	0.818888	0.136888
0.147112	0.691257	0.167598	0.440023	0.130248
0.329424	0.779610	0.503913	0.689220	0.328926
0.257353	0.652167	0.142263	0.603077	0.146770
0.997301	0.700000	0.379518	0.119699	0.125336
0.212191	0.376096	0.101701	0.429474	0.560427
0.115489	0.418000	0.929999	0.132830	0.947872
0.744966	0.173282	0.552453	0.114088	0.149039
0.984046	0.484432	0.494118	0.198347	0.075455
0.728571	0.437400	0.030494	0.200686	0.051509
0.292280	0.274042	0.030000	0.564010	0.207278
0.352227	0.323926	0.168099	0.043056	0.151555
0.137020	0.029587	0.073280	0.171789	0.646923
0.101827	0.654084	0.229405	0.041216	0.671111
0.450079	0.106202	0.171826	0.932984	0.309661
0.139376	0.179294	0.201115	0.314210	0.936700
0.406779	0.376267	0.294366	0.101250	0.289420
0.367241	0.197200	0.247362	0.153000	0.232402
0.710181	0.017222	0.020887	0.388656	0.467478

TOATE DIFERITE!!!

P. V. 25.

Funcția ADDR

Forma generală a apelului funcției este: ADDR(V).

Funcția este de tip întreg și returnează adresa din memorie la care se află identificatorul desemnat de parametrul V care poate fi de orice tip.

Funcția SIZE

Forma generală a apelului funcției este: SIZE(V).

Asemenea funcției ADDR, argumentul funcției SIZE este un identificator de variabilă. Funcția returnează, ca și în cazul anterior, un rezultat de tip întreg, care reprezintă numărul de octeți ocupați în memorie de această variabilă.

În programul ADDRSIZE (P.V.26) au fost declarate variabile de tip întreg, real, logic, caracter, enumerat, mai multe tablouri unidimensionale de diferite tipuri și un tablou bidimensional de elemente reale. În program sînt listate valorile funcțiilor ADDR și SIZE pentru fiecare din variabilele declarate. Observăm că adresa la care a fost memorată variabila INTREG (#FF54) este mai mare decît adresa la care a fost memorată variabila REALUL (#FF50) cu toate că variabila INTREG a fost declarată înainte.

Imaginea zonei de memorie rezervată variabilelor:

SIRCHAR	SIRBOOL	MATRICE	SIR	TIP	CAR	BOOL	REALUL	INTREG
---------	---------	---------	-----	-----	-----	------	--------	--------

<-10->	<-10->	<-400->	<-20->	<-1->	<-1->	<-1->	<-4->	<-2->
--------	--------	---------	--------	-------	-------	-------	-------	-------

#FD95	#FD9F	#FDA9	#FF39	#FF4D	#FF4E	#FF4F	#FF50	#FF54
-------	-------	-------	-------	-------	-------	-------	-------	-------

Funcțiile ADDR și SIZE se pot utiliza împreună pentru a salva pe casetă variabile, tablouri folosind procedura TOUT prezentată în paragraful 5.3.5. Astfel linia 270 din programul ADDRSIZE (P.V.26) are ca efect salvarea tabloului bidimensional MATRICE indicîndu-se numele, ('MATRICE', de tip ARRAY[1..8] OF CHAR), adresa din memorie unde se află (ADDR(MATRICE)) și lungimea (SIZE(MATRICE)).

```

PROGRAM ADDRSIZE;
TYPE ENUM=(INTREG,REALUL,BOOL,CAR);
VAR INTREG:INTEGER;
    REALUL:REAL;
    BOOL:BOOLEAN;
    CAR:CHAR;
    TIP:ENUM;
    SIR:ARRAY[1..10] OF INTEGER;
    MATRICE:ARRAY[1..10,1..10] OF REAL;
    SIRBOOL:ARRAY[1..10] OF BOOLEAN;
    SIRCHAR:ARRAY[1..10] OF CHAR;
BEGIN
PAGE;
WRITE(CHR(22),CHR(0),CHR(0));
Writeln(CHR(16));Writeln;
Writeln('ADDR(INTREG)=' ,ADDR(INTREG):4:H, ' SIZE(INTREG)=' ,SIZE(INTREG));
Writeln('ADDR(REALUL)=' ,ADDR(REALUL):4:H, ' SIZE(REALUL)=' ,SIZE(REALUL));
Writeln('ADDR(BOOL)=' ,ADDR(BOOL):4:H, ' SIZE(BOOL)=' ,SIZE(BOOL));
Writeln('ADDR(CAR)=' ,ADDR(CAR):4:H, ' SIZE(CAR)=' ,SIZE(CAR));
Writeln('ADDR(TIP)=' ,ADDR(TIP):4:H, ' SIZE(TIP)=' ,SIZE(TIP));
Writeln('ADDR(SIR)=' ,ADDR(SIR):4:H, ' SIZE(SIR)=' ,SIZE(SIR));
Writeln('ADDR(MATRICE)=' ,ADDR(MATRICE):4:H, ' SIZE(MATRICE)=' ,SIZE(MATRICE));
Writeln('ADDR(SIRBOOL)=' ,ADDR(SIRBOOL):4:H, ' SIZE(SIRBOOL)=' ,SIZE(SIRBOOL));
Writeln('ADDR(SIRCHAR)=' ,ADDR(SIRCHAR):4:H, ' SIZE(SIRCHAR)=' ,SIZE(SIRCHAR));

```

```

WRITELN;WRITELN(CHR(16));
TOUT('MATRICE ',ADDR(MATRICE),SIZE(MATRICE))
END {$P}.

```

```

ADDR(INTREG)=EA5D SIZE(INTREG)=2
ADDR(REALUL)=EA59 SIZE(REALUL)=4
ADDR(BOGL)=EA58 SIZE(BOGL)=1
ADDR(CAR)=EA57 SIZE(CAR)=1
ADDR(TIP)=EA56 SIZE(TIP)=1
ADDR(SIR)=EA42 SIZE(SIR)=20
ADDR(MATRICE)=E8B2 SIZE(MATRICE)=400
ADDR(SIRBOGL)=E8A8 SIZE(SIRBOGL)=10
ADDR(SIRCHAR)=E89E SIZE(SIRCHAR)=10

```

P. V. 26.

Funcția PEEK

Forma generală a apelului funcției este: PEEK(X,T).

Primul parametru al acestei funcții este de tip întreg și specifică o adresă din memorie. Al doilea argument desemnează tipul rezultatului. Funcția extrage informații din memorie de la adresa X conform tipului definit de T.

Astfel, dacă în memorie la adresa #EA60 (60000) introducem, folosind procedura POKE, informația 'PASCAL', atunci prin extragerea informației de la această adresă, conform diferitelor tipuri, vom obține informații diferite. În programul EXPEEK (P.V.27), în liniile 160, 170 au fost folosite posibilitățile oferite de funcțiile PEEK, ORD, CHR pentru a transforma prima treime a ecranului din INV VIDEO în TRUE VIDEO.

```

AE06 10 PROGRAM EXPEEK;
AE06 20 VAR I:INTEGER;
AE0F 30
AE0F 40 BEGIN
AE18 50 POKE(#EA60,'Pascal');
AE2E 60 PAGE;
AE33 70 WRITE(CHR(22),CHR(0),CHR(0));
AE48 80 WRITELN(CHR(16));WRITELN;
AE55 90 WRITELN(PEEK(#EA60,ARRAY[1..6] OF CHAR));
AE60 100 WRITELN(PEEK(#EA60,CHAR));
AE6A 110 WRITELN(PEEK(#EA60,INTEGER));
AE7A 120 WRITELN(PEEK(#EA60,REAL));
AE8B 130 WRITELN(PEEK(#EA60,BOOLEAN));
AE95 140 FOR I:=0 TO 5 DO
AEAF 150 WRITE(ORD(PEEK(#EA60+I,CHAR)):2;H,' ');
AED4 160 WRITELN(CHR(13), ' *** APASA O TASTA ***');
AF03 170 WRITELN;WRITELN(CHR(16));
AF10 180 REPEAT UNTIL INCH<>CHR(0);
AF25 190 FOR I:=0 TO 2047 DO
AF3F 200 POKE(16384+I,CHR(255-ORD(PEEK(16384+I,CHAR))))
AF71 210 END {$P}.

```

```

Pascal
P
24912
2.46228E+29
TRUE
50 61 73 63 61 6C
*** APASA O TASTA ***

```

P. V. 27

Programul DVMPBY2 (P.V.27a) reprezintă o aplicație interesantă privind folosirea funcțiilor standard PEEK și ADDR. Programul realizează înmulțirea, respectiv împărțirea cu 2 a unui număr real. Pentru realizarea înmulțirii (împărțirii) se folosește reprezentarea internă a unui număr real (vezi paragraful 5.3.3).


```

AECE 10 { PROGRAMUL DEMONSTREAZA CUM SE FOLOSESC
AECE 20 FUNCTIILE STANDARD
AECE 30 PEEK
AECE 40 ADDR
AECE 50 SI PROCEDURA POKE
AECE 60 PENTRU MODIFICAREA VARIABILELOR PASCAL
AECE 70
AECE 80 PROGRAM DVMPBY2:
AECE 90 VAR r:REAL;
AED7 100 i:INTEGER;
AED7 110
AED7 120 FUNCTION DV2(x:REAL):REAL;
AEDA 130 BEGIN
AEF2 140 i:=ADDR(x)+1; {adresa exponent}
AEF8 150 POKE(i,PRED(PEEK(i,CHAR))); {decrementeaza exponent}
AF08 160 DV2:=x
AF08 170 END;
AF2A 180
AF2A 190 FUNCTION MP2(x:REAL):REAL;
AF2A 200 BEGIN
AF45 210 i:=ADDR(x)+1; {adresa exponent}
AF50 220 POKE(i,SUCC(PEEK(i,CHAR))); {incrementeaza exponent}
AF58 230 MP2:=x
AF58 240 END;
AF7D 250
AF7D 260 BEGIN
AF86 270 REPEAT
AF86 280 WRITE('Introdu numarul r=');
AFA6 290 READ(r);
AFB0 300 WRITELN(CHR(16));
AFBA 310 WRITELN(r:7:2, ' impartit cu 2 este ',DV2(r):7:2);
B00F 320 WRITELN(r:7:2, ' inmultit cu 2 este ',MP2(r):7:2);
B064 330 WRITELN;WRITELN(CHR(16))
B06E 340 UNTIL r=0
B082 350 END {$P}.

```

```

345.55 impartit cu 2 este 172.77
345.55 inmultit cu 2 este 691.10

```

```

-13.89 impartit cu 2 este -6.94
-13.89 inmultit cu 2 este -27.78

```

```

0.02 impartit cu 2 este 0.01
0.02 inmultit cu 2 este 0.04

```

```

99.00 impartit cu 2 este 49.50
99.00 inmultit cu 2 este 198.00

```

```

-466.00 impartit cu 2 este -233.00
-466.00 inmultit cu 2 este -932.00

```

```

0.00 impartit cu 2 este 0.00
0.00 inmultit cu 2 este 0.00

```

· P. V. 27a

Funcția INP

Forma generală a apelului funcției este: INP(P).

Această funcție se folosește pentru a avea acces direct la porturile de intrare ale microprocesorului Z80, fără a mai utiliza procedura INLINE. P este un parametru de tip întreg, valoarea lui fiind încărcată în dublul registru BC. Funcția furnizează un rezultat de tip caracter care se obține prin executarea instrucțiunii Z80: IN A, (C).

5.4.3. Funcții definite în program

Flexibilitatea limbajului este asigurată de posibilitatea folosirii funcțiilor standard precum și a celor proprii, definite de utilizator.

Reamintim că o funcție este un subprogram care calculează și returnează subprogramului apelant o singură valoare. În PASCAL valoarea calculată este în mod obligatoriu de tip *simple* sau referință (pointer).

După cum se observă din diagrama de sintaxă din fig.5.7, funcția poate să nu conțină nici un parametru. În cazul în care există parametri, sintaxa lor este identică cu cea de la proceduri. De altfel, toate precizările referitoare la parametrii formali și efectivi și domeniul de valabilitate sînt valabile și în cazul funcțiilor.

Deci deosebirea între funcție și procedură constă în faptul că o funcție returnează o singură valoare, în timp ce procedura poate să returneze mai multe valori sau niciuna. Apelul de funcție este un operand într-o expresie, iar apelul de procedură este o instrucțiune.

Pentru exemplificare considerăm programul **MEDIEINMATRICE** (P.V.28) care rezolvă aceeași problemă ca și P.V.18, respectiv calculează media elementelor maxime de pe fiecare linie a unei matrici. Atît procedura **ELMAX** cît și procedura **MEDIE** din P.V.18 calculează o singură valoare, elementul maxim de pe o linie, respectiv media elementelor dintr-un vector. Se poate pune problema înlocuirii acestor proceduri cu niște funcții. Liniile 130-210 conțin în acest caz corpul funcției **ELMAX**. În declararea funcției apar doi parametri: **I** de tip întreg (reprezintă indicele de linie) și **A** de tip **MATRICE** (tip definit în program). În linia 200 avem

```
ELMAX:=MAX;
```

în care numele funcției (fără parametri) apare în stînga atribuirii. O astfel de atribuire trebuie să apară în fiecare funcție definită. În acest fel valoarea atribuită se va transmite prin intermediul numelui funcției în blocul apelant.

Observație:

În locul variabilei **MAX** nu se putea folosi direct numele funcției **ELMAX**, deoarece apariția numelui funcției în dreapta unei instrucțiuni de atribuire sau într-o instrucțiune de control este echivalentă cu un apel recursiv, caz în care trebuie specificați și parametrii actuali.

Liniile 230-300 descriu funcția **VALMED**. Se observă totală identitate a corpului funcției cu corpul procedurii **MEDIE** din P.V.18., cu excepția liniilor 210, respectiv 230.

```
210 PROCEDURE MEDIE (VAR V:LINIE;VAR VALMED:REAL);
230 FUNCTION VALMED (V:LINIE):REAL;
```

În primul caz (linia 210 din P.V.18), valoarea calculată se transmite prin intermediul parametrului **VALMED**, transmis prin referință.

În al doilea caz (linia 230 din P.V.28), valoarea calculată este transmisă prin intermediul funcției **VALMED**.

```

B010 10 (* PROGRAMUL CALCULEAZA MEDIA ELEMENTELOR MAXIME
B010 20 DE PE FIECARE LINIE A UNEI MATRICI *)
B010 30
B010 40 PROGRAM MEDIEINMATRICE:
B010 50 CONST N=5:
B010 60 TYPE LINIE=ARRAY[1..N] OF REAL:
B010 70 MATRICE=ARRAY[1..N,1..N] OF REAL:
B010 80 VAR X:REAL:
B025 90 VECTOR:LINIE:
B025 100 MAT:MATRICE:
B025 110 I,J:1..N:
B025 120
B025 130 FUNCTION ELMAX(I:INTEGER;A:MATRICE):REAL:
B028 140 VAR J:1..N:
B028 150 MAX:REAL:
B028 160 BEGIN
B040 170 MAX:=A[I,1]:
B092 180 FOR J:=2 TO N DO
B095 190 IF MAX<A[I,J] THEN MAX:=A[I,J]:
B176 200 ELMAX:=MAX
B176 210 END:
B190 220
B190 230 FUNCTION VALMED(V:LINIE):REAL:
B1A0 240 VAR I:1..N:
B1A0 250 X:REAL:
B1A0 260 BEGIN
B1B0 270 X:=0:
B1CA 280 FOR I:=1 TO N DO X:=X+V[I]:
B23A 290 VALMED:=X/N
B250 300 END:
B260 310
B260 320 BEGIN
B275 330 FOR I:=1 TO N DO
B28F 340 BEGIN
B292 350 FOR J:=1 TO N DO
B2AC 360 BEGIN
B2AF 370 READ(X):
B2B9 380 MAT[I,J]:=X
B2F1 390 END:
B305 400 WRITELN:
B308 410
B308 420 ( *** apel functie ELMAX *** )
B308 430
B308 440 VECTOR[I]:=ELMAX(I,MAT)
B33B 450 END:
B34F 460 PAGE:
B354 470 WRITELN(CHR(16)):WRITELN:
B361 480 FOR I:=1 TO N DO
B37B 490 BEGIN
B37E 500 FOR J:=1 TO N DO
B398 510 WRITE(MAT[I,J]:6:2):
B3EA 520 WRITELN
B3EA 530 END:
B3F0 540 WRITELN:
B3F3 550 WRITELN('ELEMENTELE MAXIME ALE LINIILOR'):
B41F 560 FOR J:=1 TO N DO WRITE(VECTOR[J]:6:2):
B46F 570 WRITELN:WRITELN:
B475 580
B475 590 ( *** apel functie VALMED *** )
B475 600
B475 610 WRITELN('MEDIA ELEMENTELOR MAXIME: ',VALMED(VECTOR):6:2):
B4C4 620 WRITELN:WRITELN(CHR(16))
B4CE 630 END {$P}.

```

```

5.00 77.00 89.00 -9.00 5.00
55.00 23.00 -7.00 -5.00 0.00
-1.00 6.00 88.00 45.00 88.00
75.00 12.00 -3.00 77.00 5.00
56.00 76.00 -4.00 0.00 1.00

```

```

ELEMENTELE MAXIME ALE LINIILOR
89.00 55.00 88.00 77.00 75.00

```

```

MEDIA ELEMENTELOR MAXIME: 77.00

```

Observație:

În versiunea HP4TM, procedurile și funcțiile nu sînt acceptate ca parametri, lucru permis în alte implementări ale limbajului. În aceste implementări transmiterea parametrilor funcție sau procedură se realizează specificînd parametrul respectiv prin antetulul său. La apel, antetului îi corespunde un nume de funcție sau procedură utilizator, anterior definite.

Exemplu:

```

PROGRAM PARAMFUNCTIE;
  VAR A1, B1, A2, B2: REAL;
      N1, N2: INTEGER;
  FUNCTION TG(X: REAL): REAL;
  BEGIN
    TG:=TAN(X);
  END;
  FUNCTION SINCOS(X: REAL): REAL;
  BEGIN
    SINCOS:=SIN(X)+COS(X);
  END;
  FUNCTION INTEGRALA(FUNCTION F(X: REAL): REAL; A, B: REAL;
                    N: INTEGER): REAL;
  VAR L, S: REAL;
      I: INTEGER;
  BEGIN
    S:=0;
    L:=(B-A)/N;
    X:=A;
    FOR I:=1 TO N DO
      BEGIN
        S:=S+F(X)*L;
        X:=X+L
      END
    END;
  BEGIN
    READLN(A1, B1, N1);
    WRITELN(INTEGRALA(SINCOS, A1, B1, N1));
    READLN(A2, B2, N2);
    WRITELN(INTEGRALA(TG, A2, B2, N2));
  END.

```

5.4.4. Stocarea datelor în memorie

Utilizatorul limbajului PASCAL are nevoie de informații privind modul de stocare a variabilelor în timpul executării programului, stocare care diferă pentru:

- a) variabile globale - declarate în blocul principal al programului;
- b) variabile locale - declarate într-un bloc intern;
- c) parametri și valori - transmise către, sau de la proceduri și funcții.

a) Variabile globale

Variabilelor globale le este alocat un spațiu în stiva de execuție, începînd de la adrese mari spre adrese mici. Astfel, dacă la execuție adresa stivei este #B000, iar variabilele globale ale programului sînt declarate:

```

VAR I: INTEGER;
    CH: CHAR;
    X: REAL;

```

atunci

```

I (care ocupă 2 octeți) va fi stocat la #B000-2 și #B000-1
  adică #AFFE și #AFFF;
CH (care ocupă un octet) va fi stocat la #AFFE-1 adică la
  #AFFD;
X (care ocupă 4 octeți) va fi stocat la #AFF9, #AFFA, #AFFB,
  #AFFC;

```

(vezi și paragraful 5.4.3. programul ADDRSIZE (P.V.26)).

b) Variabile locale

Adresa de început a zonei alocate variabilelor locale aparținând unui bloc se păstrează în zona a cărei adresă se obține scăzând 4 din conținutul lui IX, adică la IX-4.

Exemplu:

```

PROCEDURE TEST;
  VAR I, J: INTEGER;

```

```

I-(întreg pe 2 octeți) va fi plasat la adresele
  IX-4-2 și IX-4-1, adică la IX-6, IX-5;
J-(întreg pe 2 octeți) va fi plasat la IX-8 și IX-7.

```

c) Parametri și valori returnate

Parametrii transmiși prin valoare sînt tratați la fel ca variabilele locale. Cu cît un parametru este declarat mai devreme, cu atît adresa de memorie la care este stocat este mai mare. Spre deosebire însă de variabilele locale, la IX+2 este fixată adresa cea mai mică și nu adresa cea mai mare.

Exemplu:

```

PROCEDURE TEST(I: REAL; J: INTEGER);

```

```

J-(i se alocă primele două locații) este stocat la
  IX+2 și IX+3;
I-(i se alocă patru locații) este stocat la IX+4,
  IX+5, IX+6, IX+7.

```

Pentru parametri transmiși prin adresă se copiază adresele lor pe stivă (se vor ocupa 2 octeți pentru fiecare adresă).

Exemplu:

```

PROCEDURE TEST(I: INTEGER; VAR X: REAL);

```

```

- adresa lui X este conținută în IX+2 și IX+3;
- valoarea lui I este conținută în IX+4 și IX+5.

```

Valorile returnate de funcții sînt plasate pe stivă deasupra primului parametru.

Exemplu:

```

FUNCTION TEST(I: INTEGER): REAL;

```

```

-I se găsește la IX+2 și IX+3;
-pentru valoarea returnată se rezervă spațiu la IX+4,
  IX+5, IX+6 și IX+7.

```

În finalul acestui paragraf vom exemplifica modul de dezvoltare *descendent* al programelor PASCAL, rezolvînd problema ordonării crescătoare a unui vector A.

Bineînțeles, în practică problema nu apare în această formă simplă. În general, elementele vectorului pot fi formate dintr-un set de informații (de exemplu: nume, clasă, medie etc), cu alte cuvinte putem avea un vector de articole. Unul din cîmpurile articolului (numit *cheie*) are un rol aparte, ordonarea făcîndu-se după valoarea acestui cîmp.

În funcție de locul în care sînt păstrate elementele vectorului A în timpul prelucrării, distingem două tipuri de sortări:

- sortare internă: elementele lui A sînt păstrate în memoria internă a calculatorului;
- sortare externă: elementele lui A sînt păstrate pe un suport extern.

În mod evident, metodele de sortare diferă în funcție de tipul sortării (internă sau externă).

În cazul sortării interne, există o multitudine de strategii de sortare, fiecare avînd avantajele și dezavantajele sale care sînt analizate în funcție de diverse criterii:

- memoria ocupată;
- număr de comparații;
- număr de deplasări ale elementelor;
- timp de execuție.

Luînd în considerare aceste criterii, s-a observat (statistic) că metoda care dă cele mai bune rezultate este metoda descrisă în 1962 de G.A.R.Hoare, numită *metodă de sortare rapidă (quicksort)*, pe care o vom descrie pe scurt în continuare.

Vom folosi vectorul A cu elemente numere întregi. Prin parcurgerea vectorului pornind de la ambele capete (pe rînd) și interschimbarea elementelor care nu sînt în relația cerută, se împarte vectorul în două părți, nu neapărat de lungime egală, cu proprietatea că toate elementele din prima parte sînt mai mici (sau mai mari) decît toate elementele din cea de-a doua parte. Unul din subvectori este memorat (prin indicii de început și de sfîrșit), iar cu cel rămas se procedează analog. Subsirurile memorate sînt prelucrate apoi pe rînd în același mod, în ordinea inversă memorării lor.

În programul **SORTAREQUICK** (P.V.29q) vom da o implementare nerecursivă a acestui algoritm, folosind ca stivă un tablou în care se memorează în ordine primul și ultimul element al vectorului care trebuie păstrat.

```

B25A 10 PROGRAM SORTAREQUICK;
B25A 20 CONST N=10;
B25A 30 TYPE ELEMENT=INTEGER;
B25A 40   SIR=ARRAY(1..N)OF ELEMENT;
B25A 50   INDICE=0..N;
B25A 60 VAR A: SIR;
B263 70
B263 80 PROCEDURE QUICKSORT;
B266 90 CONST I=100;
B266 100 VAR STIVA: ARRAY(1..I)OF INDICE;
B266 110 VIRF: INTEGER;
B266 120   PRIM,ULTIM,I,J: INDICE;
B266 130   MIJLOC: ELEMENT;
B266 140
B266 150   PROCEDURE PUSH(V: INDICE);
B269 160   BEGIN
B281 170     VIRF:=VIRF+1;
B29E 180     STIVA[VIRF]:=V;
B2D4 190   END;

```

```

B2DB 200
B2DB 210 FUNCTION POP:INDICE;
B2DE 220 BEGIN
B2F6 230 POP:=STIVA[VIRF];
B32A 240 VIRF:=VIRF-1;
B346 250 END;
B34D 260
B34D 270 PROCEDURE SWAP(I1,I2:INDICE);
B350 280 VAR AUX:ELEMENT;
B350 290 BEGIN
B368 300 AUX:=A[I1];
B38F 310 A[I1]:=A[I2];
B3D3 320 A[I2]:=AUX;
B3F1 330 END;
B407 340
B407 350 BEGIN
B41F 360 VIRF:=0;
B42F 370 PUSH(1);PUSH(N);
B441 380 REPEAT
B441 390 ULTIM:=POP;PRIM:=POP;
B46C 400 REPEAT
B46C 410 I:=PRIM;J:=ULTIM;
B49F 420 MIJLOC:=A[(PRIM+ULTIM) DIV 2];
B4ED 430 REPEAT
B4ED 440 WHILE A[I]<MIJLOC DO I:=I+1;
B54E 450 WHILE A[J]>MIJLOC DO J:=J-1;
B5AB 460 IF I<=J THEN
B5D0 470 BEGIN
B5D0 480 SWAP(I,J);
B5E0 490 I:=I+1;
B606 500 J:=J-1;
B61F 510 END
B61F 520 UNTIL I>J;
B642 530 IF I<ULTIM THEN
B666 540 BEGIN
B666 550 PUSH(I);
B677 560 PUSH(ULTIM);
B688 570 END;
B688 580 ULTIM:=J;
B6A0 590 UNTIL PRIM>ULTIM;
B6C3 600 UNTIL VIRF=0;
B6DD 610 END;
B6E8 620
B6E8 630 PROCEDURE CITESIESIR;
B6E8 640 VAR I:INDICE;
B6E8 650 BEGIN
B703 660 FOR I:=1 TO N DO
B726 670 BEGIN
B729 680 WRITE('A[' ,I:2, ']=');
B752 690 READ(A[I]);
B778 700 END;
B778 710 END;
B782 720
B782 730 PROCEDURE SCRIESIR;
B785 740 VAR I:INDICE;
B785 750 BEGIN
B79D 760 FOR I:=1 TO N DO WRITE(A[I]);
B7ED 770 Writeln;
B7F0 780 END;
B7F7 790
B7F7 800 BEGIN
B800 810 Writeln('SIRUL ARE ',N:2, ' ELEMENTE');
B838 820 Writeln('INTRODU ELEMENTELE SIRULUI:');
B861 830 CITESIESIR;
B866 840 Writeln('SIRUL INITIAL ESTE:');
B887 850 SCRIESIR;
B88C 860 Writeln;
B88F 870 QUICKSORT;
B894 880 Writeln('SIRUL ORDONAT ESTE:');
B8B5 890 SCRIESIR;
B8BA 900 END {$P}.

```

P.V.29q

Procedura QUICKSORT conține la rîndul ei blocul a trei subprograme:

- PUSH pentru memorarea în stivă a unei valori;
- POP pentru extragerea din stivă a unei valori;
- SWAP pentru interschimbarea a două valori.

În program avem o singură variabilă globală: tabloul A, iar variabilele STIVA, VIRF, PRIM, ULTIM, I, J, MIJLOC

sînt locale procedurii QUICKSORT, deci sînt globale pentru procedurile PUSH, POP, SWAP și necunoscute pentru procedurile CITESTESIR și SCRIESIR.

Variabila AUX este locală procedurii SWAP iar variabila I este locală procedurilor CITESTESIR, SCRIESIR.

Intre variabila I din procedura CITESTESIR și variabila I din procedura SCRIESIR nu se poate produce nici un incident deoarece fiecare este cunoscută doar în procedura în care este declarată și necunoscută în exterior.

5.5. RECURSIVITATE

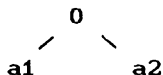
5.5.1. Generalități

Deși conceptul de recursivitate este cunoscut de mult în matematică, în programare a fost introdus odată cu apariția unor limbaje de nivel înalt (ALGOL, PASCAL).

Despre un obiect se spune că este recursiv dacă este definit prin el însuși.

Exemple:

- 1)-mulțimea numerelor naturale
 - a) 1 este număr natural;
 - b) succesorul unui număr natural este un număr natural
- 2)-funcția factorial ($n!$) -definită pentru numere naturale
 - a) $0! = 1$;
 - b) dacă $n > 0$, atunci $n! = n * (n-1)!$
- 3)-structura de tip arbore
 - a) un singur nod (O) este un arbore (arbore vid);
 - b) dacă a_1 și a_2 sînt arbori, atunci structura următoare este un arbore:



Puterea recursivității rezidă în posibilitatea de a defini un set infinit de obiecte printr-o relație sau un set finit de relații.

Recursivitatea nu trebuie confundată cu iterația, deși ele sînt strîns legate:

-iterația înseamnă execuția repetată a unei porțiuni de program, pînă în momentul în care este îndeplinită o anumită condiție; fiecare execuție se duce pînă la capăt, se verifică îndeplinirea condiției și în cazul unui anumit răspuns se reia execuția de la început (de exemplu: structurile repetitive WHILE, REPEAT, FOR);

-recursivitatea presupune, de asemenea, execuția repetată a unei porțiuni de program, dar, spre deosebire de iterație, în cadrul recursivității condiția de terminare este verificată în cursul execuției programului (nu la sfîrșit) și în cazul unui anumit rezultat, întreaga porțiune de program (care formează blocul unei proceduri sau funcții) este apelată din nou ca orice subprogram, în particular ca procedură a blocului care încă nu și-a terminat execuția; în momentul satisfacerii condiției de terminare, se reia execuția programului apelant

exact din punctul în care s-a apelat pe el însuși; acest lucru este valabil pentru toate apelurile anterioare satisfacerii condiției.

Subprogramele recursive necesită evaluarea unei condiții de terminare, fără de care un apel recursiv ar conduce la un ciclu infinit.

În programare, recursivitatea este exprimată prin proceduri sau funcții autoapelate. Dacă o procedură P conține o referință directă la ea însăși, se spune că este *direct recursivă*; dacă P conține o referință la o altă procedură Q, care la rândul ei conține o referință (directă sau indirectă) la P, se spune că P este *indirect recursivă*. Programul SPATII (P.V.39) este un exemplu pentru ilustrarea recursivității indirecte.

De regulă, unui subprogram i se asociază un set de obiecte locale subprogramului (constante, etichete, tipuri, variabile, proceduri, funcții) care sînt definite în subprogram și care nu există sau nu au sens în afara lui. Ori de cîte ori un astfel de subprogram este apelat recursiv, se creează un nou set de astfel de variabile locale, specifice apelului curent (alocîndu-li-se spațiu pe stiva de execuție). Deși aceste variabile au același nume ca și cele corespunzătoare lor din apelul anterior al subprogramului (în calitate de program apelant), ele au valori distincte și orice conflict de nume este evitat prin regulile care stabilesc domeniul de existență al identificatorilor. Identificatorii se referă întotdeauna la setul de variabile cel mai recent creat. Aceleași reguli sînt valabile și pentru parametrii de apel ai subprogramului care sînt asociați prin definiție setului de variabile.

În general, recursivitatea se folosește doar atunci cînd "adîncimea" recursivității (numărul apelurilor recursive generat de un anumit apel) este relativ mică (ea fiind în mod obligatoriu finită), deoarece fiecare apel recursiv al subprogramului necesită alocarea unui volum de memorie variabilelor sale curente. În plus, alături de aceste variabile, trebuie memorată și starea curentă a programului, cu scopul de a fi refăcută atunci cînd noua activare a subprogramului se termină și urmează ca activitatea anterioară să fie reluată. În concluzie, dacă în execuția unui program recursiv numărul apelurilor recursive este relativ mare, se mărește mult numărul valorilor care trebuie memorate și programul devine inefficient.

5.5.2. Program recursiv de ilustrare a mecanismului recursivității

Considerăm un șir de cuvinte urmate de cîte un blank. Se cere afișarea cuvintelor, așa cum au fost ele date, precum și inversate.

În programul INVERSARE (P.V.29) procedura recursivă INVERSEAZA citește cîte un caracter și îl afișează imediat prin intermediul variabilei locale Z de tip caracter. Dacă ultimul caracter citit nu este blank, procedura se autoapelează, în caz contrar se afișează caracterul Z.

Pînă la apariția primului blank, execuția procedurii are ca efect tipărirea caracterelor în ordinea în care au fost citite. Fiecare autoapel presupune salvarea în stiva sistemului

a contextului apelului, care în cazul de față este doar variabila locală Z.

Apariția primului blank presupune oprirea apelurilor recursive ale procedurii, declanșând continuarea execuției procedurii pentru fiecare din apelurile anterioare, din punctul următor apelului pînă la terminarea sa. În cazul de față aceasta înseamnă tipărirea caracterului memorat în variabila Z. Acest șir de reveniri va produce la început tipărirea unui blank, după care sînt tipărite caracterele în ordinea inversă citirii lor, deoarece fiecare terminare a execuției procedurii determină revenirea în apelul anterior, revenire care presupune reactualizarea contextului apelului. Deoarece contextele sînt salvate pe stivă, reactualizarea lor are loc în ordine inversă celei în care au fost memorate.

```

AD67 10 PROGRAM INVERSARE;
AD67 20
AD67 30 (* INVERSAREA UNOR CUVINTE
AD67 40      exemplu de
AD67 50      A P E L   R E C U R S I V
AD67 60
AD67 70 CONST EOL=CHR(13);
AD67 80 VAR I,N:INTEGER;
AD70 90
AD70 100 PROCEDURE INVERSEAZA;
AD73 110 VAR Z:CHAR;
AD73 120 BEGIN
AD88 130 READ(Z); WRITE(CHR(16),Z,CHR(16));
ADA5 140 IF Z<>' ' THEN INVERSEAZA;
ADBE 150 WRITE(CHR(16),Z,CHR(16));
ADD2 160 END;
ADD9 170
ADD9 180 BEGIN
ADE2 190 READ(N);
ADE8 200 FOR I:=1 TO N DO
AE06 210   BEGIN
AE09 220     INVERSEAZA;
AE0E 230     WRITE(CHR(16),EOL,CHR(16));
AE21 240   END
AE21 250 END {$P}.

```

INVERSARE ERASREVN!

TREI IERT
 DOI IOD
 PATRU URTAP
 CINCI ICNIC

INVERSARE

ERASREVN!

TREI IERT

DOI

IOD

P. V. 29.

În programul INVERSARE nu există nici o secvență de schimbare a elementelor și nu s-a declarat nici un tablou de caractere. Totuși, datorită apelurilor recursive, utilizînd valorile actuale de pe stiva de execuție se obține cuvîntul inversat. Modul de lucru al programului pentru inversarea cuvîntului 'DOI' se poate urmări în schema din fig.5.8.

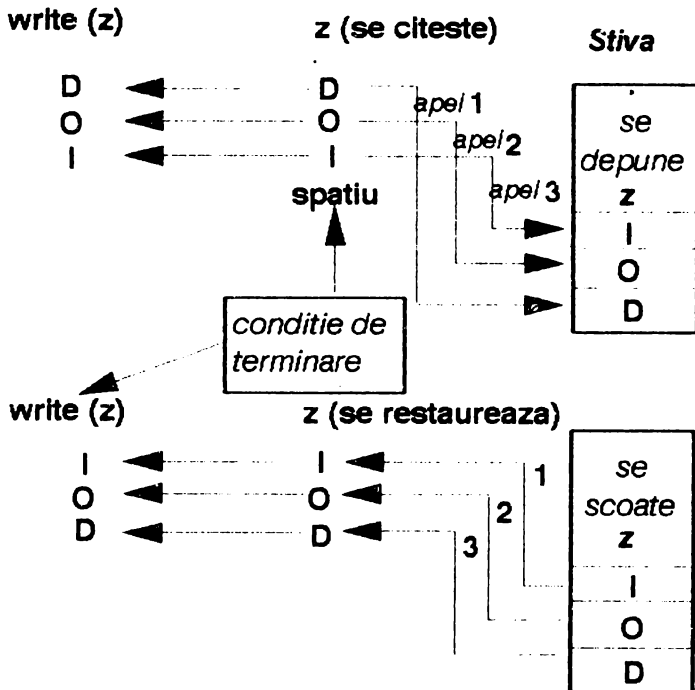


fig. 5.8

5.5.3. Utilizarea recursivității

Algoritmii recursivi sînt potriviți a fi utilizați în principiu în două cazuri:

- cînd *problema* este definită recursiv,
- cînd *datele* care trebuie prelucrate sînt definite în *termeni recursivi*.

Cu toate acestea, o formulare recursivă a problemei nu justifică întotdeauna utilizarea unui algoritm recursiv.

În cele ce urmează vom prezenta două probleme care, cu toate că se enunță în termeni recursivi, se rezolvă mai eficient iterativ.

1) Calculul factorialului

Un exemplu clasic în care se cere calculul unor valori care se definesc cu ajutorul unor relații recursive îl reprezintă factorialul unui număr, care, așa cum s-a văzut în paragraful 5.3.1., se definește astfel:

a) $0! = 1$;

b) dacă $n > 0$, atunci $n! = n * (n-1)!$

Se observă că oricare element, cu excepția primului, este definit folosind termenul anterior.

În programul FACTREC (P.V.30) este utilizat acest mod de calcul al factorialului, folosind o funcție recursivă.

```

AD5B 10 PROGRAM FACTREC:
AD5B 20
AD5B 30 (Algoritm RECURSIV pentru calculul
AD5B 40 FACTORIALULUI )
AD5B 50

```

```

ADS8 60 CONST EOL=CHR(13);
ADS8 70 VAR I:INTEGER;
AD64 80
AD64 90 FUNCTION FACT(N:INTEGER):INTEGER;
AD67 100 BEGIN
AD7F 110 IF N<0 THEN FACT:=-1
AD9A 120 ELSE
ADAA 130 IF (N=0) OR (N=1) THEN FACT:=1
ADD2 140 ELSE FACT:=N*FACT(N-1)
ADEA 150 END;
AE05 160
AE05 170 BEGIN
AE0E 180 WRITELN(CHR(16));WRITELN;
AE1B 190 FOR I:=-5 TO 7 DO
AE39 200 WRITE(I:2,'! = ',FACT(I),EOL);
AE6D 210 WRITELN(WRITELN(CHR(16)))
AE77 220 END {$P}.

```

```

-5! = -1
-4! = -1
-3! = -1
-2! = -1
-1! = -1
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040

```

P. V. 30

Observație:

Factorialul se definește pentru numere naturale. În program, dacă funcția este apelată pentru un număr negativ, anomalia se semnalează atribuind factorialului valoarea -1, urmată de abandonarea execuției funcției.

Este evident că pentru calculul factorialului recursivitatea poate fi înlocuită printr-o simplă iterație, așa cum se poate vedea în programul FACTITER (P.V.31).

```

AD3A 10 PROGRAM FACTITER;
AD3A 20 CONST P=CHR(16);
AD3A 30 VAR N,F:INTEGER;
AD43 40 { N - NUMAR CITIT }
AD43 50 { F - VALOARE FACTORIALA }
AD43 60
AD43 70 FUNCTION FAC(X:INTEGER):INTEGER;
AD46 80 VAR I,F:INTEGER;
AD46 90 BEGIN
AD5E 100 IF X<0 THEN FAC:=-1
AD79 110 ELSE
AD84 120 BEGIN
AD84 130 F:=1;
AD8D 140 FOR I:=2 TO X DO F:=F*I;
ADD5 150 FAC:=F
ADDS 160 END
ADE1 170 END;
ADEA 180
ADEA 190 BEGIN
ADF3 200 REPEAT
ADF3 210 READ(N);
ADF3 220 WRITELN(P,'N=',N,' N! = ',FAC(N),CHR(13),P)
AE44 230 UNTIL FALSE
AE49 240 END {$P}.
N=3 N!=6
N=6 N!=720
N=0 N!=1
N=-5 N!=-1
N=7 N!=5040
N=1 N!=1

```

P. V. 31

2) Problema generării numerelor lui Fibonacci

Această problemă se bazează de asemenea pe o definiție recursivă, totuși se tratează mult mai eficient cu ajutorul iterației. Numerele lui Fibonacci, sînt definite prin următoarea relație recursivă:

$FIB(0)=0$, $FIB(1)=1$,
 $FIB(n+1)=FIB(n)+FIB(n-1)$, pentru $n>0$.

Această definiție recursivă conduce la algoritmul de calcul din programul FIBOREC (P.V.32a).

```
AEC0 10 PROGRAM FIBOREC;
AEC0 20
AEC0 30 {Algoritm RECURSIV pentru calculul
AEC0 40 sirului lui FIBONACCI }
AEC0 50
AEC0 60 CONST EOL=CHR(13);
AEC0 70 VAR N:INTEGER;
AEC9 80
AEC9 90 FUNCTION FIB(N:INTEGER):INTEGER;
AEC0 100 BEGIN
AEE4 110 IF N=0 THEN FIB:=0 ELSE
AF05 120 IF N=1 THEN FIB:=1 ELSE
AF26 130 FIB:=FIB(N-1)+FIB(N-2)
AF44 140 END;
AF62 150
AF62 160 BEGIN
AF65 170 FOR N:=0 TO 23 DO
AF85 180 WRITE(' FIB(',N+1;2,')= ',FIB(N);EOL)
AFC8 190 END (#P).
```

P. V. 32a

Din păcate apelul recursiv utilizat în această situație conduce la o manieră ineficientă de calcul, deoarece numărul de apeluri crește exponențial odată cu n . De exemplu, pentru $n=5$ sînt necesare 15 apeluri, conform fig.5.9.

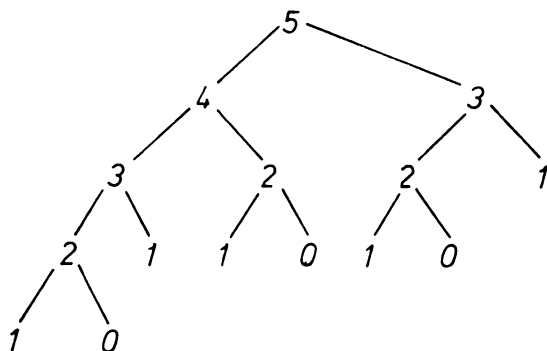


fig.5.9

Este evident că un astfel de program nu este practic și că numerele lui Fibonacci pot fi calculate cu ajutorul unei scheme iterative care elimină recalcularea valorilor deja calculate. Problema este rezolvată iterativ în programul FIBOITER (P. V. 32b).

```
AEC0 230 PROGRAM FIBOITER;
AEC0 240 VAR I:INTEGER;
AEC9 250
AEC9 260 {Algoritm ITERATIV pentru calculul
AEC9 270 sirului lui FIBONACCI }
AEC9 280
AEC9 290 FUNCTION FIB(N:INTEGER):INTEGER;
AEC0 300 VAR X,Y,Z:INTEGER;
```

```

AECC 310 BEGIN
AEE4 320 IF N=0 THEN FIB:=0 ELSE
AF05 330 BEGIN
AF05 340 I:=1; X:=1;Y:=0;
AF1D 350 WHILE I<N DO
AF37 360 BEGIN
AF37 370 I:=I+1;Z:=X;X:=X+Y;Y:=Z
AF64 380 END;
AF73 390 FIB:=X
AF73 400 END
AF7F 410 END;
AF88 420
AF88 430 BEGIN
AF94 440 FOR I:=0 TO 23 DO
AFAE 450 WRITE(CHR(16),'FIBO(',I+1;2,')=';FIB(I);CHR(13);CHR(16))
B001 460 END {$P}.

```

```

FIBO( 1)=0
FIBO( 2)=1
FIBO( 3)=1
FIBO( 4)=2
FIBO( 5)=3
FIBO( 6)=5
FIBO( 7)=8
FIBO( 8)=13
FIBO( 9)=21
FIBO(10)=34
FIBO(11)=55
FIBO(12)=89
FIBO(13)=144
FIBO(14)=233
FIBO(15)=377
FIBO(16)=610
FIBO(17)=987
FIBO(18)=1597
FIBO(19)=2584
FIBO(20)=4181
FIBO(21)=6765
FIBO(22)=10946
FIBO(23)=17711
FIBO(24)=28657

```

P. V. 32b

Eficiența celui de-al doilea mod de calcul al numerelor lui Fibonacci se observă clar urmărind timpul de calcul al primelor 20 de numere. In cazul algoritmului iterativ numerele sînt afișate practic instantaneu, pe cînd in cazul algoritmului recursiv pe măsura creșterii lui n , timpul de calcul devine din ce în ce mai mare.

Din considerentele de mai sus rezultă că utilizarea recursivității trebuie evitată ori de cîte ori există o rezolvare mai eficientă, bazată pe iterație.

Observație:

Majoritatea programelor recursive pot fi transformate în programe iterative, dar pe de altă parte recursivitatea are domenii bine stabilite, în care se aplică cu succes. In general se apreciază că algoritmi a căror natură este recursivă trebuie formulați ca proceduri recursive, ușurînd în acest fel activitatea de programare.

5.5.4. Tipuri de algoritmi recursivi

In cele ce urmează vom prezenta cîțiva algoritmi recursivi, grupați în mai multe categorii:

A)-algoritmi care implementează metode de rezolvare definite recursiv;

- B)-algoritmi de divizare; ("divide et impera")
- C)-algoritmi pentru aflarea tuturor soluțiilor unei probleme;
- D)-algoritmi cu revenire; ("backtracking")
- E)-algoritmi care necesită recursivitate indirectă;
- F)-algoritmi care prelucrează structuri de date definite recursiv (exemple în capitolul VI).

A. Algoritmi care implementează metode de rezolvare definite recursiv

Algoritmul pentru determinarea celui mai mare divizor comun (CMMDC) a două numere naturale, cunoscut sub numele de "algoritmul lui Euclid" poate fi descris recursiv în modul următor:

- a)-dacă unul dintre numere este nul, atunci CMMDC al lor este celălalt număr;
- b)-dacă nici unul din numere nu este nul, atunci CMMDC coincide cu CMMDC al numărului mai mic și al restului împărțirii numărului mai mare la cel mai mic.

Programul **EUCLID** (P.V.33) urmărește definiția recursivă de mai sus. Prin rularea lui se pot urmări pe rând valorile parametrilor M și N, pînă cînd unul devine zero.

```

AD56 10 PROGRAM EUCLID;
AD56 20 VAR A,B,X:INTEGER;
AD5F 30
AD5F 40 FUNCTION CMMDC(M,N:INTEGER):INTEGER;
AD62 50 BEGIN
AD7A 60 WRITELN('M=',M,'N=',N);
ADAF 70 IF N=0 THEN CMMDC:=M ELSE CMMDC:=CMMDC(N,M MOD N)
ADED 80 END;
AE07 90
AE07 100 BEGIN
AE10 110 WRITE('A=');READ(A);
AE23 120 WRITE('B=');READ(B);
AE36 130 WRITELN(CHR(16));WRITELN:
AE43 140 X:=CMMDC(A,B);
AE55 150 WRITE(CHR(13),'CMMDC(',A,CHR(8),' ',B,CHR(8),')=' ,X,CHR(13));
AEAF 160 WRITELN(CHR(16));
AEB9 170 END (*P).

```

```

M=456 N=234
M=234 N=222
M=222 N=12
M=12 N=6
M=6 N=0

```

```
CMMDC(456 ,234 )=6
```

P. V. 33

B. Algoritmi de divizare

Una din metodele fundamentale de proiectare a algoritmilor se bazează pe *tehnica divizării* (*divide et impera*). Ideea de bază constă în a descompune (a divide) problema complexă în mai multe subprobleme a căror rezolvare este mai simplă, soluțiile acestora contribuind la determinarea soluției problemei inițiale.

- 1) Algoritm pentru aflarea minimului și maximului unui vector

Este evident că această problemă se poate rezolva ușor printr-un proces iterativ care parcurge întregul vector, comparînd fiecare element cu cel mai mare, respectiv cel mai mic element determinat pînă la momentul comparării curente.

Pentru un vector cu n elemente numărul de comparații este $2n-2$ (numărul de comparații reprezintă unul din criteriile de clasificare a eficienței unui algoritm). Se observă că fiecare element este supus la două comparații. Acest inconvenient poate fi evitat utilizând *tehnica divizării*: se împarte vectorul în două subșiruri și se compară minimele și maximele acestor subșiruri, obținându-se minimul și maximul absolut. În continuare se procedează analog cu cele două subșiruri, de fiecare dată reducându-se la jumătate dimensiunea subșirurilor. Pentru dimensiunile 1 sau 2 soluția este imediată.

Metoda descrisă este implementată în programul MINMAX (P. V. 34).

```

AF37 10 PROGRAM MINMAX:
AF37 20 CONST N=10:
AF37 30 TYPE TAB=ARRAY[1..N] OF INTEGER;
AF37 40 VAR A:TAB;
AF40 50 I,INF,SUP:INTEGER;
AF40 60
AF40 70 PROCEDURE DOMENIU(A:TAB;I,J:INTEGER;VAR MIN,MAX:INTEGER);
AF43 80 VAR MIJLOC,MIN1,MAX1,MIN2,MAX2:INTEGER;
AF43 90 BEGIN
AF5B 100 IF J<=I+1 THEN
AF77 110 BEGIN
AF77 120 IF A[I]<A[J] THEN
AFCF 130 BEGIN
AFCF 140 MIN:=A[I];
B000 150 MAX:=A[J]
B029 160 END
B031 170 ELSE
B034 180 BEGIN
B034 190 MIN:=A[J];
B045 200 MAX:=A[I]
B08E 210 END
B096 220 END
B096 230 ELSE
B099 240 BEGIN
B099 250 MIJLOC:=(I+J) DIV 2;
B0BB 260 DOMENIU(A,I,MIJLOC,MIN1,MAX1);
B0F7 270 DOMENIU(A,MIJLOC+1,J,MIN2,MAX2);
B134 280 IF MAX1>MAX2 THEN MAX:=MAX1 ELSE MAX:=MAX2;
B174 290 IF MIN1<MIN2 THEN MIN:=MIN1 ELSE MIN:=MIN2
B1AA 300 END
B1B5 310 END:
B1C4 320
B1C4 330 BEGIN
B1CD 340 PAGE:
B1D2 350 WRITELN('Introdu elementele sirului:');
B1FB 360 FOR I:=1 TO N DO
B215 370 BEGIN
B218 380 WRITE('A',I:2,']=');READ(A[I])
B261 390 END;
B264 400 WRITE(CHR(16));
B26B 410 WRITELN;WRITELN('Sirul este:');
B287 420 FOR I:=1 TO N DO WRITE(A[I]:3);
B2CE 430 WRITELN;WRITELN;
B2DA 440 DOMENIU(A,1,N,INF,SUP);
B2FA 450 WRITELN(' INF=',INF,' SUP=',SUP,CHR(13));
B33E 460 WRITELN;WRITE(CHR(16))
B34E 470 END {$P}.

```

```

Sirul este:
 6 -9 0 3 55 1 61 -8 7 13
      INF=-9      SUP=61

```

P. V. 34.

Deși conceperea acestui program este mai dificilă decât în cazul algoritmului iterativ, din punctul de vedere al numărului de comparații este mai rapid. În literatura de specialitate se indică pentru acest algoritm un număr de $3n/2-2$ comparații, deci mai mic decât în cazul anterior.

2) Algoritm pentru determinarea permutărilor primelor n numere naturale

Principiul pe care se bazează algoritmul, folosind tehnica divizării, este următorul: *pentru a obține permutările de n elemente este suficient să fixăm pe rând câte un element și să permutăm celelalte $n-1$ elemente.* Procedând analog pentru cele $n-1$ elemente, se ajunge la permutări de 1 element (cazul banal).

Programul **PERMUTARE** (P.V.35) realizează calculul permutărilor primelor 4 numere naturale. Numerele se presupun memorate în tabloul $A[I], I=1,2,3,4$. Procedura **PERMUTA**, de parametru K , verifică dacă s-a ajuns la permutări banale ($K=1$), caz în care se afișează tabloul A , reprezentând o permutare. În cazul $K > 1$, fiecare element al tabloului este adus pe poziția K și se apelează procedura pentru $K-1$ elemente. După revenire se reface starea inițială schimbând din nou elementele din pozițiile I și K .

Condiția $K=1$ reprezintă condiția de terminare care limitează adâncimea apelului recursiv, determinând revenirea la apelurile anterioare.

```

AE75 10 PROGRAM PERMUTARE:
AE75 20
AE75 30 ( OBTINEREA PERMUTARILOR DE N ELEMENTE FOLOSIND
AE75 40 A P E L U L R E C U R S I V
AE75 50 se fixeaza un element si se obtin permutarile
AE75 60 pentru celelalte N-1 elemente ...
AE75 70
AE75 80 CONST N=4;CR=CHR(13);PR=CHR(16);
AE75 90 VAR I:INTEGER;
AE7E 100 A:ARRAY[1..N] OF 1..9;
AE7E 110
AE7E 120 PROCEDURE TIPARESTE;
AE81 130 VAR I:INTEGER;
AE81 140 BEGIN
AE99 150 FOR I:=1 TO N DO WRITE(A[I]);
AEE9 160 WRITE(CR)
AEEE 170 END;
AEF5 180
AEF5 190 PROCEDURE PERMUTA(K:INTEGER);
AEF8 200 VAR I,X:INTEGER;
AEF8 210 BEGIN
AF10 220 IF K=1 THEN TIPARESTE ELSE
AF31 230 BEGIN
AF31 240 FOR I:=1 TO K DO
AF58 250 BEGIN
AF5E 260 X:=A[I];A[I]:=A[K];A[K]:=X;
AFF2 270 PERMUTA(K-1);
B003 280 X:=A[I];A[I]:=A[K];A[K]:=X
B08C 290 END
B097 300 END
B099 310 END;
B0A4 320
B0A4 330 BEGIN
B0AD 340 FOR I:=1 TO N DO A[I]:=I;
B0F0 350 WRITE(PR);
B0F5 360 PERMUTA(N);
B0FE 370 WRITE(CR,PR)
B108 380 END {$P}.

```

```

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

```

```

2 4 1 7
4 2 1 6
4 1 1 5
1 4 4 4
2 2 1 4
1 1 1 4
1 1 1 4
2 2 1 4
1 1 1 4
1 1 1 4
1 1 1 4
1 1 1 4
1 1 1 4
1 1 1 4
1 1 1 4

```

P. V. 35

C. Algoritmi pentru determinarea tuturor soluțiilor unei probleme

- 1) Generarea partițiilor lui N
in părți egale cu 1 sau 2

Să se determine în cite moduri se poate tăia o bucată de material, astfel încît fiecare parte să aibă lungimea 1 sau 2, știind că bucata inițială are lungimea N, exprimată printr-un număr întreg.

Exemplu:

pentru N=3 există 3 posibilități:

- a) 1,1,1
b) 1,2
c) 2,1

Algoritmul se bazează pe următoarea tehnică de lucru:

-pentru N>1 există două posibilități:

- 1)-la început se taie o bucată de lungime 1 și bucata rămasă (de lungime N-1) se taie în toate modurile posibile;
2)-la început se taie o bucată de lungime 2 și bucata rămasă (de lungime N-2) se taie în toate modurile posibile;

-pentru N=1, avem o bucată de lungime 1;

-pentru N=0, nu există nici o posibilitate de tăiere.

În programul TAIETURA (P.V.36) lungimile bucăților tăiate sînt vizualizate prin semnele '.' (însemnînd o bucată de lungime 1) și '-' (semnificînd lungime 2). Acest mod de reprezentare grafică a tăieturilor necesită utilizarea unui tablou de caractere în care se depun pe rînd caracterele corespunzătoare tăieturilor.

Condiția de terminare o reprezintă atingerea dimensiunii 0 sau 1, cînd se afișează soluția găsită.

```

ADEC 10 PROGRAM TAIETURA;
ADEC 20 VAR I,K,X:INTEGER;
ADFS 30 Z:ARRAY[1..9] OF CHAR;
ADFS 40
ADFS 50 PROCEDURE F(N:INTEGER);
ADF8 60 BEGIN
AE10 70 IF N>1 THEN
AE26 80 BEGIN
AE26 90 K:=K+1;
AE2D 100 Z[K]:= '.';P(N-1);
AES8 110 Z[K]:= '-';P(N-2);
AE8A 120 K:=K-1;
AE92 130 END
AE91 140 ELSE

```

```

AE94 150 BEGIN
AE94 160 FOR I:=1 TO K DO WRITE(Z[I]);
AE94 170 IF N=1 THEN WRITE('. ');
AE94 180 WRITE(CHR(13))
AE94 190 END
AE94 200 END;
AE94 210
AE94 220 BEGIN
AE94 230 WRITELN(CHR(16));WRITELN:
AF13 240 FOR X:=1 TO 8 DO
AF20 250 BEGIN
AF30 260 PAGE;
AF35 270 WRITE(CHR(22),CHR(0),CHR(0));
AF4A 280 WRITELN('BUCATA ARE LUNGIMEA:'.X);
AF75 290 K:=0;
AF7B 300 WRITE(CHR(13));
AF82 310 P(X);
AF9B 320 REPEAT UNTIL INCH<>CHR(0);
AFA0 330 WRITELN:
AFA3 340 END;
AFA6 350 WRITELN;WRITELN(CHR(16))
AFB0 360 END (*P).
    
```

BUCATA ARE LUNGIMEA:1

.

BUCATA ARE LUNGIMEA:2

..

BUCATA ARE LUNGIMEA:3

...

BUCATA ARE LUNGIMEA:4

....

BUCATA ARE LUNGIMEA:5

.....

BUCATA ARE LUNGIMEA:6

.....

BUCATA ARE LUNGIMEA:7

.....

Drumul de ieșire din labirint este determinat cu ajutorul procedurii CAUTA căreia i se transmit ca și parametri coordonatele punctului curent (X,Y), ele fiind în același timp indici ai tabloului care materializează labirintul. Punctul de pornire este centrul labirintului, dar el poate fi oricare alt punct din interior (caz în care se modifică parametrul din apelul procedurii CAUTA). Căutarea se face în felul următor:

- dacă elementul curent este ' ', atunci el se marchează cu '.', fiind un element al unui posibil drum de ieșire;
- dacă s-a ajuns la una din margini, rezultă că s-a găsit un drum de ieșire din labirint și se afișează tabloul;
- în caz contrar se apelează recursiv procedura CAUTA pentru cele patru puncte din vecinătatea imediată a celui curent, deci pentru punctele (X+1,Y), (X,Y+1), (X-1,Y), (X,Y-1).

Deoarece traseul este marcat cu ajutorul caracterului '.', parcurgerea unui drum deja ales nu este posibilă. Este de asemenea important că marcajul de drum se șterge imediat ce s-a ajuns într-o fundătură. Ștergerea se execută prin punerea în tablou a unui caracter ' ' pe poziția curentă, înainte de părăsirea procedurii.

```

0061 10 PROGRAM LABIRINT;
0061 20 CONST N=6;
0061 30 VAR I,J,SOL:INTEGER;
006A 40 M:ARRAY[0..N,0..N] OF CHAR;
006A 50
006A 60 PROCEDURE TIPARIRE;
006D 70 BEGIN
0085 80 WRITE(CHR(22),CHR(0),CHR(0));
009A 90 SOL:=SOL+1;
00A1 100 WRITELN(CHR(16));
00AB 110 WRITELN;WRITELN('Solutia:',SOL);WRITELN;
00D0 120 FOR I:=0 TO N DO
00EA 130 BEGIN
00ED 140 WRITE(' ');
0100 150 FOR J:=0 TO N DO WRITE(M[I,J]);
0157 160 WRITELN
0157 170 END;
015D 180 WRITELN;WRITELN(CHR(16))
0167 190 END;
0170 200
0170 210 PROCEDURE CAUTA(X,Y:INTEGER);
0173 220 BEGIN
018B 230 IF M[X,Y]=' ' THEN
01D3 240 BEGIN
01D3 250 M[X,Y]='.';
0211 260 IF ((X MOD N)=0) OR ((Y MOD N)=0) THEN
0248 270 BEGIN
0248 280 TIPARIRE;
0251 290 REPEAT UNTIL INCH<>CHR(0)
025C 300 END
0266 310 ELSE
0269 320 BEGIN
0269 330 CAUTA(X+1,Y);
0281 340 CAUTA(X,Y+1);
0299 350 CAUTA(X-1,Y);
02B1 360 CAUTA(X,Y-1)
02C0 370 END;
02C9 380 M[X,Y]=' '
0306 390 END
0307 400 END;
0311 410
0311 420 PROCEDURE POKELABIRINT;
0314 430
0314 440 ( *** generare LABIRINT ***
0314 450 se poate modifica )
0314 460
0314 470 BEGIN
032C 480 POKE(ADDR(M[0]),'*****');
035E 490 POKE(ADDR(M[1]),' * * ');
0390 500 POKE(ADDR(M[2]),' * * ');
03C2 510 POKE(ADDR(M[3]),' * * * ');
03F4 520 POKE(ADDR(M[4]),' * * * ');
0426 530 POKE(ADDR(M[5]),' ** * ');
0458 540 POKE(ADDR(M[6]),' ** **** ');
048A 550 END;

```

```

B490 560
B490 570 BEGIN
B499 580 ( *** introducere LABIRINT ***
B499 590 READLN;
B499 600 FOR I:=0 TO N DO
B499 610 BEGIN
B499 620 FOR J:=0 TO N DO READ(MCI,J);
B499 630 READLN;
B499 640 END; )
B499 650
B499 660 PAGE;
B49E 670 POKELABIRINT;
B4A3 680 SOL:=-1;
B4AC 690 TIPARIRE;
B4B1 700 REPEAT UNTIL INCH<>CHR(0);
B4C6 710 CAUTA(N DIV 2,N DIV 2)
B4DE 720 END {$P}.

```

Solutia:0

```

*****
* * *
* * *
* * **
* **
** *
** ****

```

Solutia:1

```

*****
* * *
* * *
* * **
* **
** *
** ****

```

Solutia:2

```

*****
*...*
*...*
*...**
*...**
**.*
**.*

```

Solutia:3

```

*****
*...*
*...*
*...**
*...**
**.*
**.*

```

Solutia:4

```

*****
* * *
* * *
* * **
* **
** *
** ****

```

Solutia:5

```

*****
*...* *
*. . . *
*.*.*.*
*...**
** *...
** ****

```

Solutia:6

```

*****
* * *
*. * *
*.*. **
*... **
** *
** ****

```

P.V.37.

D) Algoritmi cu revenire

Acești algoritmi se bazează pe ideea găsirii soluțiilor unor probleme, nu pe baza unui set fix de reguli de calcul, ci prin încercări repetate și reveniri în caz de nereușită. Modalitatea de realizare a acestei tehnici constă în descompunerea obiectivului în obiective parțiale. De obicei acestea sînt exprimate în termeni recursivi și constau în prelucrarea unui număr finit de situații. Obținerea unor soluții parțiale sau finale care nu satisfac cerințele, provoacă revenirea recursivă în cadrul procesului de calcul pînă la obținerea soluției dorite, motiv pentru care acești algoritmi se numesc *algoritmi cu revenire (backtracking)*.

Problema celor opt regine

Un exemplu binecunoscut de utilizare a algoritmilor cu revenire îl reprezintă problema celor 8 regine. Această problemă poate fi rezolvată prin încercări, necesitînd o mare cantitate de muncă, răbdare, precizie, - atribute în care calculatorul excelează, depășind omul chiar atunci cînd acesta este un geniu. (Pînă în prezent nu a fost găsită o soluție analitică completă a acestei probleme, deși cercetarea ei a fost începută în 1850 de către C.F. GAUSS, care însă nu a rezolvat-o complet.)

Problema este următoarea: să se dispună pe tabla de șah 8 regine, astfel încît nici una dintre ele să nu le atace pe celelalte, ținînd cont de regulile jocului de șah.

Prima problemă care apare este aceea a reprezentării pozițiilor celor 8 regine pe tabla de șah. Ideea de a reprezenta tabla de șah cu ajutorul unei matrici de 8×8 este cea mai simplă, dar, din păcate, conduce la calcule groaie și complicate pentru determinarea cîmpurilor disponibile.

Intrucît pe fiecare coloană L este plasată o singură regină ($1 \leq L \leq I$), se poate face următoarea alegere de structurare a datelor:

```

VAR X: ARRAY[1..8] OF INTEGER;
    A: ARRAY[1..8] OF BOOLEAN;
    B: ARRAY[2..16] OF BOOLEAN;
    C: ARRAY[0..14] OF BOOLEAN;

```

cu următoarele semnificații:

- X[I] precizează poziția reginei în coloana I;
- A[J]=TRUE precizează că nici o regină nu amenință linia J
- B[K]=TRUE precizează că nici o regină nu amenință diagonala secundară K (/);
- C[K]=TRUE precizează că nici o regină nu amenință diagonala principală K (\).

Procedura recursivă de plasare a unei regine arată astfel:

```

PROCEDURA INCEARCA(I: INTEGER);
BEGIN
  {INITIALIZEAZA SELECTIA POZITIILOR PENTRU A I-A REGINA}
  REPEAT
    {SELECTEAZA POZITIA URMATOARE}
    IF SIGURA THEN
      BEGIN
        {PUNE REGINA}
        IF I<8 THEN
          BEGIN
            INCEARCA(I+1);
            IF NEREUSITA THEN {IA REGINA}
          END
        END
      END
    UNTIL REUSITA OR {NU MAI EXISTA POZITII}
  END;

```

Este clar că această procedură exprimată cu ajutorul comentariilor trebuie explicată cu ajutorul instrucțiunilor.

În programul REGINE (P.V.38), nu se vor reprezenta pozițiile reginelor pe tabla de șah, ci faptul că o regină a fost sau nu plasată de-a lungul fiecărei linii sau diagonale. Din regulile jocului de șah se știe că regina amenință coloana, linia și cele două diagonale care conțin câmpul pe care se află. Rezultă deci că fiecare coloană va putea conține o singură regină. În acest fel alegerea poziției celei de-a I-a regine poate fi restrinsă numai la coloana I. Parametrul I reprezintă numărul coloanei, iar procesul de selecție se restrânge la una din cele 8 valori posibile ale indicelui J care precizează linia.

Se știe că pe o tablă de șah există 15 diagonale principale (\) și 15 diagonale secundare (/) - câte una de lungime maximă, dintr-un colț în colțul opus, celelalte paralele cu acestea.

Caracteristica unei diagonale secundare (/) este aceea că suma coordonatelor I și J este constantă, iar a unei diagonale principale (\), că diferența este constantă.

Folosind aceste reprezentări și observații, comentariile din procedura INCEARCA se pot explica astfel:

```

{PUNE REGINA} în poziția (I,J) devine:
X[I]:=J;
A[J]:=FALSE;
B[I+J]:=FALSE;      {2<=I+J<=16}
C[I-J+7]:=FALSE;   {0<=I-J+7<=14}

```


(IA REGINA) devine:

```
A[J]:=TRUE;
B[I+J]:=TRUE;
C[I-J+7]:=TRUE;
```

Condiția SIGURA este îndeplinită dacă destinația (I,J) aparține unei linii și unor diagonale libere (TRUE), ceea ce poate fi exprimat astfel: A[J] AND B[I+J] AND C[I-J+7].

Pentru vizualizarea celor 92 de soluții pe monitor (în implementarea HP4TM), în program a mai fost folosit un tablou de caractere Y:ARRAY[1..8] OF CHAR, care are rolul de a memora câte o linie a tablei de șah. De asemenea pentru reprezentarea grafică a tablei de șah și a reginelor, a fost folosită procedura UDG.

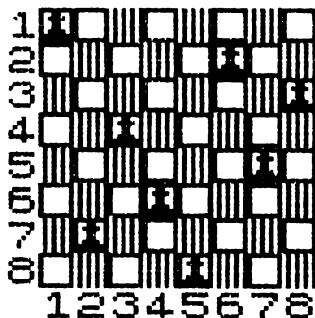
Programul afișează pe ecran toate cele 92 soluții, dar este de remarcat faptul că există doar 12 soluții distincte, celelalte putându-se obține prin rotirea tablei de șah.

```
B190 10 PROGRAM REGINE;
B190 20 VAR I,SOL:INTEGER;
B190 30 A:ARRAY[1..8] OF BOOLEAN;
B190 40 B:ARRAY[2..16] OF BOOLEAN;
B190 50 C:ARRAY[0..14] OF BOOLEAN;
B190 60 X:ARRAY[1..8] OF INTEGER;
B190 70 Y:ARRAY[1..9] OF CHAR;
B190 80
B190 90 PROCEDURE TIPARIRE;
B190 100 VAR I,K:INTEGER;
B190 110 C:CHAR;
B190 120 BEGIN
B190 130 WRITE(CHR(22),CHR(0),CHR(0));Writeln;
B190 140 SOL:=SOL+1;
B190 150 FOR K:=1 TO 8 DO
B190 160 BEGIN
B190 170 I:=1;
B190 180 WHILE I<8 DO
B190 190 BEGIN
B190 200 Y[I]:=CHR(144);
B190 210 Y[I+1]:=CHR(146);
B190 220 I:=I+2
B190 230 END;
B190 240 IF ODD(K)=TRUE THEN
B190 250 BEGIN
B190 260 C:=Y[2];
B190 270 FOR I:=8 DOWNTO 2 DO
B190 280 Y[I]:=Y[I-1];
B190 290 Y[1]:=C
B190 300 END;
B190 310 Y[X[K]]:=CHR(145);
B190 320 WRITE(K:1);
B190 330 FOR I:=1 TO 8 DO WRITE(Y[I]);
B190 340 Writeln
B190 350 END;
B190 360 WRITE(' 12345678');
B190 370 WRITE(CHR(22),CHR(7),CHR(13),'Solutia:',SOL);
B190 380 WRITE(CHR(22),CHR(21),CHR(10),'APASA O TASTA');
B190 390 REPEAT UNTIL INCH<>CHR(0);
B190 400 IF INCH IN ['S'] THEN
B190 410 TOUT('SCR-REG ',16384,6912)
B190 420 END;
B190 430
B190 440 PROCEDURE INCEARCA(I:INTEGER);
B190 450 VAR J:INTEGER;
B190 460 BEGIN
B190 470 FOR J:=1 TO 8 DO
B190 480 IF A[J] AND B[I+J] AND C[I-J+7] THEN
B190 490 BEGIN
B190 500 X[I]:=J;
B190 510 A[J]:=FALSE;B[I+J]:=FALSE;
B190 520 C[I-J+7]:=FALSE;
B190 530 IF I<8 THEN INCEARCA(I+1)
B190 540 ELSE TIPARIRE;
B190 550 A[J]:=TRUE;B[I+J]:=TRUE;
B190 560 C[I-J+7]:=TRUE
B190 570 END
B190 580 END;
```

```

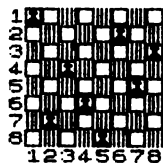
B6DF 590
B6DF 600 PROCEDURE UDG;
B6E2 610 BEGIN
B6FA 620 POKE(23275,23296);
B706 630 POKE(23296,255);
B712 640 FOR I:=0 TO 5 DO POKE(23297+I,129);
B749 650 POKE(23303,255);
B755 660 POKE(23304,255);
B761 670 POKE(23305,153);
B76D 680 POKE(23306,189);
B779 690 POKE(23307,153);
B785 700 POKE(23308,153);
B791 710 POKE(23309,189);
B79D 720 POKE(23310,255);
B7A9 730 POKE(23311,255);
B7B5 740 FOR I:=0 TO 7 DO POKE(23312+I,85)
B7E5 750 END;
B7F2 760
B7F2 770 BEGIN
B7FB 780 UDG:SOL:=0;
B806 790 FOR I:=1 TO 9 DO A(I):=TRUE;
B844 800 FOR I:=2 TO 16 DO B(I):=TRUE;
B882 810 FOR I:=0 TO 14 DO C(I):=TRUE;
B8C0 820 PAGE;
B8C5 830 INCEARCA(1)
B8C9 840 END (*P*).

```



Solutia: 2

APASA O TASTA



Solutia: 2

APASA O TASTA

E. Algoritmi care necesită recursivitate indirectă

Dacă o procedură se autoapelează, ea se numește *direct recursivă*, iar dacă această autoapelare se realizează prin intermediul altor proceduri, ea se numește *indirect recursivă*.

Programul SPATII (P.V.39) prelucrează un șir de caractere al cărui sfârșit este marcat de caracterul '\$'. Se cere eliminarea spațiilor suplimentare din acest șir, adică reducerea numărului de spații, care separă două cuvinte adiacente, la unul singur și suprimarea spațiilor inițiale și finale.

Fie A șirul (nevid) dat și B șirul rezultat, posibil identic cu A, dar care nu mai conține caracterul final '\$'.

Se vor utiliza trei proceduri și anume:

-procedura SPINIT, are ca scop eliminarea spațiilor inițiale din acest text; ea se autoapelează pînă la detectarea unui caracter diferit de spațiu sau '\$'; dacă caracterul care urmează nu este '\$', caracterul respectiv se depune în șirul B și se apelează procedura CUVINT care nu a fost încă definită; se impune deci declararea procedurii CUVINT utilizînd directiva FORWARD înainte de definirea procedurii SPINIT;

-procedura SPATIU tratează spațiile dintre cuvinte, reducînd numărul acestora la unul singur, pe care îl depune în șirul B; dacă următorul caracter tratat este diferit de spațiu sau '\$', se apelează procedura CUVINT declarată anterior cu FORWARD;

-procedura CUVINT tratează caracterele din cadrul unui cuvînt și le depune în șirul B; dacă caracterul tratat este spațiu se apelează procedura SPATIU; orice alt caracter (diferit de spațiu sau de '\$') va provoca autoapelarea procedurii CUVINT.

Se observă că fiecare procedură în parte este recursivă direct și procedurile împreună folosesc o recursivitate indirectă prin intermediul procedurii CUVINT.

Evident, problema poate fi rezolvată mult mai simplu nerecursiv, dar am ales această tehnică din motive didactice.

```

B0F5 10 PROGRAM SPATII;
B0F5 20 VAR I,J:0..79;
B0FE 30   A,B:ARRAY[0..79] OF CHAR;
B0FE 40
B0FE 50 PROCEDURE CUVINT;FORWARD;
B101 60
B101 70 PROCEDURE SPINIT;
B104 80   { elimina spatiile initiale }
B104 90 BEGIN
B11C 100   I:=I+1;
B123 110   IF A[I]=' '
B140 120     THEN SPINIT
B14A 130     ELSE WHILE A[I]<>'%' DO { caracter <> % ? }
B17F 140       BEGIN
B17F 150         J:=J+1;   { memoreaza }
B186 160         B[J]:=A[I]; { caracter }
B18A 170         CUVINT   { trateaza cuvint }
B18A 180       END
B1C3 190 END;
B1CC 200
B1CC 210 PROCEDURE SPATIU;
B1CF 220   { trateaza spatii inter-cuvinte }
B1CF 230 BEGIN
B1E7 240   I:=I+1;

```

```

B1EE 250   IF A[I]=
B20B 260   THEN SPATIU
B215 270   ELSE WHILE A[I]<>'%' DO
B24A 280     BEGIN
B24A 290       J:=J+1;B[J]:=' '; { memoreaza un spatiu}
B26E 300       J:=J+1;B[J]:=A[I]; { si primul caracter }
B2A9 310       CUVINT; { trateaza cuvint }
B2A9 320     END
B2B2 330   END;
B2BB 340
B2BB 350   PROCEDURE CUVINT;
B2BE 360   { memoreaza un cuvint }
B2BE 370   BEGIN
B2D6 380     I:=I+1;
B2DD 390     IF A[I]='%'
B2FA 400       THEN SPATIU
B304 410       ELSE WHILE A[I]<>'%' DO
B339 420         BEGIN
B339 430           J:=J+1;
B340 440           B[J]:=A[I];
B374 450           CUVINT
B374 460         END
B37D 470   END;
B386 480
B386 490   BEGIN
B38F 500     I:=0;J:=0;
B39B 510     { citire text }
B39B 520     REPEAT
B39B 530       READ(A[I]);
B3BC 540       I:=I+1
B3C4 550     UNTIL A[I-1]='%';
B3EB 560     { tratare text }
B3EB 570     I:=0;
B3F1 580     SPINIT;
B3F6 590     { scrie text tratat }
B3F6 600     I:=1;
B3FC 610     REPEAT
B3FC 620       WRITE(B[I]);
B41B 630       I:=I+1
B423 640     UNTIL I>J;
B435 650     WRITELN
B435 660   END {$P}.

```

P. V. 39.

Probleme propuse

1. Să se scrie o procedură care să realizeze împărțirea a două numere întregi utilizând doar operații de adunare și scădere.

2. Să se verifice și să se discute corectitudinea următoarelor declarații de proceduri și apelurile lor:

```

a) TYPE T=(BINE,RAU,POTRIVIT);
   VAR A: INTEGER;
       B: T;
   PROCEDURE P(A: T);
   BEGIN
     IF A=5
     THEN WRITELN('NU E BINE')
     ELSE WRITELN('E BINE')
   END;
   .....
   BEGIN
   .....
     A:=7;
     P(RAU);
   .....
   END.

```

```

b)  VAR X: INTEGER;
    PROCEDURE P(VAR A: REAL; B: REAL);
        BEGIN
            A:=B+5
        END;
    .....
    BEGIN
        .....
        P(X+7,X);
        .....
    END.

```

3. Să se realizeze un subprogram care să calculeze valoarea 2^{300} . Să se transforme, apoi subprogramul pentru calculul oricărei puteri de forma N^M , $N < 10$, M întreg pozitiv.

4. Să se realizeze o procedură PAUSE(N: INTEGER) care să aibă același efect, ca și instrucțiunea PAUSE din BASIC.

5. Să se realizeze o procedură care rotește cu 90° matrice pătratică.

6. Să se realizeze o procedură pentru formarea unei linii text dintr-o pagină astfel: procedura primește un cuvânt pe care îl adaugă la linia în curs de formare (cuvintele sînt separate printr-un spațiu); dacă prin adăugarea cuvintului lungimea liniei ar depăși 64 caractere, acesta urmează să fie primul cuvînt din linia următoare.

7. Aceeași problemă, dar tipărirea liniei să se facă după "aranjarea" (filarea) ei, prin adăugarea de spații între cuvinte, dacă este cazul, astfel încît ultimul cuvînt al liniei să fie aliniat la dreapta (ultimul caracter al liniei este al 64-lea).

8. Să se realizeze procedurile de bază ale unui editor de texte care lucrează asupra unei pagini de text formată din 32 de rînduri, fiecare rînd avînd 64 caractere:

- localizarea unui șir de caractere: LOC(X: SIR);
- inserarea unui șir de caractere: INS(X: SIR);
- ștergerea unui șir: STG(X: SIR);
- înlocuirea unui șir cu alt șir: INLC(X, Y: SIR);
- poziționarea la începutul unei linii: POZ(N: INTEGER).

9. Să se realizeze un program care să analizeze o pagină de text de 50 de rînduri, fiecare rînd avînd 64 caractere; programul va afișa toate cuvintele o singură dată și rîndurile pe care apar cuvintele respective.

10. Presupunînd că avem în memorie două seturi de caractere astfel reprezentate încît pe un caracter obișnuit Spectrum încap două asemenea caractere -"stînga" și "dreapta"- , să se realizeze un subprogram care să permită scrierea unui text cu astfel de caractere mici, deci pe un rînd al monitorului să poată fi scrise 64 de caractere.

11. a). Să se scrie subprograme care realizează diferite operații asupra valorilor complexe: adunare, scădere, înmulțire, împărțire, modul, argument, citire, scriere.

b). Să se folosească procedurile realizate la punctul a) pentru a realiza un program care să citească un șir de numere

complexe, să calculeze produsul lor P și suma S precum și expresia:

$$E = \left(\frac{P - S}{P * S} \right)^n$$

Calculul se va face în două moduri: direct și folosind formula lui Moivre. Se vor tipări rezultatele obținute prin ambele metode.

12. Să se modifice programul LABIRINT (P.V.37), adăugându-i-se o procedură pentru generarea chiar în program a unui labirint.

13. Să se realizeze un subprogram de tip funcție pentru ridicarea unui număr întreg X la o putere naturală N .

14. Să se realizeze o funcție PUTERE pentru calculul expresiei X^Y , unde X și Y sînt de tip REAL.

15. Se cere o funcție POINT, care să realizeze aceeași operație ca aceea din BASIC.

16. Să se realizeze o funcție CIFRA(N,M) care returnează valoarea celei de a M -a cifre, de la dreapta spre stînga, a numărului N , scris în sistemul zecimal.

Exemplu: CIFRA(7283,3)=2.

17. Să se realizeze funcția recursivă care calculează valorile funcției lui Ackermann definită pentru $m \geq 0$ și $n \geq 0$ astfel:

$$\begin{aligned} \text{Ack}(0, n) &= n + 1 \\ \text{Ack}(m, 0) &= \text{Ack}(m - 1, 1) && \text{pentru } m > 0 \\ \text{Ack}(m, n) &= \text{Ack}(m - 1, \text{Ack}(m, n - 1)) && \text{pentru } m > 0 \text{ și } n > 0. \end{aligned}$$

18. Să se realizeze funcția recursivă pentru calculul valorilor polinoamelor lui Hermite, $H(x)$, știind că:

$$\begin{aligned} H_0(x) &= 1 \\ H_1(x) &= 2x \\ H_n(x) &= 2H_{n-1}(x) - 2(n-1)H_{n-2}(x) && \text{pentru } n > 1 \end{aligned}$$

Să se realizeze și o funcție iterativă și să se compare cele două metode de calcul.

19. Dîndu-se un șir de funcții și variabile sub forma unui șir de caractere și aritatea fiecărei funcții, să se modifice șirul inițial, folosind apelul recursiv, punîndu-se la locul lor parantezele și virgulele.

Exemplu:

-șirul inițial: FXGXHZ
-aritățile: ar(F)=3
ar(G)=2
ar(H)=1

- se va obține: F(X,G(X,Y),H(Z)).

(Prin aritatea unei funcții înțelegem numărul argumentelor acelei funcții).

VI. ALOCAREA DINAMICA A MEMORIEI

In limbajul PASCAL se poate lucra cu
 -structuri de date statice și cu
 -structuri dinamice.

Structurilor statice

- li se alocă spațiu pe parcursul compilării și
- pot fi referite prin nume.

Structurile dinamice

- nu vor avea nume și
- nu li se alocă spațiu decât în timpul execuției
- în plus, acest spațiu se poate elibera la cererea explicită a programatorului și astfel, se poate reda sistemului de gestiune a memoriei.

6.1. TIPUL REFERINTA (POINTER)

Structurile dinamice, neavând nume, sînt accesibile prin intermediul altor variabile, care conțin adresele lor, și în felul acesta "arată spre ele", "le indică". Aceste variabile se numesc *pointeri* și formează *tipul referință* (tipul pointer sau tipul reper). *Variabilele de tip pointer sînt variabile statice obișnuite și au ca valori adrese*, care pot fi adresele de început ale unor structuri dinamice sau a unei componente a unei asemenea structuri. Prin urmare accesul la o variabilă dinamică, indicată de o variabilă pointer se realizează prin *adresarea indirectă* a variabilei pointer. Această indirectare din punct de vedere formal se pune în evidență prin atașarea caracterului '^' după numele variabilei pointer.

Variabila dinamică, indicată de o variabilă pointer, se va referi, nu printr-un nume propriu, ci prin intermediul pointerului său: *variabilă de tip pointer^*.

Exemplu:

- P - variabilă pointer,
- P^ - variabilă dinamică indicată de pointerul P.

In PASCAL există o legătură fixă între o variabilă pointer și tipul variabilelor indicate. O variabilă pointer dată se poate referi la variabile indicate de un anumit tip, conform definiției sale.

Pointerul nul se identifică prin cuvîntul rezervat *NIL*; el este compatibil cu orice tip pointer.

Un tip pointer se definește precizînd tipul variabilei indicate, precedat de caracterul '^'

Exemple:

- a) TYPE TIPPOINTER=^T;

```

b) TYPE COMPLEX=RECORD
      REEL,IMAG:REAL
      END;
VAR P:^COMPLEX;

```

Variabila dinamică din exemplul anterior poate fi vizualizată ca în fig 6.1.

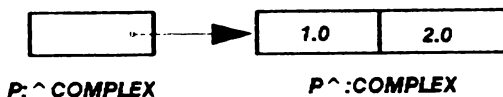


fig.6.1

Se observă că simbolul '^' poate precede un identificator de tip, definind astfel un tip pointer, sau poate urma un identificator de variabilă pointer, precizând în acest caz o variabilă indicată.

Menționăm că în limbajul *PASCAL standard* și în majoritatea implementărilor sale este permisă și o secvență de declarații de forma:

```

TYPE TIPPOINTER=^T;
T=...

```

adică o referire a identificatorului *T* înainte ca el să fie definit.

Alocarea memoriei pentru o structură dinamică se realizează cu ajutorul procedurii *NEW* care are un singur parametru de tip pointer.

Exemplu:

Fie *P* o variabilă de tip pointer. Efectul apelului de procedură *NEW(P)* este de a rezerva o zonă de memorie pentru o variabilă indicată *P^* și de a inițializa variabila pointer *P* cu adresa zonei rezervate. Menționăm că *NEW(P)* nu realizează și inițializarea variabilei dinamice indicate de *P*.

Pentru exemplul anterior, efectul instrucțiunii *NEW(P)* este de a rezerva o zonă de memorie necesară pentru o variabilă de tip *COMPLEX*, (fără a înregistra valori în această zonă) și de a inițializa variabila *P* cu adresa zonei rezervate. Prin urmare, această zonă se poate indica (referi) prin *P^*.

Observații:

1)-mai multe variabile pointer pot referi aceeași variabilă indicată conform fig.6.2.

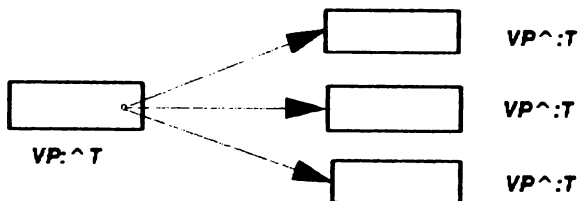


fig.6.2

2)-mai multe variabile indicate pot fi referite de aceeași variabilă pointer conform fig.6.3). Situația din figură se poate realiza în timp; un pointer indică la un moment dat o singură variabilă dinamică, dar schimbându-și valoarea, își va schimba și indicația.

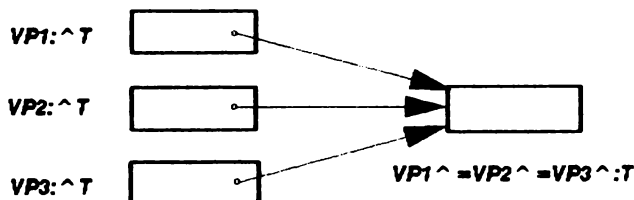


fig. 6.3

Programul POINT1 (P.VI.1) exemplifică (în liniile 150-200) modul în care se poate introduce și afișa un număr complex cu structură articol (RECORD). În liniile 210-260 se generează același număr complex ca variabilă dinamică, iar în liniile 290-350 se generează zece numere complexe prin intermediul aceleiași variabile de tip pointer.

```

0000 20 (INTRODUCERE,AFISARE NR.COMPLEXE,GENERARE,AFISARE NR.COMPLEXE)
0000 30 PROGRAM POINT1;
0000 40 CONST P=CHR(10);
0000 50 TYPE COMPLEX=RECORD
0000 60 RE,IM:REAL
0000 70 END;
0000 80 VAR Z:COMPLEX;
0009 90 PZ:^COMPLEX;
0009 100 C:CHAR;
0009 110 N,I:INTEGER;
0009 120 BEGIN
0012 130 PAGE;
0017 140 WRITELN('DATI NR.COMPLEX:');
0035 150 READ(Z.RE,Z.IM);
0057 160 IF Z.IM<0 THEN
0079 170 C:='-';
0070 180 ELSE
0081 190 C:='+';
0088 200 WRITELN('Z=',Z.RE:5:2,C,ABS(Z.IM):5:2,'I');
00D3 210 NEW(PZ);
00DC 220 PZ^.RE:=Z.RE;
00F5 230 PZ^.IM:=Z.IM;
0112 240 WRITELN(P);
011A 250 WRITELN('PRIN POINTER:');
0135 260 WRITELN('Z=',PZ^.RE:5:2,C,ABS(PZ^.IM):5:2,'I');
0186 270 { O SINGURA VARIABILA POINTER}
0186 280 { GENEREAZA MAI MULTE VARIABILE INDICATE:}
0186 290 FOR I:=1 TO 10 DO
01A0 300 BEGIN
01A3 310 NEW(PZ);
01AC 320 PZ^.RE:=I;
01C0 330 PZ^.IM:=2*I;
01E3 340 WRITELN('Z',I,'=',PZ^.RE:2:0,'+',PZ^.IM:2:0,'I')
0234 350 END;
023A 360 WRITELN(P);
0242 370 END {$P}.

```

```

PRIN POINTER:
Z= 5.65- 7.83I
Z1 = 1+ 2I
Z2 = 2+ 4I
Z3 = 3+ 6I
Z4 = 4+ 8I
Z5 = 5+10I
Z6 = 6+12I
Z7 = 7+14I
Z8 = 8+16I
Z9 = 9+18I
Z10 =10+20I

```

6.2. STRUCTURI DE DATE DE TIP LISTA LINIARA

Lista este o structură dinamică în care toate elementele sînt de același tip, sînt concomitent în memorie și între ele există o relație de ordine. În timpul execuției numărul elementelor variază, poate fi chiar nul.

Listele pot crește sau descrește (pot fi create începînd cu primul element pînă la ultimul sau invers, de la ultimul spre primul) elementele lor putînd fi referite, inserate sau șterse în/din orice poziție din cadrul listei.

Listele pot fi concatenate sau scindate în sublistele.

Lista se mai poate defini ca fiind o secvență de mai multe elemente (noduri) de un anumit tip, numit tip de bază (T_B)

Pentru $n=0$ lista este vidă.

$a_1, a_2, \dots, a_n, n \geq 0, a_i: T_B$

Pentru $n \geq 1, a_1$ = primul nod al listei;

a_n = ultimul nod al listei.

Proprietăți:

-nodurile pot fi ordonate liniar în funcție de poziția lor în cadrul listei:

- a_i precede pe $a_{i+1}, i=1,2,\dots,n-1;$

- a_i succede pe $a_{i-1}, i=2,3,\dots,n;$

-nodul i se află pe poziția a_i .

Observație:

Lista liniară în principiu este o structură dinamică de date, dar ea nu poate fi implementată dinamic decît în limbajele de programare care posedă mecanismul de alocare dinamică.

6.2.1. Implementarea listelor cu ajutorul tipului pointer (liste înlănțuite)

Utilitatea tipului pointer devine evidentă în cazul structurilor de date dinamice înlănțuite, structuri care în general se definesc recursiv.

Înlănțuirea se pune în evidență cu ajutorul unor informații de legătură care la rîndul lor pot fi memorate în unul sau mai multe cîmpuri ale articolului care conține informațiile atașate unui nod (element) al listei. Evident, informațiile propriu-zise atașate nodurilor pot fi memorate de asemenea în unul sau mai multe cîmpuri.

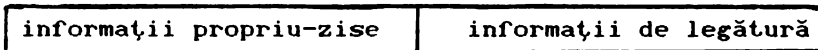


fig. 6.4

O listă liniară poate fi definită în termeni recursivi în limbajul PASCAL standard astfel:

```

TYPE LISTA = ^NOD;
      NOD = RECORD
              CHEIE: INTEGER;
              INFO: ...;
              URM: LISTA
            END;

```

Se observă că identificatorul NOD este referit înainte de a fi definit. Deoarece acest lucru este nepermis în HP4TM, definirea listei liniare în această implementare se face în felul următor:

```

TYPE NOD=RECORD
    CHEIE: INTEGER;
    INFO: ...;
    URM: ^NOD;
END;
LISTA=^NOD;

```

Orice nod din lista definită astfel are trei câmpuri:
 -o cheie pentru identificarea nodului (CHEIE) care nu este obligatorie;
 -o zonă pentru informația utilă (INFO) care poate diferi ca structură și tip în funcție de problemă;
 -un pointer de înlănțuire la nodul următor (URM).

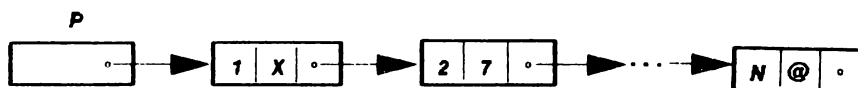


fig. 6.5

6.2.2. Tehnici de inserare a nodurilor

a. Inserare la începutul listei

Vom utiliza următoarele notații:

P-variabilă pointer care indică primul nod al listei, necesară deoarece fiecare nod al listei se inserează la începutul ei;

Q-variabilă de lucru de tip pointer.

Secvența de inserare este următoarea:

```

NEW(Q);
Q^.URM:=P;
P:=Q;

```

care trebuie completată cu inițializarea câmpurilor informației propriu-zise.

Efectul primei instrucțiuni este alocarea spațiului necesar câmpurilor noului nod și inițializarea pointerului Q cu adresa acestui nod. În urma celei de-a doua instrucțiuni se realizează legătura dintre noul nod și primul nod al listei (câmpul de tip pointer al noului nod va indica vechiul nod din listă). Operațiile descrise sînt vizualizate în fig.6.6.

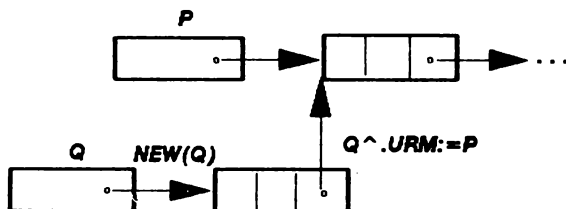


fig. 6.6

În urma celei de-a treia instrucțiuni se modifică conținutul pointerului P astfel încît el să indice nodul inserat (fig.6.7).

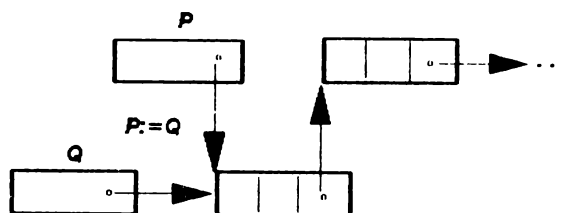


fig. 6.7

Având în vedere că o listă se construiește inserind elementele în lista inițial vidă, secvența de program PASCAL pentru construirea unei liste este următoarea:

```

P:=NIL;
WHILE N>0 DO      (N REPREZINTA NR. ELEMENTELOR DIN LISTA)
  BEGIN
    NEW(Q);
    Q^.URM:=P;
    P:=Q;
    Q^.CHEIE:=N;
    N:=N-1
  END;

```

Observație:

În secvența de mai sus, instrucțiunea WHILE se poate înlocui cu FOR, dar deoarece nu întotdeauna se cunoaște numărul elementelor (întrucât ele se selectează pe baza anumitor criterii), instrucțiunea WHILE este mai adecvată.

Programul LISTAPOINT (P.VI.2) creează o listă prin tehnica inserării la începutul listei și o afișează, traversând-o (traversarea se va trata în subparagraful 6.2.4).

```

B0C4 20 ( INSERTIE NODURI LA INCEPUTUL UNEI LISTE SI AFISARE )
B0C4 30 PROGRAM LISTAPOINT1;
B0C4 40 TYPE TAB=ARRAY[1..20]OF CHAR;
B0C4 50     NOD=RECORD
B0C4 60     CHEIE:INTEGER;
B0C4 70     NUME:TAB;
B0C4 80     URM:^NOD
B0C4 90     END;
B0C4 100 VAR P,Q:^NOD;
B0CD 110     N,I:INTEGER;
B0CD 120     NUM:TAB;
B0CD 130 PROCEDURE CITNUM;
B0D0 140 BEGIN
B0E2 150   WRITE('DATI NUMELE:');
B0FF 160   READLN;READ(NUM);
B10A 170 END; { CITNUM }
B110 180 PROCEDURE PREGTIP;
B113 190 BEGIN
B12B 200   FOR I:=20 DOWNT0 1 DO
B144 210     IF NUM[I]=CHR(0)
B165 220       THEN NUM[I]:=CHR(32)
B18D 230 END; { PREGTIP }
B198 240 PROCEDURE INSEREAZAINFATA;
B19B 250 BEGIN
B1B3 260   WHILE N>0 DO
B1C9 270     BEGIN
B1C9 280       NEW(Q);
B1D2 290       Q^.URM:=P;
B1E2 300       P:=Q;
B1E8 310       Q^.CHEIE:=N;
B1F4 320       CITNUM;
B1FD 330       PREGTIP;
B206 340       Q^.NUME:=NUM;
B215 350       N:=N-1;
B21D 360     END
B21C 370 END; { INSEREAZAINFATA }

```

```

B225 380 PROCEDURE TRAVLISTA;
B228 390 BEGIN
B240 400 Q:=P;
B246 410 WHILE Q<>NIL DO
B252 420 BEGIN
B255 430 WRITELN;
B25E 440 WITH Q^ DO
B269 450 WRITELN(CHEIE,' ',NUME);
B278 460 Q:=Q^.URM
B298 470 END
B29E 480 END; ( TRAVLISTA )
B2A7 490 BEGIN ( MAIN )
B2B0 500 WRITE('DATI NR.DE NODURI:');
B2CD 510 READ(N);
B2D3 520 P:=NIL;
B2D9 530 INSEREAZAINFATA;
B22E 540 WRITELN(CHR(16));
B2E8 550 WRITELN;
B2EB 560 WRITE('LISTA A FOST CREATA SI CONTINE NODURILE:');
B31E 570 TRAVLISTA;
B323 580 WRITE(CHR(16))
B32A 590 END {$P}.

```

LISTA A FOST CREATA SI CONTINE NODURILE:

- 1 PACURARU MIHAI
- 2 SPERNEAC PAUL
- 3 RADULESCU CORNELIU
- 4 HEREDI CARINA
- 5 SERBAN MIHAELA
- 6 TUDOR MIRON

P.VI.2

b. Inserare la sfîrşitul listei

Vom utiliza următoarele notații:

P-variabilă pointer care indică primul nod, necesară pentru traversarea listei;

Q-variabilă pointer care indică ultimul nod al listei, necesară deoarece fiecare nod se inserează la sfîrşitul ei

R-variabilă de lucru de tip pointer.

Inserarea unui element la sfîrşitul listei se realizează cu următoarea secvență de program PASCAL:

```

NEW(R);
Q^.URM:=R;
R^.URM:=NIL;
Q:=R;

```

Inainte de inserare lista are forma din fig.6.8.

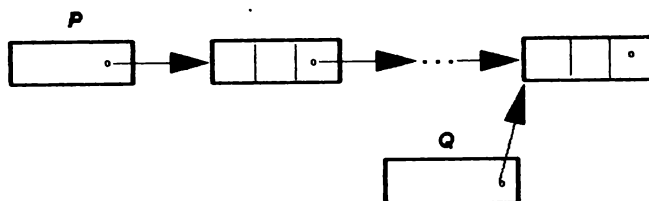


fig.6.8

In urma execuției primelor trei instrucțiuni din secvența anterioară, lista are configurația din fig.6.9.

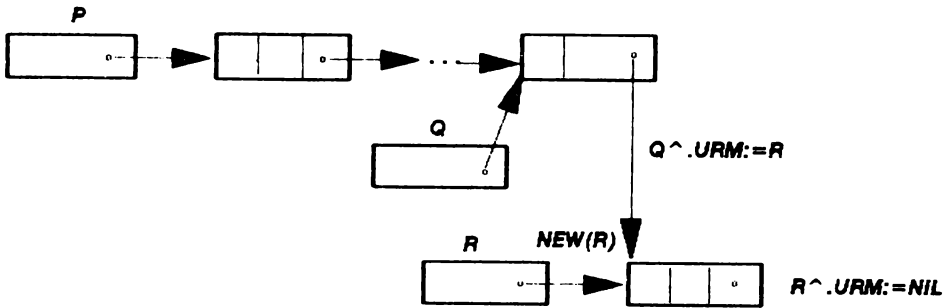


fig.6.9

Pentru a putea introduce secvența de inserare într-o structură repetitivă, pointerul Q trebuie să indice mereu ultimul element din listă, deci variabila Q trebuie actualizată (instrucțiunea a patra). In final, lista va avea structura din fig.6.10.

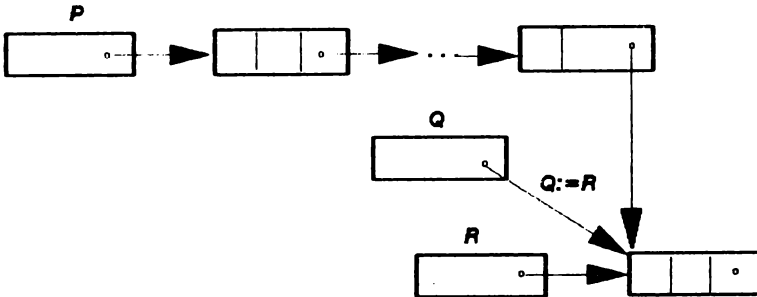


fig.6.10

Observație:

Deoarece într-o listă vidă $Q^{\wedge}.URM$ nu există, primul nod trebuie inserat într-un alt mod, prezentat în programul P.VI.3.

Programul LISTAPOINT2 (P.VI.3) creează o listă prin tehnica inserării la sfârșitul listei.

```

B170 20 { INSERTIE NODURI IN SPATELE UNEI LISTE SI AFISARE }
B170 30 PROGRAM LISTAPOINT2;
B170 40 TYPE TAB=ARRAY[1..20]OF CHAR;
B170 50     NOD=RECORD
B170 60         CHEIE: INTEGER;
B170 70         NUME: TAB;
B170 80         URM: ^NOD
B170 90     END;
B170 100 VAR P,Q,R: ^NOD;
B179 110     N,I,J: INTEGER;
B179 120     NUM: TAB;
B179 130 PROCEDURE CITNUM;
B17C 140 BEGIN
B194 150     WRITE('DATI NUMELE: ');
B1AB 160     READLN; READ(NUM);
B1B6 170 END; { CITNUM }
B1BC 180 PROCEDURE PREGTIP;
B1BF 190 BEGIN
B1D7 200     FOR I:=20 DOWNT0 1 DO
B1F0 210         IF NUM[I]=CHR(0)
B211 220             THEN NUM[I]:=CHR(32)
B239 230 END; { PREGTIP }

```

```

B244 240 PROCEDURE TRAVLISTA;
B247 250 BEGIN
B25F 260   Q:=P;
B265 270   WHILE Q<>NIL DO
B27A 280     BEGIN
B27A 290       WRITELN;
B27D 300       WITH Q^ DO
B288 310         WRITELN(CHEIE,' ',NUME);
B2AF 320         Q:=Q^.URM
B2B7 330     END
B2BD 340 END; ( TRAVLISTA )
B2C6 350 PROCEDURE INSEREAZAPRIMUL;
B2C9 360 BEGIN
B2E1 370   P:=NIL;
B2E7 380   NEW(Q);
B2F0 390   WRITE('DATI CHEIA:');
B306 400   READ(Q^.CHEIE);
B312 410   P:=Q;
B318 420   CITNUM;
B321 430   PRESTIP;
B32A 440   Q^.NUME:=NUM
B334 450 END; ( INSEREAZAPRIMUL )
B33F 460 PROCEDURE INSEREAZAREST;
B342 470 BEGIN
B35A 480   NEW(R);
B363 490   R^.URM:=NIL;
B373 500   Q^.URM:=R;
B383 510   WRITE('DATI CHEIA:');
B399 520   READ(R^.CHEIE);
B3A5 530   CITNUM;
B3AE 540   PREGTIP;
B3E7 550   R^.NUME:=NUM;
B3C6 560   Q:=R
B3C6 570 END; ( INSEREAZAREST )
B3D2 580 BEGIN ( MAIN )
B3D8 590   WRITE('DATI NR.DE NODURI:');
B3F8 600   READ(N);
B3FE 610   INSEREAZAPRIMUL;
B403 620   FOR J:=2 TO N DO
B421 630     INSEREAZAREST;
B42E 640   WRITELN(CHR(16));
B437 650   WRITELN('LISTA ESTE URMATOAREA:');
B458 660   TRAVLISTA;
B460 670   WRITELN(CHR(16))
B467 680 END (3F).

```

LISTA ESTE URMATOAREA:

```

10   POP ION
20   IGONICA FLORIN
25   COSTESCU MARIUS
40   GHITA IOSIF
66   DARIE IOANA

```

P. VI.3

c. Inserare într-un loc oarecare al listei

Notăm cu:

P-un pointer care indică nodul listei după care se dorește inserarea;
 Q-o variabilă de lucru de tip pointer.

Inserarea unui nod *după* nodul P[^] se realizează cu următoarea secvență de program PASCAL:

```

NEW(Q);
Q^.URM:=P^.URM;
P^.URM:=Q;

```

Inainte de inserare, lista are structura din fig.6.11

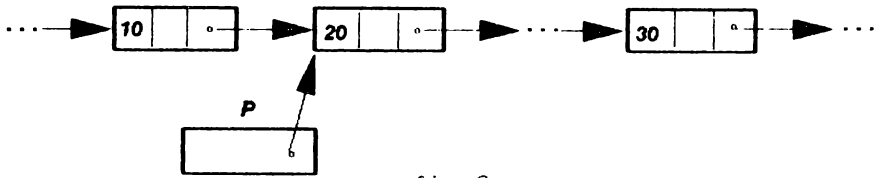


fig. 6.11

În urma secvenței anterioare de instrucțiuni, lista va avea configurația din fig. 6.12.

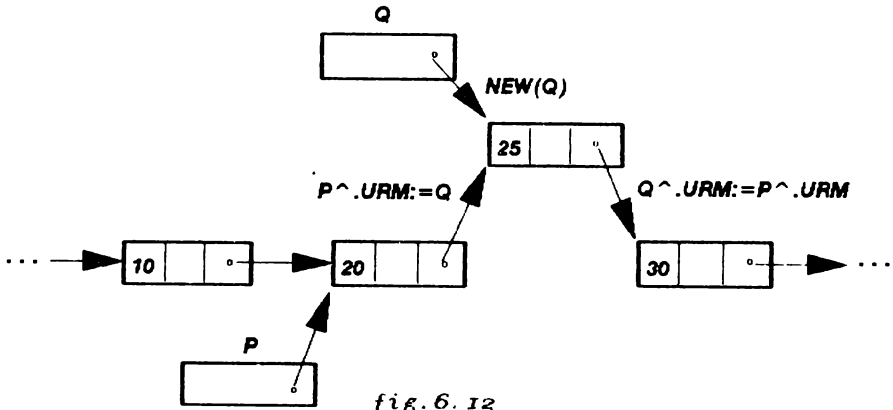


fig. 6.12

Observație:

Dacă se dorește inserarea unui nod înaintea unui nod indicat P^{\wedge} , neavînd nici un pointer care să indice predecesorul lui P^{\wedge} , inserarea se va realiza prin următoarea tehnică:

- se inserează un nod după P^{\wedge} ;
- se inițializează acest nod cu P^{\wedge} , creîndu-se astfel o "dublură" a variabilei indicate de pointerul P ;
- cîmpurile CHEIE și INFO din vechiul nod P^{\wedge} se actualizează corespunzător noului nod.

Secvența de program PASCAL care realizează o inserare în acest fel este următoarea:

```
NEW(Q);
Q^:=P^;
P^.URM:=Q;
P^.CHEIE:=CHEIE;
P^.INFO:=INFO;
```

Efectul primelor două instrucțiuni se poate urmări în fig. 6.13.

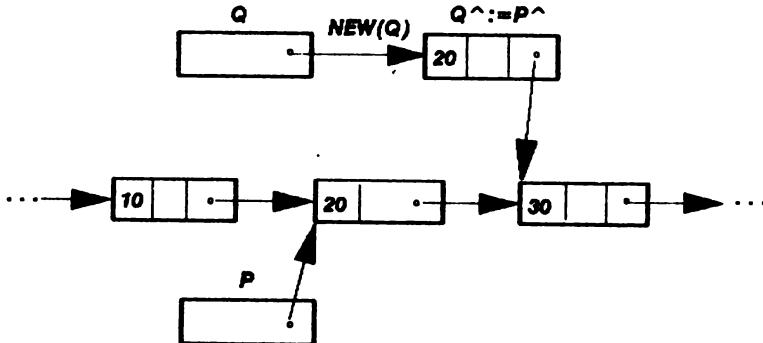


fig. 6.13

După crearea acestei "dubluri" a lui P^{\wedge} urmează legarea lui după vechiul P^{\wedge} , conform celei de-a treia instrucțiuni, așa cum rezultă din fig.6.14.

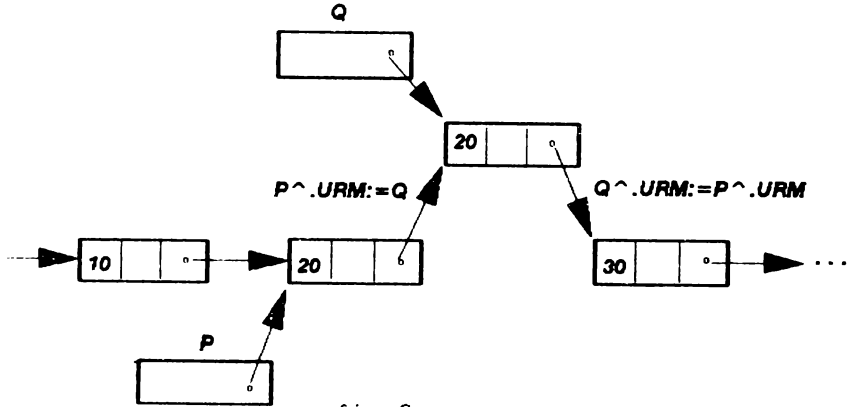


fig.6.14

Astfel, s-a pregătit loc pentru noul nod al listei în vechiul P^{\wedge} ale cărui cimpuri se vor inițializa cu informațiile nodului nou (fig.6.15).

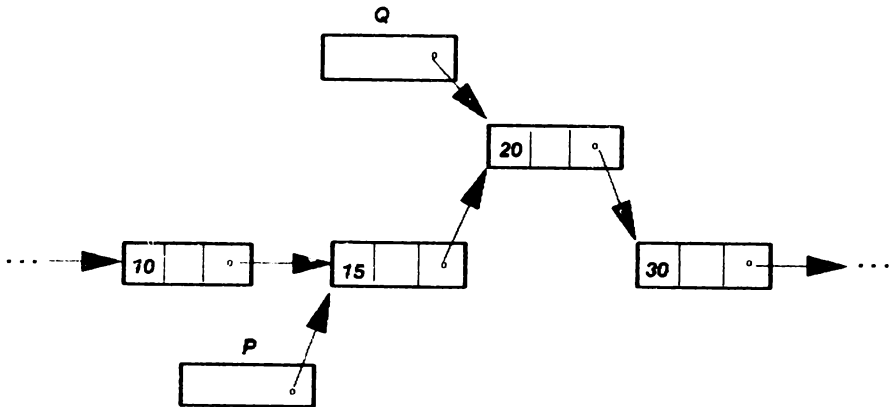


fig.6.15

6.2.3. Tehnici de suprimare (ștergere) a nodurilor unei liste

Considerăm o listă (fig.6.16); se pune problema ștergerii unui nod al acesteia. Problema va avea rezolvări diferite în funcție de poziția nodului relativ la un pointer cunoscut P .

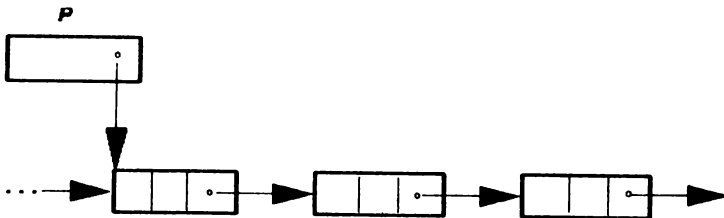


fig.6.16

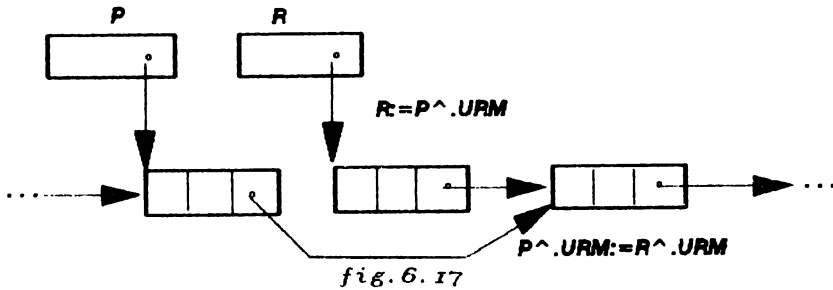
1)-dacă nodul pe care dorim să-l suprimăm este succesorul nodului P^{\wedge} , și în plus dorim să păstrăm posibilitatea de a ajunge la el, secvența instrucțiunilor care realizează ștergerea este următoarea:

```
R:=P^.URM;
P^.URM:=R^.URM;
```

Așa cum rezultă din fig.6.17, prin pointerul de lucru R se poate ajunge la informațiile atașate nodului eliminat din structură, dacă acestea mai sînt necesare.

Eliberarea fizică a spațiului de memorie alocat variabilelor dinamice se realizează cu procedurile standard MARK și RELEASE, avînd același parametru actual de tip pointer.

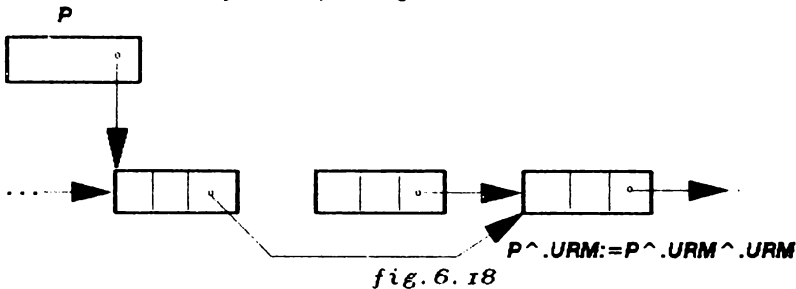
Toată memoria alocată variabilelor dinamice în zona de program cuprinsă între MARK și RELEASE se eliberează după execuția procedurii RELEASE, putînd fi realocată altor variabile dinamice. Deoarece sînt distruse toate variabilele dinamice create după executarea procedurii MARK, procedurile MARK și RELEASE se vor folosi cu precauție, mai ales dacă se lucrează simultan cu mai multe structuri dinamice de date.



Dacă ștergerea se realizează cu instrucțiunea:

```
P^.URM:=P^.URM^.URM
```

informațiile din nodul eliminat se pierd deoarece nici un pointer nu indică nodul suprimat, (fig.6.18).



2)-dacă dorim ștergerea nodului indicat de P, se aduce succesorul lui P^{\wedge} în locul elementului indicat de P, apoi se șterge vechiul nod; aceasta se poate realiza printr-o singură instrucțiune și anume:

```
P^:=P^.URM^;
```

Pentru exemplificare, să considerăm o listă cu structura din fig.6.19.

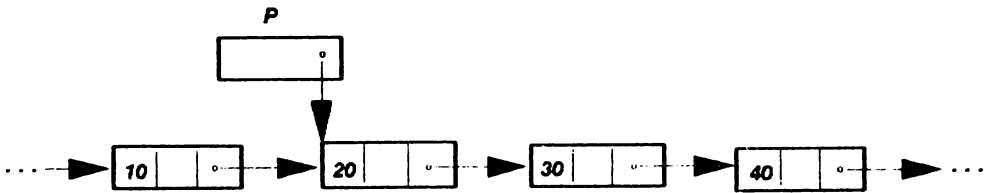


fig. 6.19

Să presupunem că dorim să ștergem elementul cu cheia 20. Efectul instrucțiunii de mai sus se poate urmări în fig. 6.20.

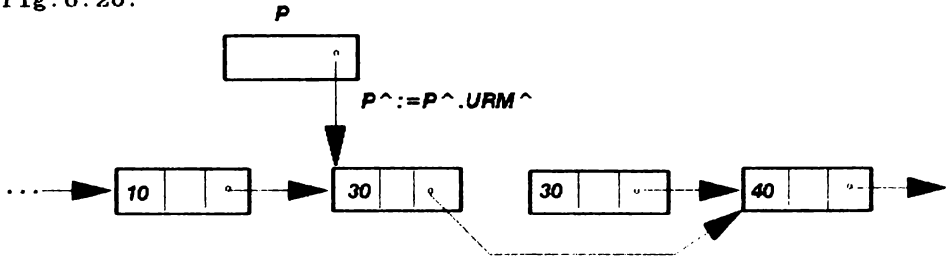


fig. 6.20

Dacă trebuie păstrată posibilitatea utilizării informațiilor propriu-zise atașate nodului suprimat, ștergerea se va realiza cu instrucțiunile:

```
R:=P^.URM;
P^:=R^;
```

Dacă lista are configurația din fig.6.19, în urma execuției primelor două instrucțiuni din secvența de mai sus structura listei va fi cea din fig.6.21.

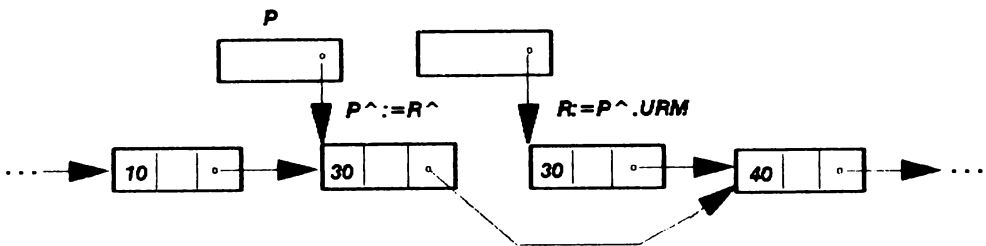


fig. 6.21

Observație:

Aceste tehnici se pot aplica numai dacă P^ nu este ultimul nod al listei, deoarece în acest caz nu există variabilă dinamică indicată de pointerul P^.URM, el avînd valoarea NIL.

6.2.4. Traversarea unei liste înlănțuite

A traversa o listă înlănțuită înseamnă a trece de la nod la nod în vederea executării anumitor operații asupra tuturor nodurilor listei.

Pentru traversare se vor folosi doi pointeri:

- P indică primul nod al listei;
- Q un pointer de lucru.

Dacă $Q^$ este un nod oarecare al listei, vom nota cu $O(Q^)$ o operație a cărei natură în contextul de față nu este nevoie să o precizăm.

Secvența de traversare a listei este următoarea:

```
Q:=P;
WHILE Q<>NIL DO
  BEGIN
    (O(Q^);)
    Q:=Q^.URM
  END;
```

Una din operațiile cele mai frecvente este căutarea, adică depistarea unui nod având cheia egală cu o valoare dată, X. (Prin cheie se înțelege oricare cîmp al informației propriu-zise.)

Secvența de căutare este:

```
Q:=P;
WHILE (Q<>NIL) AND (Q^.CHEIE<>X) DO
  Q:=Q^.URM;
IF Q=NIL THEN
  WRITE('NOD INEXISTENT')
ELSE
  WRITE('PRIMUL NOD CU CHEIA DORITA=',Q^.INFO);
```

Observație:

Dacă $Q=NIL$ atunci nodul $Q^$ nu există, și în acest caz o eventuală evaluare (dependentă de implementare) a subexpresiei $(Q^.CHEIE<>X)$ ar genera o eroare. O variantă corectă independent de implementare este următoarea:

```
B:=FALSE;
Q:=P;
WHILE (Q<>NIL) AND NOT B DO
  IF Q^.CHEIE=X
  THEN B:=TRUE
  ELSE Q:=Q^.URM;
```

Programul LISTAPOINT2 (P.VI.4) exemplifică operațiile executabile asupra listelor, prezentate în acest subparagraf.

```

888E 20 ( CREAZA PRIN INSERARE IN FATA SAU IN SPATE,AFISEAZA,CAUTA,SUPRI
MA )
888E 30 PROGRAM LISTAPOINT2;
888E 40 TYPE TAB=ARRAY[1..20]OF CHAR;
888E 50     NOD=RECORD
888E 60     CHEIE:INTEGER;
888E 70     NUME:TAB;
888E 80     URM:^NOD
888E 90     END;
888E 100 VAR P,Q,R:^NOD;
88C7 110     N,M,I,J,K,X:INTEGER;
88C7 120     NUM:TAB;
88C7 130     B:BOOLEAN;
88C7 140 PROCEDURE CITNUM;
88CA 150 BEGIN
88E2 160     WRITE('DATI NUMELE:');
88F9 170     READLN;READ(NUM);
8904 180 END; { CITNUM }
890A 190 PROCEDURE PREGTIP;
890D 200 BEGIN
8925 210     FOR I:=20 DOWNT0 1 DO
893E 220         IF NUM[I]=CHR(0)
895F 230             THEN NUM[I]:=CHR(32)
8987 240 END; { PREGTIP }
```

```

B992 250 PROCEDURE TRAVLISTA;
B995 260 BEGIN
B9AD 270 Q:=P;
B9B3 280 WHILE Q<>NIL DO
B9C8 290 BEGIN
B9C8 300 WRITELN;
B9C8 310 WITH Q^ DO
B9D6 320 WRITELN(CHEIE, ' ',NUME);
B9FD 330 Q:=Q^.URM
BA05 340 END
BA08 350 END; ( TRAVLISTA )
BA14 360 PROCEDURE INSEREAZAPRIMUL;
BA17 370 BEGIN
BA2F 380 P:=NIL;
BA35 390 NEW(C);
BA3E 400 WRITE('DATI CHEIA:');
BA54 410 READ(Q^.CHEIE);
BA60 420 P:=Q;
BA66 430 CITNUM;
BA6F 440 PRESTIF;
BA78 450 Q^.NUME:=NUM
BA82 460 END; ( INSEREAZAPRIMUL )
BA8D 470 PROCEDURE INSEREAZAREST;
BA90 480 BEGIN
BAA8 490 NEW(R);
BAB1 500 R^.URM:=NIL;
BAC1 510 Q^.URM:=R;
BAD1 520 WRITE('DATI CHEIA:');
BAE7 530 READ(R^.CHEIE);
BAF3 540 CITNUM;
BAFC 550 PRESTIF;
BB05 560 R^.NUME:=NUM;
BB14 570 Q:=R;
BB14 580 END; ( INSEREAZAREST )
BB20 590 PROCEDURE MENU;
BB23 600 BEGIN
BB3B 610 WRITELN('1. CREAARE LISTA');
BB58 620 WRITELN('2. AFISARE LISTA');
BB76 630 WRITELN('3. INSERARE DUFA UN NOD');
BB98 640 WRITELN('4. INSERARE INAINTEA UNUI NOD');
BBC6 650 WRITELN('5. CAUTARE NOD DUFA CHEIE');
BBED 660 WRITELN('6. SUPRIMARE NOD');
BC0E 670 WRITELN('7. STOP');
BC20 680 END; ( MENU )
BC26 690 PROCEDURE CREAARE;
BC29 700 BEGIN
BC41 710 WRITE('DATI NR.DE NODURI:');
BC5E 720 READ(N);
BC64 730 INSEREAZAPRIMUL;
BC6E 740 IF N#1 THEN
BC80 750 INSEREAZAREST;
BC89 760 END; ( CREAARE )
BC8F 770 PROCEDURE STAI;
BC92 780 BEGIN
BCAA 790 REPEAT UNTIL INCH<>CHR(0)
BCB5 800 END; ( STAI )
BCC5 810 PROCEDURE CAUTA(VAR B:BOOLEAN);
BCC8 820 BEGIN
BCE0 830 B:=FALSE;
BCEA 840 Q:=P;
BCF0 850 WHILE (Q<>NIL) AND NOT B DO
BD11 860 IF Q^.CHEIE=X
BD19 870 THEN B:=TRUE
BD31 880 ELSE Q:=Q^.URM
BD3D 890 END; ( CAUTA )
BD4D 900 PROCEDURE AFISEAZA;
BD50 910 BEGIN
BD68 920 PAGE;
BD6D 930 WRITELN;
BD70 940 WRITELN('LISTA ESTE URMATOAREA:');
BD94 950 TRAVLISTA
BD94 960 END; ( AFISEAZA )
BDA3 970 PROCEDURE INSDNOD;
BDA6 980 BEGIN
BDBE 990 WRITE('DATI CHEIA DUFA CARE SE INSEREAZA:');
BDEB 1000 READ(X);
BDF1 1010 CAUTA(B);
BDFE 1020 IF NOT B
BDFE 1030 THEN
BE07 1040 WRITE('CHEIA NU EXISTA!!!')

```

```

BE24 1050 ELSE
BE27 1060 BEGIN
BE27 1070 WRITE('DATI CHEIA:');
BE3D 1080 READ(M);
BE43 1090 CITNUM;
BE4C 1100 PREGTIP;
BE55 1110 N:=N+1;
BE5C 1120 NEW(R);
BE65 1130 R^.CHEIE:=M;
BE71 1140 R^.NUME:=NUM;
BE80 1150 R^.URM:=Q^.URM;
BE90 1160 Q^.URM:=R
BEA0 1170 END
BEA8 1180 END; ( INSDNOD )
BEAE 1190 PROCEDURE INSINOD;
BEB: 1200 BEGIN
BEC9 1210 WRITE('DATI CHEIA INAINTEA CAREIA SE INSEREAZA:');
BEFC 1220 READ(X);
BF02 1230 CAUTA(B);
BF0F 1240 IF NOT B
BF0F 1250 THEN
BF18 1260 WRITE('CHEIA NU EXISTA!!!');
BF35 1270 ELSE
BF38 1280 BEGIN
BF38 1290 WRITE('DATI CHEIA:');
BF4E 1300 READ(M);
BF54 1310 CITNUM;
BF5D 1320 PREGTIP;
BF66 1330 N:=N+1;
BF6D 1340 NEW(R);
BF76 1350 R^:=Q^;
BF83 1360 Q^.URM:=R;
BF93 1370 Q^.CHEIE:=M;
BF9F 1380 Q^.NUME:=NUM
BFA9 1390 END
BFAE 1400 END; ( INSINOD )
BFB4 1410 PROCEDURE CAUTARE;
BFB7 1420 BEGIN
BFCE 1430 WRITE('DATI CHEIA DORITA:');
BFEC 1440 READ(X);
BFF2 1450 CAUTA(B);
BFFF 1460 IF NOT B
BFFF 1470 THEN
C008 1480 WRITELN('CHEIA NU EXISTA!!!')
C025 1490 ELSE
C02B 1500 WRITELN('NODUL ESTE: ',Q^.CHEIE,' ',Q^.NUME);
C061 1510 END; ( CAUTARE )
C067 1520 PROCEDURE SUPRIMARE;
C08A 1530 BEGIN
C082 1540 WRITE('DATI CHEIA NODULUI CE SE VA SUPRIMA:');
C0B1 1550 READ(X);
C0B7 1560 CAUTA(B);
C0C4 1570 IF NOT B
C0C4 1580 THEN
C0CD 1590 WRITE('CHEIA NU EXISTA!!!');
C0EA 1600 ELSE
C0ED 1610 R:=Q^.URM;
C0FE 1620 Q^:=R^;
C108 1630 END; ( SUPRIMA )
C10E 1640
C10E 1650 BEGIN ( MAIN )
C117 1660 REPEAT
C117 1670 STAI;
C11F 1680 PAGE;
C124 1690 WRITELN(CHR(18),CHR(1),CHR(22),CHR(5),CHR(14),'OPTIUNI',CHR(18
),CHR(0));
C15A 1700 MENU;
C16F 1710 WRITE(CHR(22),CHR(21),CHR(10),'ALEGETI OPTIUNEA:');
C1A0 1720 READ(K);
C1A6 1730 CASE K OF
C1A9 1740 1:CREARE;
C1B3 1750 2:AFISEAZA;
C1C0 1760 3:BEGIN INSDNOD;STAI END;
C1C7 1770 4:INSINOD;
C1F6 1780 5:CAUTARE;
C208 1790 6:SUPRIMARE;
C21A 1800 7:HALT
C224 1810 END (CASE)
C227 1820 UNTIL K=7
C22F 1830 END {$P}.

```

6.2.5. Aplicații ale listelor înlănțuite

a. Determinarea frecvenței cuvintelor într-un text

Fiind dat un text format dintr-o succesiune de cuvinte, se cere să se afișeze cuvintele distincte și numărul aparițiilor fiecărui cuvint. Pentru rezolvare, se parcurge textul și se depistează cuvintele. La prima apariție cuvintul se inserează în lista cuvintelor distincte, în rest se incrementează contorul care înregistrează numărul de apariții ale cuvintului. În final, traversând lista, se vor afișa toate cuvintele distincte ale textului împreună cu numărul de apariții ale fiecărui cuvint.

Programul CUVINTE (P.VI.5) rezolvă această problemă pentru prima strofă din "Luceafărul" de Mihai Eminescu.

```

B1C0 20 ( AFISEAZA CUVINTELE UNUI TEXT SI FRECVENTA LOR )
B1C0 30 PROGRAM CUVINTE;
B1C0 40 TYPE TAB=ARRAY[1..10]OF CHAR;
B1C0 50     NOD=RECORD
B1C0 60     CHEIE: TAB;
B1C0 70     NUMAR: INTEGER;
B1C0 80     URM: ^NOD
B1C0 90     END;
B1C0 100    POINT=^NOD;
B1C0 110   VAR CUV: TAB;
B1C9 120   INCEPUT: POINT;
B1C9 130   PROCEDURE CAUTA(X: TAB);
B1CC 140   VAR Q: POINT;
B1CC 150   B: BOOLEAN;
B1CC 160   BEGIN
B1E4 170   Q:=INCEPUT;
B1ED 180   B:=TRUE;
B1F2 190   WHILE (Q<>NIL) AND B DO
B210 200     IF Q^.CHEIE=X
B21F 210       THEN
B231 220         B:=FALSE
B233 230         ELSE
B238 240           Q:=Q^.URM;
B24F 250     IF B
B24F 260       THEN
B256 270         BEGIN
B256 280           Q:=INCEPUT;
B25F 290           NEW(INCEPUT);
B268 300           WITH INCEPUT DO
B273 310             BEGIN
B273 320               CHEIE:=X;
B284 330               NUMAR:=1;
B294 340               URM:=Q
B29C 350             END
B2A7 360           END
B2A7 370         ELSE
B2AA 380           Q^.NUMAR:=Q^.NUMAR+1
B2C8 390         END; { CAUTA }
B2D5 400   PROCEDURE AFISARE(Q: POINT);
B2D8 410   VAR R: POINT;
B2D8 420   BEGIN
B2F0 430     R:=Q;
B2FC 440     PAGE;
B301 450     WRITE(CHR(22),CHR(1),CHR(0),'CUVINTE', ' ':14,'FRECVENTA');
B34B 460     WRITELN;
B34E 470     WRITELN;
B351 480     WHILE R<>NIL DO
B369 490       BEGIN
B369 500         WRITELN(R^.CHEIE, ' ':14,R^.NUMAR);
B39A 510         R:=R^.URM
B3A5 520       END
B3AE 530     END; { AFISARE }
B3B9 540   PROCEDURE PREBTIP;
B3BC 550   VAR I: INTEGER;

```

```

B39C 560 BEGIN
B3D4 570   FOR I:=10 DOWNT0 1 DO
B3F6 580     IF CUV[I]=CHR(0)
B41A 590       THEN CUV[I]:=CHR(32)
B445 600 END; { PREGTIP }
B451 610 BEGIN { MAIN }
B45A 620   INCEPUT:=NIL;
B460 630   PAGE;
B465 640   WRITELN('DATI CUVINTELE TEXTULUI SEPARATE PRIN <ENTER>(SFIRSIT=
<ENTER>));
B481 650   READLN;
B484 660   READ(CUV);
B48C 670   WHILE CUV[1]<<CHR(0) DO
B4E7 680     BEGIN
B4E7 690       PREGTIP;
B4EC 700       CAUTA(CUV);
B502 710       READLN;
B505 720       READ(CUV)
B50D 730     END;
B510 740   WRITELN(CHR(16));
B51A 750   AFISARE(INCEPUT);
B523 760   WRITELN(CHR(16));
B52D 770 END {$P}.

```

CUVINTE	FRECVENTA
FATA	1
FRUMOASA	1
PREA	1
O	1
IMPARATEST	1
MARI	1
RUDE	1
DIN	1
NICIODATA	1
CA	1
POVESTI	1
CA-N	1
ODATA	1
FOST	2
A	2

P. VI. 5

Observație:

Deoarece fiecare cuvânt nou s-a adăugat în listă la începutul ei, în urma traversării, cuvintele distincte apar în ordinea inversă citirii.

b. Metoda fanionului

Metoda aplicată în programul CUVINTE (P.VI.5) se poate perfecționa prin metoda fanionului implementată în programul FANION (P.VI.6), metodă care constă în adăugarea unui nod fictiv în listă, referit de pointerul FANION care permite folosirea aceleiași tehnici de inserare și pentru primul nod din listă.

```

B1F8 20 { AFISEAZA CUVINTELE UNUI TEXT S
I FRECVENTA LOR }
B1F8 30 { UTILIZIND UN POINTER CA FANION }
B1F8 40 PROGRAM FANION;
B1F8 50 TYPE TAB=ARRAY[1..10]OF CHAR;
B1F8 60   NOD=RECORD
B1F8 70     CHEIE: TAB;
B1F8 80     NUMAR: INTEGER;
B1F8 90     URM: ^NOD
B1F8 100   END;
B1F8 110   POINT:=^NOD;
B1F8 120 VAR CUV: TAB;
B201 130 INCEPUT, FANION: POINT;
B201 140 PROCEDURE CAUTA1(X: TAB; VAR INCEPUT: POINT);
B204 150 VAR Q: POINT;
B204 160 BEGIN
B21C 170   Q:=INCEPUT;
B22C 180   FANION^.CHEIE:=X;
B23D 190   WHILE Q^.CHEIE<>X DO
B261 200     Q:=Q^.URM;

```



```

B278 210 IF Q=FANION (NU S-A GASIT)
B27F 220 THEN
B28D 230 BEGIN
B28D 240 Q:=INCEPUT;
B29D 250 NEW(INCEPUT);
B2A9 260 WITH INCEPUT DO
B2BB 270 BEGIN
B2BB 280 CHEIE:=X;
B2CC 290 NUMAR:=1;
B2DC 300 URM:=Q
B2E4 310 END
B2EF 320 END
B2EF 330 ELSE
B2F2 340 Q^.NUMAR:=Q^.NUMAR+1
B310 350 END; { CAUTA }
B31C 360 PROCEDURE AFISARE(Q:POINT);
B31F 370 VAR R:POINT;
B31F 380 BEGIN
B337 390 R:=Q;
B343 400 PAGE;
B348 410 WRITELN('CUVINTE FRECVENTA');
B36E 420 WRITELN;
B371 430 WRITELN;
B374 440 WHILE R<>FANION DO
B38C 450 BEGIN
B38C 460 WRITELN(R^.CHEIE, ' ',R^.NUMAR);
B3C4 470 R:=R^.URM
B3CF 480 END
B3D8 490 END; { AFISARE }
B3E3 500 PROCEDURE PREGTIP;
B3E6 510 VAR I:INTEGER;
B3E6 520 BEGIN
B3FE 530 FOR I:=10 DOWNT0 1 DO
B420 540 IF CUV[I]=CHR(0)
B444 550 THEN
B44F 560 CUV[I]:=CHR(32)
B46F 570 END; { PREGTIP }
B47B 580 BEGIN { MAIN }
B484 590 NEW(INCEPUT);
B48D 600 FANION:=INCEPUT;
B493 610 WRITELN('DATI CUVINTELE TEXTULUI SEPARATE PRIN <ENTER>(SFIRSIT=
ENTER)');
B4DD 620 READLN;
B4E0 630 READ(CUV);
B4E8 640 WHILE CUV[I]<>CHR(0) DO
B513 650 BEGIN
B513 660 PREGTIP;
B518 670 CAUTA(CUV,INCEPUT);
B532 680 READLN;
B535 690 READ(CUV)
B53D 700 END;
B540 710 WRITELN(CHR(16));
B54A 720 AFISARE(INCEPUT);
B553 730 WRITELN(CHR(16))
B55A 740 END {$P}.

```

CUVINTE FRECVENTA

PICIOARE	1
IN	1
TOTUL	1
CALCA	1
VINE	3
CAE	1
NGROZITDAR	1
VIJELIA-	1
LUPTA	1
MINA-N	1
INSUSI	1
MIRCEA	1
BUCIUM	1
ARME	1
ZGOMOT	1
DE	2
CLOCOTI	1
CODRUL	1
ZBUCIUM	1
FREAMAT	1
MAI	2
CE	2

BATRINUL	1
PLECA	1
ABIA	1
SI	3

P. VI. 6

6.2.6. Liste dublu înlanțuite

Unele aplicații necesită traversarea listelor și înainte și înapoi (adică atât de la primul element spre ultimul cit și invers). Altfel pusă problema, fiind dat un element oarecare al listei, trebuie determinat succesorul sau predecesorul acestuia.

Dacă informația de legătură atașată unui nod al listei se completează astfel încît pe lângă cîmpul de tip pointer care indică următorul element din listă să mai existe un cîmp (tot de tip pointer) care să indice elementul din fața nodului considerat, obținem liste dublu înlanțuite.

Structura unei liste dublu înlanțuite este cea din fig. 6.22.

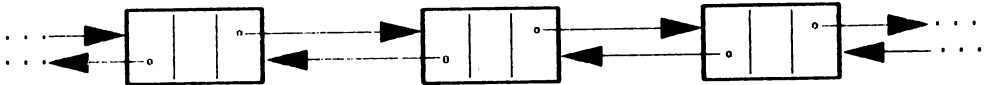


fig. 6.22

Dacă implementarea acestor liste se realizează cu pointeri, în PASCAL HP4TM definirea listei este următoarea:

```

TYPE NOD=RECORD
    ELEMENT:...;
    ANTERIOR, URMATOR: ^NOD
END;
POZITIE=^NOD;

```

Operațiile prezentate și exemplificate pe liste simplu înlanțuite (inserare, ștergere, căutare, traversare) se pot efectua și pe liste dublu înlanțuite cu precizarea că ambele cîmpuri de legătură trebuie gestionate în mod adecvat.

Pentru exemplificare, vom prezenta modul de ștergere a elementului P^{\wedge} dintr-o listă dublu înlanțuită. În fig. 6.23 avem structura listei în care pointerul P indică un element oarecare (nici primul, nici ultimul).

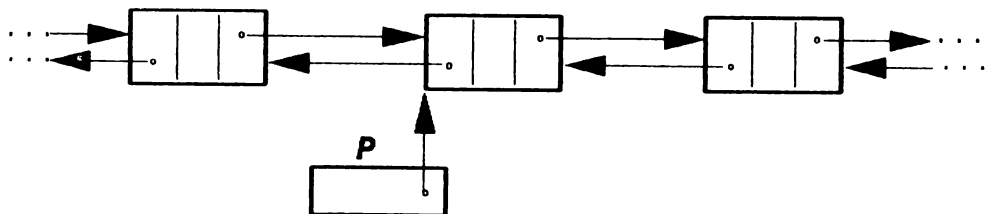


fig. 6.23

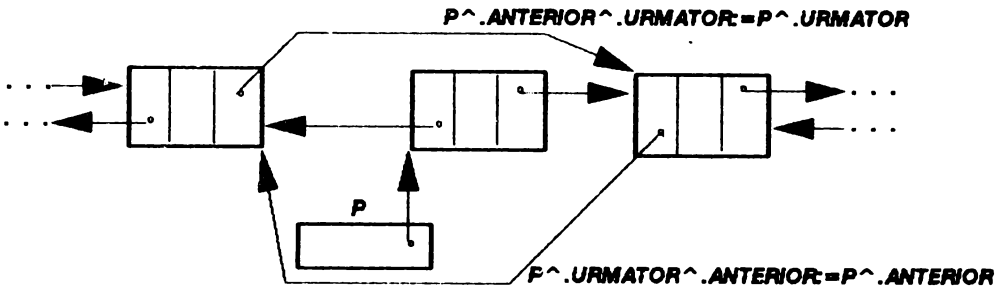
Procedura PASCAL care realizează ștergerea nodului P^{\wedge} , este:

```

PROCEDURE SUPRIMA(VAR P:POZITIE);
BEGIN
  IF P^.ANTERIOR<>NIL
    THEN P^.ANTERIOR^.URMATOR:=P^.URMATOR;
  IF P^.URMATOR<>NIL
    THEN P^.URMATOR^.ANTERIOR:=P^.ANTERIOR;
END;      {SUPRIMA}

```

Efectul execuției procedurii SUPRIMA este vizualizat în fig.6.24.



Efectul primei instrucțiuni este localizarea pointerului care indică elementul care urmează să fie șters și modificarea valorii sale astfel încât să indice succesorul nodului P. În continuare se modifică câmpul ANTERIOR al nodului care urmează celui indicat de P astfel încât el să indice nodul precedent celui indicat de P.

Nodul suprimat este indicat în continuare de P, spațiul afectat lui putînd fi reutilizat în regim de alocare dinamică a memoriei, dacă în procedură se adaugă instrucțiunea

```
RELEASE(P);
```

Observație:

Lista dublu înlănțuită devine circulară prin introducerea informațiilor de legătură care indică drept nod anterior al primului - ultimul nod și drept nod succesori al ultimului - primul nod al listei. În acest caz nu mai este necesară (în procedura SUPRIMA) comparația cu NIL.

Programul LISTADUBLINL (P.VI.7) exemplifică operațiile elementare (creare prin inserare, ștergere, căutare, traversare) posibile de efectuat și pe liste dublu înlănțuite.

```

BA43 20 { CREAZA,AFISEAZA,CAUTA,SUPRIMA,INSEREAZA INTR-O LISTA DUBLU INL
ANTUITA }
BA43 30 PROGRAM LISTADUBLINL;
BA43 40 TYPE TAB=ARRAY[1..20]OF CHAR;
BA43 50 NOD=RECORD
BA43 60 CHEIE: INTEGER;
BA43 70 NUME: TAB;
BA43 80 ANT,URM: ^NOD
BA43 90 END;
BA43 100 VAR P,Q,R,T: ^NOD;
BA4C 110 N,M,I,J,K,X: INTEGER;
BA4C 120 NUM: TAB;
BA4C 130 B: BOOLEAN;
BA4C 140 B: BOOLEAN;
BA4C 150 PROCEDURE CITNUM;
BA4F 160 BEGIN
BA67 170 WRITE('DATI NUMELE: ');
BA7E 180 READLN;READ(NUM);
BA89 190 END; { CITNUM }

```

```

BA8F 200 PROCEDURE PREGTIP;
BA92 210 BEGIN
BAAA 220   FOR I:=20 DOWNT0 1 DO
BAC3 230     IF NUM[I]=CHR(0)
BAE4 240       THEN NUM[I]:=CHR(32)
BB0C 250 END; { PREGTIP }
BB17 260 PROCEDURE TRAVLISTA;
BB1A 270 BEGIN
BB32 280   T:=P;
BB38 290   I:=1;
BB3E 300   WHILE (T<>NIL) AND (I<=N) DO
BB67 310     BEGIN
BB67 320       WRITELN;
BB6A 330       WITH T^ DO
BB75 340         WRITELN(CHEIE, ' ', NUME);
BB9C 350       I:=I+1;
BBA3 360       T:=T^.URM
BBAB 370     END
BBB1 380 END; { TRAVLISTA }
BBBA 390 PROCEDURE INSEREAZAPRIMUL;
BBBD 400 BEGIN
BBD5 410   P:=NIL;
BBDB 420   NEW(Q);
BBE4 430   WRITE('DATI CHEIA: ');
BBFA 440   READ(Q^.CHEIE);
BC06 450   P:=Q;
BC0C 460   P^.ANT:=NIL;
BC1C 470   Q^.URM:=NIL;
BC2C 480   CITNUM;
BC35 490   PREGTIP;
BC3E 500   Q^.NUME:=NUM
BC48 510 END; { INSEREAZAPRIMUL }
BC53 520 PROCEDURE INSEREAZAREST;
BC56 530 BEGIN
BC6E 540   NEW(R);
BC77 550   R^.URM:=NIL;
BC87 560   R^.ANT:=Q;
BC97 570   Q^.URM:=R;
BCA7 580   WRITE('DATI CHEIA: ');
BCED 590   READ(R^.CHEIE);
BCC9 600   CITNUM;
BCD2 610   PREGTIP;
BCDB 620   R^.NUME:=NUM;
BCEA 630   Q:=R
BCEA 640 END; { INSEREAZAREST }
BCF6 650 PROCEDURE MENUU;
BCF9 660 BEGIN
BD11 670   WRITELN('1. CREARE LISTA');
BD2E 680   WRITELN('2. AFISARE LISTA');
BD4C 690   WRITELN('3. INSERARE DUFA UN NOD');
BD71 700   WRITELN('4. INSERARE INAINTEA UNUI NOD');
BD9C 710   WRITELN('5. CAUTARE NOD DUFA CHEIE');
BDC3 720   WRITELN('6. SUPRIMARE NOD');
BDE1 730   WRITELN('7. STOP');
BDF6 740 END; { MENUU }
BDFC 750 PROCEDURE CREARE;
BDFE 760 BEGIN
BE17 770   WRITE('DATI NR.DE NODURI: ');
BE34 780   READ(N);
BE3A 790   INSEREAZAPRIMUL;
BE43 800   IF N>1 THEN
BE56 810     FOR J:=2 TO N DO
BE74 820       INSEREAZAREST;
BE84 830 END; { CREARE }
BE8A 840 PROCEDURE STAI;
BE8D 850 BEGIN
BEA5 860   REPEAT UNTIL INCH<>CHR(0)
BE80 870 END; { STAI }
BEC0 880 PROCEDURE CAUTA(WAR B:BOOLEAN);
BEC3 890 BEGIN
BED8 900   B:=FALSE;
BEES 910   T:=P;
BEEB 920   WHILE (T<>NIL) AND NOT B DO
BF0C 930     IF T^.CHEIE=X
BF14 940       THEN B:=TRUE
BF2C 950       ELSE T:=T^.URM
BF38 960 END; { CAUTA }
BF48 970 PROCEDURE AFISEAZA;
BF4B 980 BEGIN
BF63 990   PAGE;
BF68 1000  WRITELN;

```

```

BF6B 1010 IF NOT THEN
BF7E 1020 BEGIN
BF7E 1030 WRITELN('LISTA ESTE URMATOAREA:');
BFA2 1040 TRAVLISTA
BFA2 1050 END
BFAB 1060 ELSE
BFAE 1070 WRITELN('LISTA VIDA!!!!');
BFC7 1080 END; ( AFISEAZA )
BFD3 1090 PROCEDURE INSDNOD;
BFD3 1100 BEGIN
BFE3 1110 WRITE('DATI CHEIA DUPA CARE SE INSEREAZA:');
C018 1120 READ(X);
C01E 1130 CAUTA(B);
C02B 1140 IF NOT B
C02B 1150 THEN
C034 1160 WRITE('CHEIA NU EXISTA!!!')
C051 1170 ELSE
C054 1180 BEGIN
C054 1190 WRITE('DATI CHEIA:');
C06A 1200 READ(M);
C070 1210 CITNUM;
C079 1220 PREGTIP;
C082 1230 N:=N+1;
C089 1240 NEW(R);
C092 1250 R^.CHEIE:=M;
C09E 1260 R^.NUME:=NUM;
C0AD 1270 R^.URM:=T^.URM;
C0C5 1280 R^.ANT:=T;
C0D5 1290 T^.URM:=R;
C0DD 1300 END
C0E5 1310 END; ( INSDNOD )
C0EB 1320 PROCEDURE INSINOD;
C0EE 1330 BEGIN
C106 1340 WRITE('DATI CHEIA INAINTEA CAREIA SE INSEREAZA:');
C139 1350 READ(X);
C13F 1360 CAUTA(B);
C14C 1370 IF NOT B
C14C 1380 THEN
C155 1390 WRITE('CHEIA NU EXISTA!!!')
C172 1400 ELSE
C175 1410 BEGIN
C175 1420 WRITE('DATI CHEIA:');
C18B 1430 READ(M);
C191 1440 CITNUM;
C19A 1450 PREGTIP;
C1A3 1460 N:=N+1;
C1AA 1470 NEW(R);
C1B3 1480 R^.CHEIE:=M;
C1BF 1490 R^.NUME:=NUM;
C1CE 1500 IF T^.ANT=NIL
C1DE 1510 THEN
C1E8 1520 BEGIN
C1E8 1530 R^.URM:=T;
C1F8 1540 T^.ANT:=R;
C208 1550 R^.ANT:=NIL;
C218 1560 P:=R;
C218 1570 END
C21E 1580 ELSE
C221 1590 BEGIN
C221 1600 R^.URM:=T;
C231 1610 T^.ANT^.URM:=R;
C249 1620 END
C249 1630 END
C249 1640 END; ( INSINOD )
C24F 1650 PROCEDURE CAUTARE;
C252 1660 BEGIN
C26A 1670 WRITE('DATI CHEIA DORITA:');
C287 1680 READ(X);
C28D 1690 CAUTA(B);
C29A 1700 IF NOT B
C29A 1710 THEN
C2A3 1720 WRITELN('CHEIA NU EXISTA!!!')
C2C0 1730 ELSE
C2C6 1740 WRITELN('NODUL ESTE: ',T^.CHEIE,' ',T^.NUME);
C2FC 1750 END; ( CAUTARE )
C302 1760 PROCEDURE SUPRIMARE;
C305 1770 BEGIN
C31D 1780 WRITE('DATI CHEIA NODULUI CE SE VA SUPRIMA:');
C34C 1790 READ(X);
C352 1800 CAUTA(B);

```

```

C35F 1810 IF NOT B
C35F 1820 THEN
C368 1830 WRITE('CHEIA NU EXISTA!!!')
C385 1840 ELSE
C388 1850 IF T^.ANT<>NIL THEN T^.ANT^.URM:=T^.URM ELSE P:=T^.URM;
C3D3 1860 IF T^.URM<>NIL THEN T^.URM^.ANT:=T^.ANT ELSE T^.ANT^.URM:=
NIL;
C428 1870 IF B THEN N:=N-1
C437 1880 END; ( SUPRIMA )
C43C 1890
C43C 1900 BEGIN ( MAIN )
C445 1910 REPEAT
C445 1920 STAI;
C44D 1930 PAGE;
C452 1940 WRITELN(CHR(18),CHR(1),CHR(22),CHR(5),CHR(14), 'OPTIUNI',CHR(18
),CHR(0));
C498 1950 MENU;
C49D 1960 WRITE(CHR(22),CHR(21),CHR(10), 'ALEGETI OPTIUNEA:');
C4CE 1970 READ(K);
C4D4 1980 CASE K OF
C4D7 1990 1:CREARE;
C4E9 2000 2:AFISEAZA;
C4FB 2010 3:BEGIN INSDNOD;STAI END;
C512 2020 4:INSINOD;
C524 2030 5:CAUTARE;
C536 2040 6:SUPRIMARE;
C548 2050 7:HALT
C552 2060 END (CASE)
C555 2070 UNTIL K=7
C55D 2080 END (P) .

```

P. VI. 7

6.2.7. Structuri de date de tip stivă

O stivă este un tip special de listă, în care inserările și suprimările se aplică la un singur capăt, care se numește vârful stivei.

Stivele se mai numesc structuri listă de tip "LIFO" (LAST-IN - FIRST-OUT), adică "ultimul sosit - primul servit". Modelul intuitiv al unei stive este acela al unui vraf de farfurii pe o masă, în care maniera cea mai convenabilă de a lua un obiect sau de a adăuga un altul este, din motive lesne de înțeles, aceea de a acționa în vârful vrafului.

Asupra tipului de date de tip stivă putem defini următoarele proceduri care vor fi utilizate în programul STIVA (P. VI. 8).

- 1) INITIALIZARE(S)-creează stiva vidă S;
- 2) VIRFST(S) -furnizează elementul din vârful stivei S;
- 3) POP(S) -suprimă elementul din vârful stivei; de regulă este convenabil ca funcția POP să returneze elementul suprimat, necesitând astfel o implementare proprie (POP(S,X));
- 4) PUSH(X,S) -introduce elementul X în vârful stivei S;
- 5) STIVID(S) -returnează valoarea adevărată dacă S este vidă și fals în caz contrar.

Utilizarea deosebit de frecventă și cu mare eficiență a structurii de date de tip stivă în domeniul programării, a generat implementarea hardware a acestui tip de structură în toate sistemele de calcul moderne și includerea operatorilor specifici acestei structuri în setul de instrucțiuni cablate.

```

B472 33 ( CREATE,AFISARE,TIPARIRE VIRF,SCOATERE ELEMENT DIN STIVA )
B472 30 PROGRAM STIVA;
B472 40 TYPE STIVA=RECORD
B472 50     ELEM:INTEGER;
B472 60     URM:^STIVA
B472 70     END;
B472 80     POINT:^STIVA;
B472 90 VAR VIRF,P:POINT;
B472 100 STIVAB:STIVA;
B472 110 OPT:CHAR;
B472 120 OPTIUNI:SET OF CHAR;
B472 130 I,N,W:INTEGER;
B472 140 PROCEDURE INIT(VAR S:STIVA);
B472 150 BEGIN
B472 160     NEW(VIRF);
B472 170     VIRF^.URM:=NIL
B472 180 END; ( INIT )
B472 190 FUNCTION STIVID(S:STIVA):BOOLEAN;
B472 200 BEGIN
B472 210     IF VIRF^.URM=NIL
B472 220     THEN
B472 230         STIVID:=TRUE
B472 240     ELSE
B472 250         STIVID:=FALSE
B472 260 END; ( STIVID )
B472 270 FUNCTION VIRFST(S:STIVA):INTEGER;
B472 280 BEGIN
B472 290     IF STIVID(S)
B472 300     THEN
B472 310         WRITE('STIVA VIDA')
B472 320     ELSE
B472 330         VIRFST:=VIRF^.ELEM
B472 340 END; ( VIRFST )
B472 350 PROCEDURE POP(VAR S:STIVA);
B472 360 BEGIN
B472 370     VIRF:=VIRF^.URM
B472 380 END; ( POP )
B472 390 PROCEDURE PUSH(X:INTEGER;VAR S:STIVA);
B472 400 BEGIN
B472 410     NEW(P);
B472 420     P^:=VIRF^;
B472 430     VIRF^.URM:=P;
B472 440     VIRF^.ELEM:=X
B472 450 END; ( PUSH )
B472 460 PROCEDURE LISTARE(S:STIVA);
B472 470 BEGIN
B472 480     P:=VIRF;
B472 490     IF P^.URM=NIL
B472 500     THEN
B472 510         WRITE('STIVA VIDA!!!')
B472 520     ELSE
B472 530         WHILE P^.URM<>NIL DO
B472 540             BEGIN
B472 550                 WRITE(P^.ELEM);
B472 560                 P:=P^.URM
B472 570             END
B472 580 END; ( LISTARE )
B472 590 PROCEDURE MENU;
B472 600 BEGIN
B472 610     WRITELN(CHR(22),CHR(10),CHR(5),'OPTIUNI');
B472 620     WRITELN;WRITELN;
B472 630     WRITELN(' [P]JUNERE IN STIVA');
B472 640     WRITELN(' [S]COATERE DIN STIVA');
B472 650     WRITELN(' [L]LISTARE STIVA');
B472 660     WRITELN(' [T]IPARIRE VIRF STIVA');
B472 670     WRITELN(' [E]ND');
B472 680     REPEAT UNTIL INCH IN OPTIUNI
B472 690 END; ( MENU )
B472 700 BEGIN (MAIN)
B472 710     OPTIUNI:=[ 'P', 'S', 'L', 'T', 'E' ];
B472 720     INIT(STIVAB);
B472 730     REPEAT
B472 740         PAGE;
B472 750         REPEAT
B472 760             MENU;
B472 770             FOR W:=1 TO 150 DO
B472 780                 UNTIL INCH IN OPTIUNI;
B472 790             IF INCH='P'
B472 800             THEN
B472 810                 BEGIN

```

```

B856 820      WRITE('CITE ELEMENTE?');
B870 830      READ(N);
B876 840      FOR I:=1 TO N DO
B894 850        BEGIN
B897 860          WRITE('DATI ELEM.',I);
B8B5 870          READ(STIVAB.ELEM);
B8C1 880          PUSH(STIVAB.ELEM,STIVAB);
B8CD 890        END
B8D2 900      END
B8D6 910      ELSE
B8D9 920        IF INCH='S'
B8E0 930          THEN
B8EA 940            BEGIN
B8EA 950              WRITE('AM SCOS UN ELEMENT DIN STIVA!!!');
B914 960              POP(STIVAB);
B918 970            END
B91D 980          ELSE
B920 990            IF (INCH='T') AND NOT STIVID(STIVAB)
B940 1000          THEN
B94F 1010            BEGIN
B94F 1020              WRITELN;
B952 1030              WRITE('VIRFUL STIVEI ESTE:');
B974 1040              WRITE(VIRFST(STIVAB));
B992 1050              REPEAT UNTIL INCH<>CHR(0)
B99D 1060            END
B9A7 1070          ELSE
B9AA 1080            IF INCH='L'
B9B1 1090              THEN
B9B8 1100                BEGIN
B9B8 1110                  WRITELN;
B9BE 1120                  WRITELN('STIVA ESTE:');
B9DD 1130                  LISTARE(STIVAB);
B9EE 1140                END;
B9F3 1150              REPEAT UNTIL INCH IN ['G','E'];
BA24 1160              UNTIL INCH='E'
BA2B 1170            END ($P).

```

P. VI. 8

6.2.8. Structuri de date de tip coadă

Cozile sînt o altă categorie specială de liste în care elementele sînt inserate la un capăt (spate) și sînt suprimate la celălalt capăt (față). Cozile se mai numesc liste "FIFO" (FIRST-IN - FIRST-OUT) adică de tipul "primul venit, primul servit".

Operațiile care se pot efectua asupra cozii sînt analoage celor asupra stivei, cu deosebirea că inserările se fac la spatele cozii și nu la începutul ei.

În programul COADA (P.VI.9) se vor utiliza următoarele proceduri:

- 1) INITIALIZARE(C) -creează coada vidă C;
- 2) FATZE(C) -este o funcție care returnează primul element al cozii C;
- 3) ADAUGA(X,C) -inserează elementul X în spatele cozii;
- 4) SCOATE(C) -suprimă primul element al lui C;
- 5) VID(C) -returnează valoarea adevărată dacă C este vidă și fals în caz contrar.

Observație:

Deoarece inserările se fac numai la spatele cozii, în procedura ADAUGA se va păstra un pointer la ultimul element. De asemenea, se va păstra și pointerul la primul element al listei utilizat în execuția procedurilor FATZE și SCOATE. În implementare se poate utiliza un nod fictiv ca prim nod al cozii, caz în care pointerul de început va indica acest nod. Primul nod al cozii este unul fictiv în care cîmpurile informațiilor propriu-zise sînt ignorate.


```

B3D9 20 PROGRAM COADA;
B3D9 30 TYPE NOD=RECORD
B3D9 40     ELEM:INTEGER;
B3D9 50     URM:^NOD
B3D9 60     END;
B3D9 70     PNOD=^NOD;
B3D9 80     COADA=RECORD
B3D9 90         FATZA,SPATE:PNOD
B3D9 100        END;
B3D9 110 VAR EL:INTEGER;
B3E2 120 COADA8:COADA;
B3E2 130 ER:BOOLEAN;
B3E2 140 I,N:INTEGER;
B3E2 150 A:CHAR;
B3E2 160 PROCEDURE INIT(VAR C:COADA);
B3E5 170 BEGIN
B3FD 180     NEW(C.FATZA);
B409 190     C.FATZA^.URM:=NIL;
B41E 200     C.SPATE:=C.FATZA;
B436 210 END; { INIT }
B43D 220 FUNCTION VID(C:COADA):BOOLEAN;
B440 230 BEGIN
B458 240     IF C.FATZA=C.SPATE
B46E 250         THEN
B47C 260             VID:=TRUE
B47F 270             ELSE
B484 280                 VID:=FALSE
B486 290 END; { VID }
B492 300 FUNCTION FATZE(C:COADA):INTEGER;
B495 310 BEGIN
B4AD 320     IF VID(C)
B4C3 330         THEN
B4D2 340             BEGIN
B4D7 350                 ER:=TRUE;
B4D7 360                 WRITE('COADA VIDA!!!')
B4EF 370             END
B4EF 380             ELSE
B4F2 390                 FATZE:=C.FATZA^.URM^.ELEM
B504 400 END; { FATZE }
B517 410 PROCEDURE ADAUGA(X:INTEGER;VAR C:COADA);
B51A 420 BEGIN
B532 430     NEW(C.SPATE^.URM);
B546 440     C.SPATE:=C.SPATE^.URM;
B566 450     C.SPATE^.ELEM:=X;
B57E 460     C.SPATE^.URM:=NIL
B591 470 END; { ADAUGA }
B59F 480 PROCEDURE SCOATE(VAR C:COADA);
B5A2 490 BEGIN
B5BA 500     IF VID(C)
B5CF 510         THEN
B5DE 520             BEGIN
B5DE 530                 ER:=TRUE;
B5E3 540                 WRITE('COADA VIDA!!!')
B5FB 550             END
B5FB 560             ELSE
B5FE 570                 C.FATZA:=C.FATZA^.URM
B612 580 END; { SCOATE }
B621 590 PROCEDURE TIPARIRE(C:COADA);
B624 600 VAR P:PNOD;
B624 610 BEGIN
B63C 620     P:=C.FATZA;
B64D 630     P:=P^.URM;
B65F 640     WHILE P<>NIL DO
B677 650         BEGIN
B677 660             WRITE(P^.ELEM,' ');
B68C 670             P:=P^.URM
B695 680         END
B69E 690 END; { TIPARIRE }
B6AC 700 PROCEDURE MENU;
B6AF 710 BEGIN
B6C7 720     WRITELN(CHR(22),CHR(10),CHR(10),'OPTIUNI');
B6F1 730     WRITELN;WRITELN;
B6F7 740     WRITELN('ADAUGARE IN COADA');
B718 750     WRITELN('TIPARIRE ELEMENT FATZA COADA');
B744 760     WRITELN('SCOATARE ELEMENT DIN COADA');
B76E 770     WRITELN('LISTARE COADA');
B78B 780     WRITELN('OPRIRE')
B79E 790 END; { MENU }

```

```

B7A7 800 BEGIN ( MAIN )
B7B0 810   INIT(COADA);
B7B9 820   REPEAT
B7B9 830     PAGE;
B7C1 840     MENU;
B7C6 850     READLN;
B7C9 860   READ(A);
B7CF 870   CASE A OF
B7D2 880     'A':BEGIN
B7D2 890       WRITE('DATI NR.DE ELEMENTE:');
B7D7 900       READ(N);
B7F6 910       FOR I:=1 TO N DO
B7FC 920         BEGIN
B81A 930           WRITELN('DATI ELEMENTUL:',I);
B81B 940           READ(EL);
B843 950           ADAUGA(EL,COADA);
B849 960         END
B851 970       END;
B856 980     'S':SCDATE(COADA);
B85D 990     'L':TIPARIRE(COADA);
B86E 1000    'D':HALT
B89C 1010  END;
B891 1020  IF (A='T')AND NOT VID(COADA)
B894 1030  THEN
B8B4 1040    WRITE('ELEMENTUL DIN FATZA ESTE:',FATZA(COADA));
B8C3 1050  REPEAT UNTIL INCH<>CHR(0);
B905 1260  UNTIL A IN ['D']
B91A 1070  END ($P).
B92C 1080

```

P. VI. 9

6.2.9. Structura de date de tip arbore binar

Arborele, ca structură dinamică de date, s-a dovedit a fi util într-o gamă foarte variată de aplicații (căutări, sortări etc) unde se poate obține economie de timp și memorie, o minimizare a numărului de operații, ajungându-se la algoritmi eficienți și, implicit, la programe performante.

Prin *arbore binar* se înțelege o mulțime de $n \geq 0$ noduri care dacă nu este vidă, conține un nod numit *rădăcină*, iar restul nodurilor formează doi arbori binari disjunși, numiți: *subarborele stîng* și *subarborele drept*.

In fig.6.25. sînt exemple de arbori binari distincți:

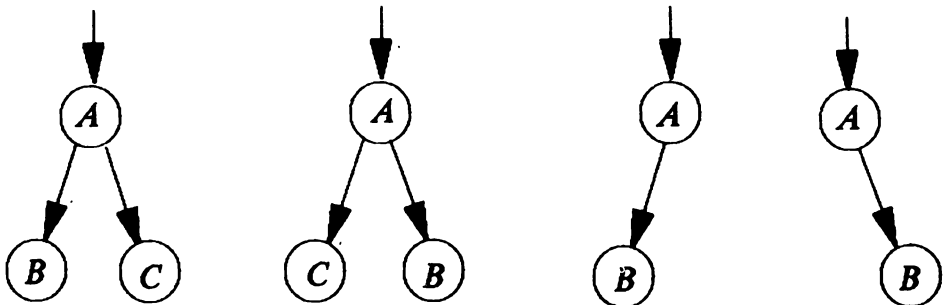


fig. 6.25

Un alt exemplu, mai complex, îl constituie *arborele atașat unei expresii aritmetice cu operatori binari*.

Fie spre exemplu expresia

$$(a+b/c)*(d-e*f)$$

căreia i se poate atașa arborele binar din fig.6.26.

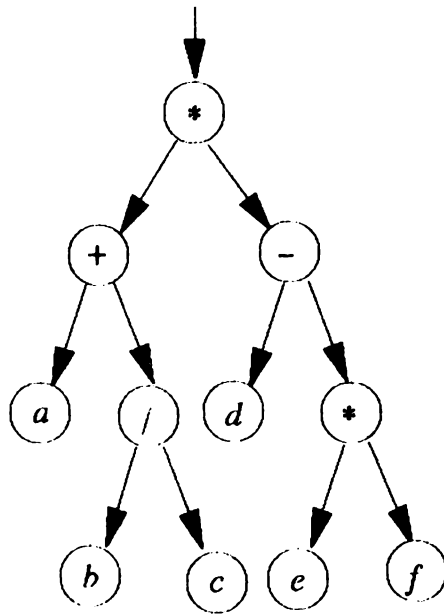


fig. 6.26

Avind la bază definiția recursivă a arborilor binari (paragraful 5.5), aceștia pot fi descriși în PASCAL HP4TM în felul următor:

```

TYPE NOD=RECORD
    INFO: ...;
    STING, DREPT: ^NOD;
END;
ARBORE=^NOD;
    
```

Astfel, structura dinamică corespunzătoare arborelui din fig.6.26, în care câmpul INFO este de tip CHAR, are configurația din fig.6.27.

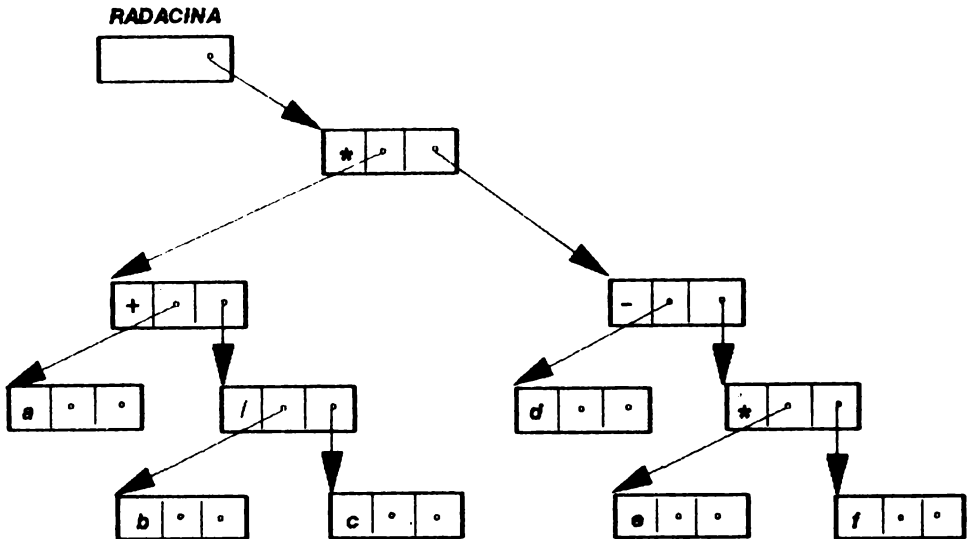


fig. 6.27

Operații fundamentale asupra arborilor binari

Din multitudinea operațiilor care se pot efectua asupra unui arbore binar vom prezenta câteva, și anume: traversarea, căutarea (localizarea), crearea, inserarea și ștergerea nodurilor unei structuri de tip arbore.

1. Traversarea

Traversarea arborilor binari constă în parcurgerea pe rînd a nodurilor arborelui în vederea efectuării unor prelucrări ale informațiilor atașate nodurilor. În timpul traversării, nodurile sînt vizitate într-o anumită ordine, astfel încît ele pot fi considerate ca și cum ar fi integrate într-o listă liniară. De fapt, descrierea celor mai mulți algoritmi este mult ușurată dacă în cursul prelucrării se poate preciza elementul următor al structurii arborescente, respectiv se poate liniariza arborele.

Algoritmul de traversare este foarte des utilizat deoarece majoritatea prelucrărilor care vizează toate nodurile unui arbore necesită un "drum" prin structură astfel încît să fie prelucrat fiecare nod o singură dată.

Există două posibilități de parcurgere a arborilor: în adîncime și în lățime, cel mai des folosită fiind cea în adîncime, care la rîndul ei poate fi de trei feluri. Ele sînt cunoscute sub denumirile de parcurgere în *inordine*, *preordine* și *postordine*. Aceste denumiri sugerează poziția rădăcinii în traversare (între cei doi descendenți, înaintea lor, respectiv după ei).

Aceste moduri de traversare se definesc recursiv astfel:

-dacă arborele A este vid, atunci traversarea lui A generează o listă vidă;

-dacă A se reduce la un singur nod, atunci traversarea este banală, reducîndu-se la acest unic nod în oricare din cele trei moduri;

-pentru restul cazurilor, considerăm arborele A cu rădăcina R și cu A1, A2 subarborii stîngi, respectiv drepte;

a) traversarea arborelui în *preordine* înseamnă lista formată din R, nodurile lui A1 traversat în preordine și nodurile lui A2 traversat tot în preordine;

b) traversarea arborelui în *inordine* înseamnă lista formată din nodurile lui A1 vizitate în inordine, R și nodurile lui A2 vizitate tot în inordine;

c) traversarea arborelui în *postordine* înseamnă lista obținută prin traversarea lui A1 în postordine, apoi a lui A2 tot în postordine și în final R.

Traversînd arborele din fig.6.27 și afișînd caracterul corespunzător nodului vizitat, se obțin următoarele secvențe:

```
preordine:    *+a/bc-d*ef;
inordine:    a+b/c*d-e*f;
postordine:  abc/+def*-*
```

Ultima secvență este cunoscută în matematică sub numele de notație poloneză postfixată.

Observație:

Prin scrierea unei expresii aritmetice în notație poloneză se înțelege scrierea operatorului după ce doi operanzi, în loc de a-l scrie între ei ca în notația algebrică obișnuită (infixată).

Exemple:

a+b	devine	ab+
a*b+c	devine	ab*c+
a*(b+c)	devine	abc+*
a-b/c	devine	abc/-
(a-b)/c	devine	ab-c/

Se observă că forma poloneză a unei expresii aritmetice se obține parcurgând în postordine arborele binar atașat.

În notația poloneză inversă (prefixată) operatorul se scrie înaintea celor doi operanzi. De exemplu: a+b se scrie +ab. Se observă că notația prefixată corespunde traversării în preordine a arborelui expresiei.

O proprietate esențială a notației poloneze este aceea că la ambele forme ea păstrează semnificația expresiei aritmetice fără a utiliza paranteze.

În paragraful 5.5 au fost enumerate câteva categorii de probleme pentru care se recomandă algoritmi recursivi. Printre ele s-a menționat și cazul problemelor care prelucrează structuri de date definite recursiv. Traversarea arborilor binari constituie un exemplu clasic pentru această categorie de probleme.

Cele trei metode de traversare le prezentăm în trei proceduri recursive în care R este o variabilă de tip pointer care indică rădăcina arborelui, iar PREL reprezintă operația care trebuie executată asupra fiecărui nod în parte.

Pentru structura de arbore în PASCAL HP4TM propunem următoarea definiție:

```

TYPE NOD=RECORD
      INFO:...;
      STING,DREPT:^NOD
    END;
REF:^NOD;

```

Procedurile corespunzătoare diferitelor moduri de traversare conțin o linie comentariu care trebuie înlocuită cu descrierea prelucrării concrete a nodului vizitat sau cu un apel al procedurii având același efect.

```

PROCEDURE PREORDINE(R:REF);
BEGIN
  IF R<>NIL THEN
    BEGIN
      {PREL(R^);}
      PREORDINE(R^.STING);
      PREORDINE(R^.DREPT)
    END;
END; {PREORDINE}

```

```

PROCEDURE INORDINE(R: REF);
BEGIN
  IF R<>NIL THEN BEGIN
    INORDINE(R^.STING);
    {PREL(R^);}
    INORDINE(R^.DREPT)
  END;
END; {INORDINE}

PROCEDURE POSTORDINE(R: REF);
BEGIN
  IF R<>NIL THEN BEGIN
    POSTORDINE(R^.STING);
    POSTORDINE(R^.DREPT);
    {PREL(R^);}
  END;
END; {POSTORDINE}

```

2. Căutarea

Problema căutării (localizării) unui nod se poate pune doar într-un arbore binar de căutare. Prin *arbore binar de căutare* se înțelege un arbore binar care are proprietatea că, parcurgând nodurile sale în inordine, secvența cheilor este monoton crescătoare (cheia este un cimp unic de identificare a nodurilor în cadrul structurii arborescente). Într-un arbore binar de căutare cheile tuturor nodurilor din subarboarele stîng al nodului N (avînd cheia C) au valori mai mici decît C și toate nodurile din subarboarele drept au valorile cheilor mai mari decît C . Din această proprietate rezultă un procedeu foarte simplu de căutare:

- se compară cheia rădăcinii (C) cu cheia căutată X ; dacă $C=X$ algoritmul se termină;
- dacă $X<C$ se reia algoritmul pentru subarboarele stîng;
- dacă $X>C$ se reia algoritmul pentru subarboarele drept.

Fie T un pointer care indică rădăcina unui arbore binar de căutare și fie X un număr întreg dat. Atunci funcția $LOC(X, T)$ prezentată în continuare execută căutarea în arborele dat a nodului cu cheia egală cu X .

Această funcție ia valoarea NIL dacă nu se găsește nici un nod cu cheia X , altfel valoarea ei este egală cu pointerul care indică acest nod.

```

TYPE NOD=RECORD
  CHEIE: INTEGER;
  STING, DREPT: ^NOD
END;
REF=^NOD;
FUNCTION LOC(X: INTEGER; VAR T: REF): REF;
VAR GASIT: BOOLEAN;
BEGIN
  GASIT:=FALSE;
  WHILE (T<>NIL) AND NOT GASIT DO
    IF T^.CHEIE=X
      THEN GASIT:=TRUE
      ELSE IF T^.CHEIE<X
            THEN T:=T^.STING
            ELSE T:=T^.DREPT;
  LOC:=T
END; {LOC}

```

Tehnica de căutare se poate simplifica dacă se aplică *metoda fanionului*. Aceasta constă în completarea arborelui cu un nod fictiv (fanion) indicat de un pointer notat cu F. În continuare structura arborelui se modifică înlocuind cu F toate referințele egale cu NIL. Înainte de demararea procesului de căutare propriu-zisă, cheia fanionului se inițializează cu X. În procesul căutării, nodul cu cheia X se găsește acum cu certitudine; dacă acest nod este fanionul, atunci în arbore nu există un nod cu cheia X, în caz contrar, nodul găsit este cel căutat.

Pentru structura astfel modificată, funcția LOC devine LOC1:

```
FUNCTION LOC1(X: INTEGER; VAR T: REF): REF;
BEGIN
  F^.CHEIE:=X;
  WHILE T^.CHEIE<>X DO
    IF X<T^.CHEIE
      THEN T:=T^.STING
      ELSE T:=T^.DREPT;
  LOC1:=T
END; {LOC1}
```

3. Crearea

Procesul de creare constă în inserarea a cîte unui nod într-un arbore binar de căutare inițial vid. Problema care se pune este de a executa inserarea astfel încît arborele să rămînă ordonat și după adăugarea noului nod. Aceasta se realizează traversînd arborele începînd cu rădăcina și selectînd fiul stîng sau drept, după cum cheia de inserat este mai mică sau mai mare decît cea a nodului parcurs. Aceasta se repetă pînă cînd se ajunge la un pointer NIL. În continuare inserarea se realizează modificînd acest pointer astfel încît să indice noul nod. Precizăm că inserarea se poate realiza chiar dacă arborele conține deja un nod cu cheia egală cu cea nouă, iar în cazul în care nu se dorește introducerea acestuia în structură, în loc de inserare se poate emite un mesaj de forma "Nodul cu cheia ... există".

În continuare se va prezenta o procedură recursivă de inserare a unui nod într-un arbore binar de căutare.

Dacă X este un număr întreg reprezentînd cheia nodului de inserat și T un pointer care indică rădăcina arborelui, atunci următoarea procedură (INARBORE) realizează inserarea noului nod cu cheia X.

Se observă că pentru funcționarea corectă a acestei proceduri este esențial ca T să fie parametru transmis prin referință, deoarece noua valoare pe care o primește T prin instrucțiunea NEW(T) numai astfel se transmite parametrului actual corespunzător.

```
PROCEDURE INARBORE(X: INTEGER; VAR T: REF);
BEGIN
  IF T<>NIL
    THEN [*]
      IF X<T^.CHEIE
        THEN
          INARBORE(X, T^.STING)
        ELSE
          INARBORE(X, T^.DREPT)
```

```

ELSE
  BEGIN
    NEW(T);
    WITH T^ DO
      BEGIN
        CHEIE:=X;
        STING:=NIL; [**]
        DREPT:=NIL [***]
      END
    END
  END; (INARBORE)

```

Considerind un arbore de căutare cu structura din fig.6.28, efectul procedurii INARBORE pentru inserarea nodului cu cheia 8 este vizualizat în fig.6.29.

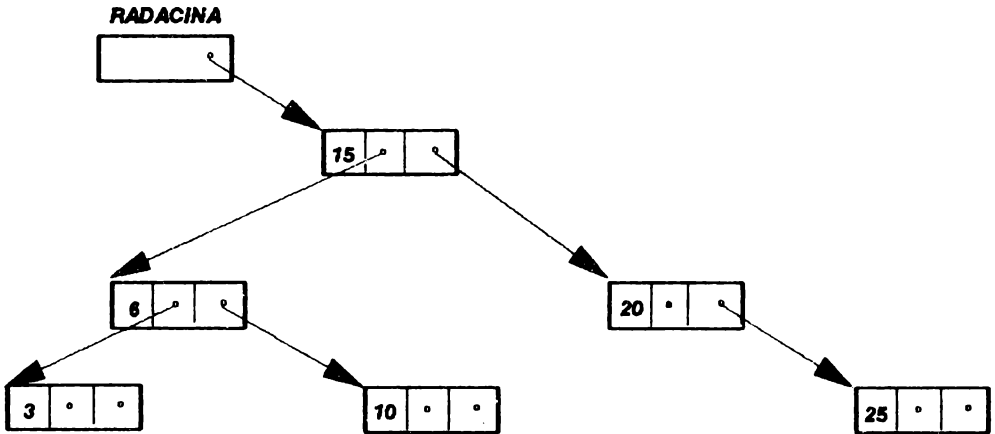


fig.6.28

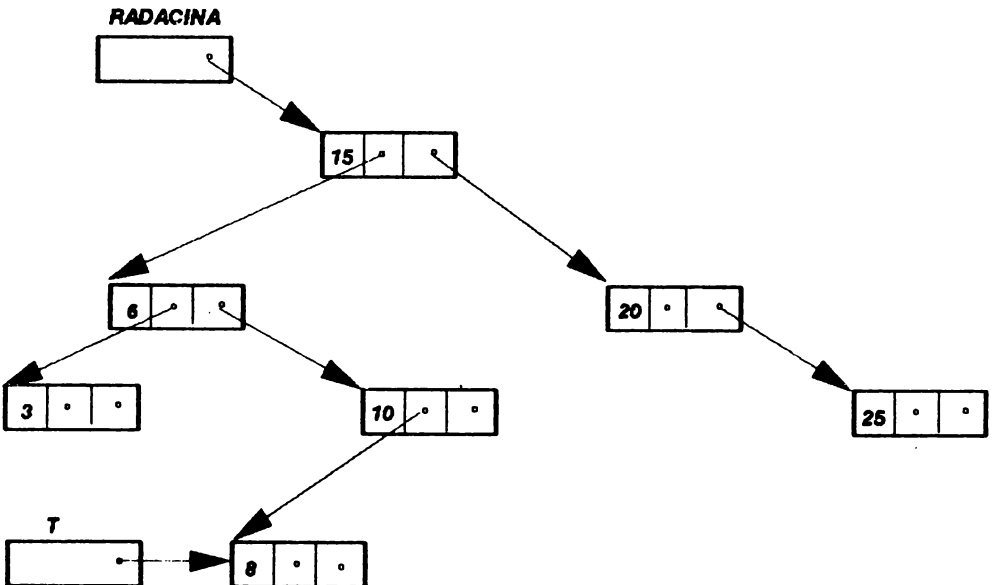


fig.6.29

În continuare se prezintă un fragment de program principal care apelează procedura de mai sus. Se presupune că toate cheile sînt diferite de zero, se introduc de la tastatură și sînt urmate de o cheie fictivă egală cu zero, valoare care încheie secvența cheilor.

```

VAR RADACINA: REF;
    C: INTEGER;
BEGIN
    RADACINA:=NIL; [*]
    READ(C);
    WHILE C<>0 DO
        BEGIN
            INARBORE(C,RADACINA);
            WRITE('DATI CHEIA: ');
            READ(C)
        END
    END;
END;

```

Utilizarea metodei fanionului în crearea arborilor binari de căutare presupune următoarele modificări:

-în secvența de program principal se declară variabila F:REF, iar instrucțiunea RADACINA:=NIL marcată cu [*] în secvența de instrucțiuni se înlocuiește cu NEW(F); RADACINA:=F; servind la inițializarea rădăcinii. În procedura INARBORE, condiția instrucțiunii IF din linia marcată cu [*] (T<>NIL) se înlocuiește cu T<>F, iar instrucțiunile STING:=NIL și DREPT:=NIL marcate cu [**] respectiv [***] se înlocuiesc cu STING:=F; și DREPT:=F;.

4. Ștergerea

Se consideră o structură de arbore binar de căutare și X o cheie precizată. Se cere să se suprimă din structura de arbore nodul avînd cheia X. Pentru aceasta, în prealabil, se caută dacă există un nod cu o astfel de cheie. Dacă nu, ștergerea se consideră încheiată și eventual se emite un mesaj. În caz contrar se execută ștergerea propriu-zisă, de o asemenea manieră încît arborele să rămînă arbore de căutare și după eliminarea nodului respectiv.

Se disting două cazuri, după cum nodul care trebuie șters are *cel mult un fiu sau doi fii*.

1. Primul caz se rezolvă conform fig.6.30-6.35 în care se prezintă cele trei alternative posibile:

- a)-nodul avînd cheia X are doar subarbore stîng;
- b)-nodul avînd cheia X are doar subarbore drept;
- c)-nodul avînd cheia X nu are subarbori.

Pentru ștergerea nodului cu cheia X, în aceste cazuri, notăm cu P pointerul corespunzător cîmpului de referință al predecesorului (tatălui) nodului avînd cheia X, care indică acest nod. valoarea lui P se modifică astfel încît acesta să indice unicul fiu al lui X (dacă acesta există) sau, în caz contrar, P devine NIL.

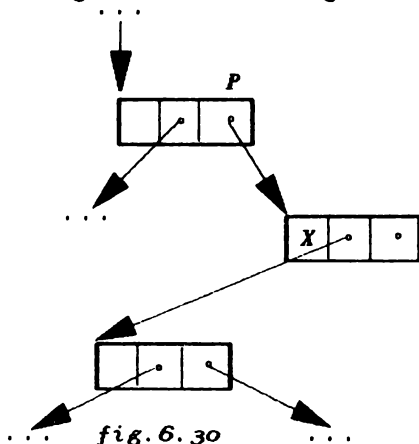
Fragmentul de program care realizează această operație este următorul:

```

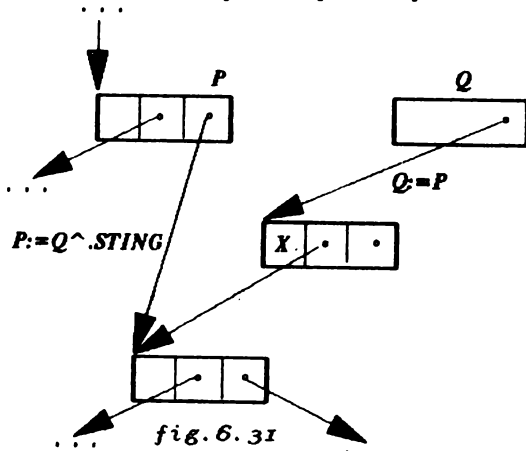
Q:=P;                {P indică nodul cu cheia X}
IF Q^.DREPT=NIL THEN P:=Q^.STING
                    ELSE IF Q^.STING=NIL
                        THEN P:=Q^.DREPT;

```

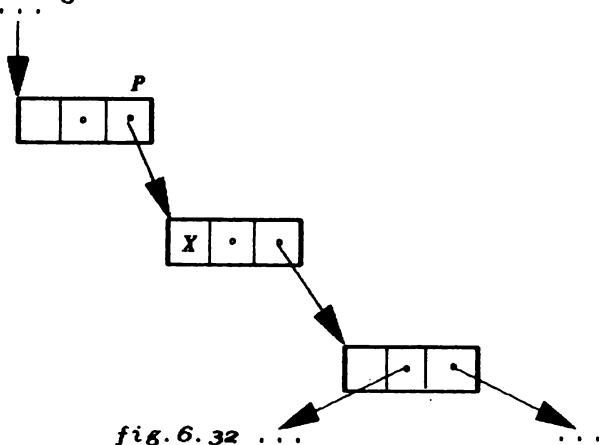
Pentru a putea urmări efectul secvenței anterioare, în cazul a), considerăm fig.6.30 cu un fragment de arbore.

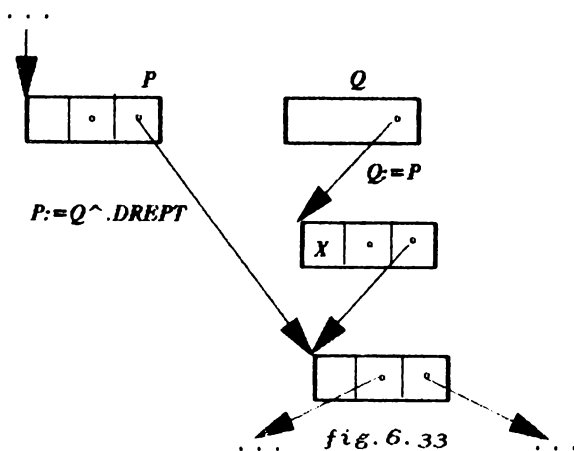


După execuția instrucțiunilor specificate, structura arborelui este cea din fig.6.31. Spațiul alocat nodului având cheia X poate fi redat memoriei prin apelul procedurii `RELEASE`.



În cazul b), pornim de la fragmentul de arbore din fig.6.32 care după efectuarea suprimării nodului având cheia X va avea structura din fig.6.33.





Cazul c) este redat grafic în fig.6.34 și 6.35.

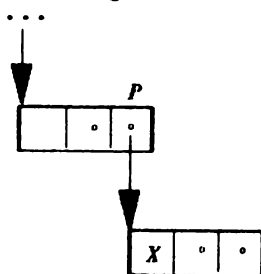


fig.6.34

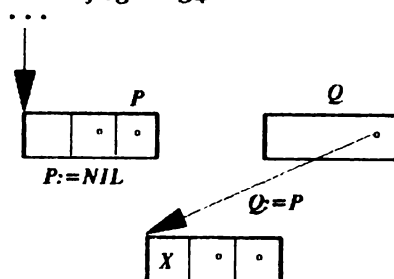


fig.6.35

2. Pentru a trata cazul în care nodul cu cheia X are doi fii, considerăm fragmentul de arbore din fig.6.36.

Suprimarea nodului X în acest caz necesită localizarea celui mai din dreapta fiu al subarborelui stâng sau a celui mai din stînga fiu al subarborelui drept. Primul are cheia precedentă cheii X , al doilea are cheia imediat următoare în secvența ordonată a cheilor, deci pentru ca arborele să rămînă arbore de căutare, unul din cele două noduri (în general cel mai din dreapta fiu al subarborelui stîng), fie acesta nodul cu cheia Y , trebuie adus în locul nodului suprimat. După ce informațiile propriu-zise ale nodului care trebuie suprimat se înlocuiesc cu informațiile propriu-zise ale nodului cu cheia Y , vechiul nod cu cheia Y (care nu are fiu drept) se suprimă ca în cazul 1 (a).

Nodul Y se detectează după următoarea metodă: se construiește o secvență de noduri care începe cu fiul stîng al lui X , după care se alege drept succesori al fiecărui nod, fiul său

drept. Primul nod al secvenței care nu are fiu drept este Y.

În urma operațiilor descrise, fragmentul de arbore din fig.6.36 va avea structura din fig.6.37.

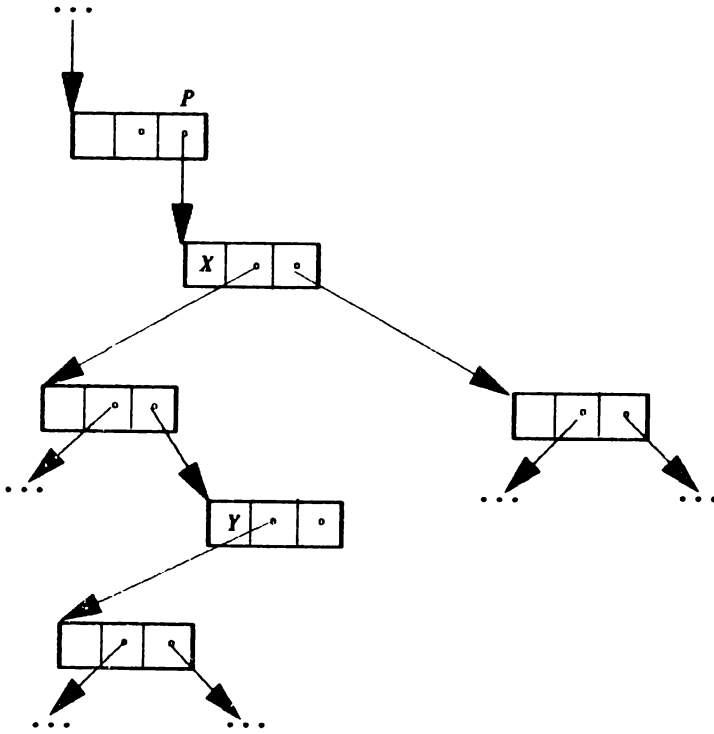


fig.6.36

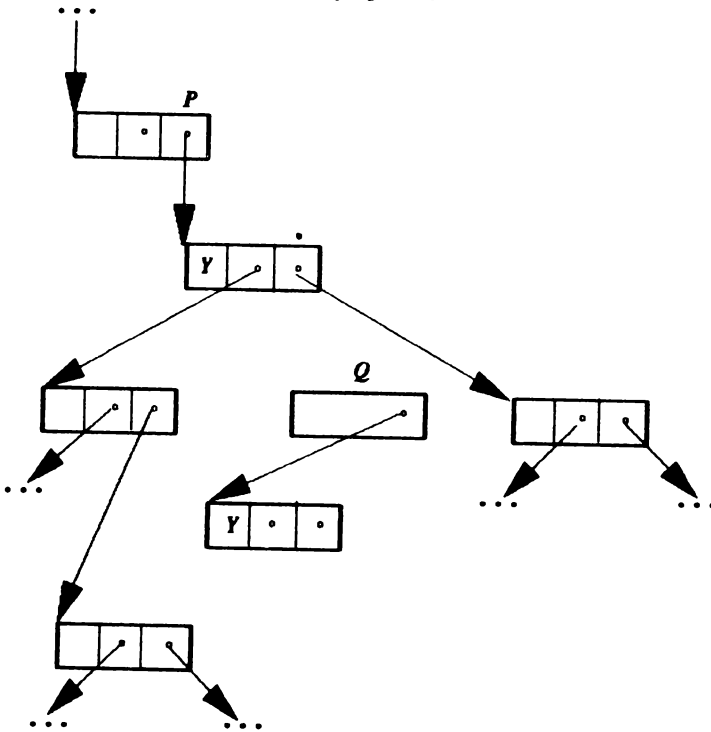


fig.6.37

Suprimarea unui nod dintr-un arbore binar de căutare se realizează cu procedura SUPRIMARE. Procedura locală SUPRED caută predecesorul în inordine al nodului cu cheia X, realizând suprimarea acestuia conform metodei descrise. Procedura SUPRED se utilizează numai în situația în care nodul cu cheia X are doi fii.

```

PROCEDURE SUPRIMARE(X: INTEGER; VAR F: REF);
VAR Q: REF;

PROCEDURE SUPRED(VAR R: REF);
BEGIN
  IF R^.DREPT<>NIL
    THEN SUPRED(R^.DREPT)
    ELSE
      BEGIN
        Q^.CHEIE:=P^.CHEIE;
        Q^.NUMAR:=P^.NUMAR;
        Q:=R;
        R:=R^.STING
      END
END; (SUPRED)

BEGIN (SUPRIMARE)
  IF P=NIL
    THEN WRITELN('NOD INEXISTENT')
    ELSE IF X<P^.CHEIE
      THEN
        SUPRIMARE(X,P^.STING)
      ELSE
        IF X>P^.CHEIE
          THEN
            SUPRIMARE(X,P^.DREPT)
          ELSE
            BEGIN
              Q:=P;
              IF Q^.DREPT=NIL
                THEN
                  P:=Q^.STING
                ELSE
                  IF Q^.STING=NIL
                    THEN
                      P:=Q^.DREPT
                    ELSE
                      SUPRED(Q^.STING);
            END
            [*]
          END;
END; (SUPRIMARE)

```

Dacă în locul marcajului [*] apelăm procedura RELEASE pentru pointerul Q, spațiul alocat nodului suprimat se redă memoriei.

În final, vom ilustra modul în care se suprimă anumite noduri dintr-un arbore de căutare prin apelul procedurii SUPRIMARE. În fig.6.38 am pus în evidență doar cimpul cheie atașat nodului, iar nodul care urmează să fie șters este marcat cu o săgeată.

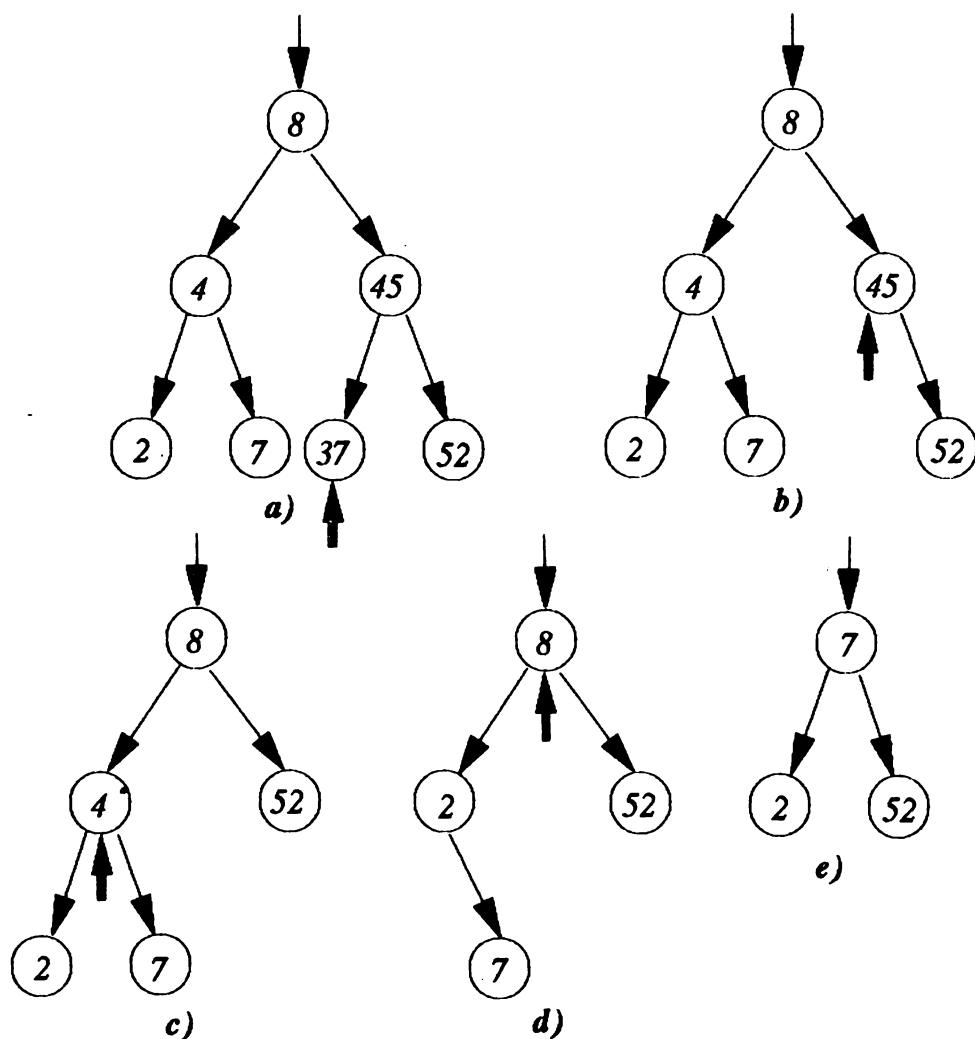


fig. 6.38

Programul **ARBOCOM** (P.VI.10) creează un arbore binar de căutare (arbore binar ordonat) și execută asupra lui operațiile prezentate în acest subparagraf.

```

BA3E 20 PROGRAM ARBOCOM;
BA4E 30 TYPE NOD=RECORD
BA5E 40     CHEIE:CHAR;
BA6E 50     STING,DREPT:^NOD
BA7E 60     END;
BA8E 70     POINT:=^NOD;
BA9E 80 VAR R,P,FANION,S:POINT;
BAAE 90     N,I:INTEGER;.
BABE 100     A,X:CHAR;
BACE 110     OPT:SET OF CHAR;
BADE 120 PROCEDURE PREORDINE(R:POINT);
BAE7 130 BEGIN
BAF7 140     IF R<>NIL
BAE6 150     THEN
BAF4 160         BEGIN
BAF4 170             WRITE(R^.CHEIE,' ');
BA83 180             PREORDINE(R^.STING);
BA99 190             PREORDINE(R^.DREPT)
BAA6 200         END
BAAF 210 END; ( PREORDINE )

```

```

BAB6 220  PROCEDURE INORDINE(R:POINT);
BAB7 230  BEGIN
BAD1 240  IF R<>NIL
BADC 250  THEN
BAE6 260  BEGIN
BAE6 270  INORDINE(R^.STING);
BAFB 280  WRITE(R^.CHEIE);
BBOE 290  INORDINE(R^.DREPT);
BE13 300  END
BE1C 310  END; { INORDINE }
BE27 320  PROCEDURE POSTORDINE(R:POINT);
BE28 330  BEGIN
BE3E 340  IF R<>NIL
BE49 350  THEN
BE5C 360  BEGIN
BE57 370  POSTORDINE(R^.STING);
BE6E 380  POSTORDINE(R^.DREPT);
BE7F 390  WRITE(R^.CHEIE)
BE89 400  END
BE92 410  END; { POSTORDINE }
BE92 420  PROCEDURE CREARE(VAR X:CHAR;VAR P:POINT);
BE93 430  BEGIN
BBAB 440  READLN;
BBAC 450  READ(X);
BBBA 460  IF X<>'.
BBC5 470  THEN
BBCE 480  BEGIN
BBCE 490  NEW(P);
BBDA 500  P^.CHEIE:=X;
BBEE 510  CREARE(X,P^.STING);
BC0A 520  CREARE(X,P^.DREPT)
BC1F 530  END
BC28 540  ELSE
BC2B 550  P:=NIL
BC36 560  END; { CREARE }
BC44 570  FUNCTION CAUTA(X:CHAR;VAR G:POINT):POINT;
BC47 580  VAR GASIT:BOOLEAN;
BC47 590  BEGIN
BC5F 600  GASIT:=FALSE;
BC63 610  WHILE (G<>NIL) AND NOT GASIT DO
BC87 620  IF G^.CHEIE=X
BC93 630  THEN
BCA1 640  GASIT:=TRUE
BCA4 650  ELSE
BCA9 660  IF G^.CHEIE<<X
BCB5 670  THEN
BCC2 680  G:=G^.STING
BCD5 690  ELSE
BCE0 700  G:=G^.DREPT;
BD00 710  CAUTA:=G
BD00 720  END; { CAUTA }
BD19 730  FUNCTION CAUTACUFANION(X:CHAR;VAR T:POINT):POINT;
BD1C 740  BEGIN
BD34 750  FANION^.CHEIE:=X;
BD3D 760  WHILE T^.CHEIE<>X DO
BD59 770  IF X<T^.CHEIE
BD68 780  THEN
BD72 790  T:=T^.STING
BD85 800  ELSE
BD90 810  T:=T^.DREPT;
BDB0 820  CAUTACUFANION:=T
BDB0 830  END; { CAUTACUFANION }
BDC8 840  PROCEDURE CREARBINORD(X:CHAR;VAR T:POINT);
BDCB 850  BEGIN
BDE3 860  IF T<>NIL
BDF2 870  THEN
BDFC 880  IF X<T^.CHEIE
BE0B 890  THEN
BE15 900  CREARBINORD(X,T^.STING)
BE26 910  ELSE
BE32 920  CREARBINORD(X,T^.DREPT)
BE45 930  ELSE
BE51 940  BEGIN
BE51 950  NEW(T);
BE5D 960  WITH T DO
BE6F 970  BEGIN
BE6F 980  CHEIE:=X;
BE78 990  STING:=NIL;
BE85 1000  DREPT:=NIL
BE90 1010  END
BE94 1020  END
BE94 1030  END; { CREARBINORD }

```

```

BE9C 1040 PROCEDURE SUPRIMARE(X:CHAR;VAR P:POINT);
BE9F 1050 VAR Q:POINT;
BE9F 1060 PROCEDURE SUPRED(VAR R:POINT);
BEA2 1070 BEGIN
BEBA 1080 IF R^.DREPT<>NIL
BED0 1090 THEN
BEDA 1100 SUPRED(R^.DREPT)
BEE8 1110 ELSE
BEF4 1120 BEGIN
BEF4 1130 Q^.CHEIE:=P^.CHEIE;
BF16 1140 Q:=R;
BF2F 1150 R:=R^.STING
BF42 1160 END
BF4A 1170 END; { SUPRED }
BF51 1180 BEGIN { SUPRIMARE }
PF69-1190 IF P=NIL
BF78 1200 THEN
BF82 1210 WRITELN('NODUL NU SE GASESTE!');
BFA1 1220 ELSE
BFA7 1230 IF X<P^.CHEIE
BF86 1240 THEN
BFC0 1250 SUPRIMARE(X,P^.STING)
BFD1 1260 ELSE
BFDD 1270 IF X>P^.CHEIE
BFEC 1280 THEN
BFF5 1290 SUPRIMARE(X,P^.DREPT)
C008 1300 ELSE
C014 1310 BEGIN
C014 1320 Q:=P;
C024 1330 IF Q^.DREPT=NIL
C036 1340 THEN
C040 1350 P:=Q^.STING
C04F 1360 ELSE
C05A 1370 IF Q^.STING=NIL
C06A 1380 THEN
C074 1390 P:=Q^.DREPT
C085 1400 ELSE
C090 1410 SUPRED(Q^.STING);
C09D 1420 RELEASE(Q)
C0A5 1430 END
C0AB 1440 END; { SUPRIMARE }
C0B4 1450 PROCEDURE MENU;
C0B7 1460 BEGIN
C0CF 1470 WRITELN(CHR(22),CHR(10),CHR(5),'OPTIUNI');
C0F9 1480 WRITELN('C)CREARE ARBORE BINAR');
C11C 1490 WRITELN('CA)RBORE BINAR ORDONAT');
C140 1500 WRITELN('CT)IPARIRE NOD ARBORE BINAR');
C169 1510 WRITELN('CL)ISTARE ARBORE BINAR');
C18D 1520 WRITELN('CS)UPRIMARE NOD');
C1AA 1530 WRITELN('CO)IPRIRE');
C1C0 1540 OPT:=['O','L','C','T','A','S'];
C203 1550 REPEAT UNTIL INCH IN OPT
C210 1560 END; { MENU }
C238 1570 BEGIN {MAIN}
C241 1580 NEW(P);
C24A 1590 REPEAT
C24A 1600 PAGE;
C252 1610 MENU;
C257 1620 READLN;
C25A 1630 READ(A);
C260 1640 CASE A OF
C263 1650 'C':BEGIN
C268 1660 WRITELN('DATI ELEMENTELE SEPARAT:');
C28E 1670 CREARE(X,P);
C29B 1680 WRITELN;
C29E 1690 WRITELN('ARBORELE BINAR ESTE CREAT!')
C2C3 1700 END;
C2C9 1710 'A':BEGIN
C2CE 1720 WRITELN('DATI ELEMENTELE SEPARAT:');
C2F4 1730 P:=NIL;
C2FA 1740 READLN;
C2FD 1750 READ(X);
C303 1760 WHILE X<>' ' DO
C316 1770 BEGIN
C316 1780 CREARBINORD(X,P);
C324 1790 READLN;
C327 1800 READ(X)
C32D 1810 END;
C330 1820 WRITELN;
C333 1830 WRITELN('ARBORELE BINAR ESTE CREAT!')
C358 1840 END;

```



```

C35E 1850      'L':BEGIN
C353 1860          WRITELN('1) PREORDINE');
C37D 1870          WRITELN('2) INORDINE');
C396 1880          WRITELN('3) POSTORDINE');
C3B1 1890          READLN;
C3B4 1900          READ(A);
C3BA 1910          CASE A OF
C3BD 1920              '1':PREORDINE(P);
C3CE 1930              '2':INORDINE(P);
C3DF 1940              '3':POSTORDINE(P)
C3E0 1950          END
C3ED 1960          END;
C3F0 1970      'T':BEGIN
C3F5 1980          WRITELN('1) FARA FANION');
C411 1990          WRITELN('2) CU FANION');
C42B 2000          READLN;
C42E 2010          READ(A);
C434 2020          WRITELN('DATI CHEIA NODULUI:');
C455 2030          READLN;
C458 2040          READ(X);
C45E 2050          CASE A OF
C461 2060              '1':BEGIN
C466 2070                  R:=CAUTA(X,P);
C479 2080                  WRITELN(R^.CHEIE)
C480 2090                  END;
C486 2100              '2':BEGIN
C488 2110                  R:=CAUTACUFANION(X,P);
C49E 2120                  WRITELN(R^.CHEIE)
C4A5 2130                  END
C4A8 2140              END
C4A8 2150          END;
C4AB 2160      'S':BEGIN
C4B0 2170          WRITELN('DATI CHEIA NODULUI:');
C4D1 2180          READLN;
C4D4 2190          READ(X);
C4DA 2200          SUPRIMARE(X,P)
C4E3 2210          END
C4E8 2220          END;
C4E8 2230          READLN;
C4EB 2240          UNTIL A='0';
C4FC 2250          END {$P}.

```

P.VI.10

Observație:

Intr-o structură de arbore, nodurile se găsesc pe niveluri definite astfel: nivelul 1 îl formează rădăcina, succesorii rădăcinii (fiii ei) formează nivelul 2, în general fiii tuturor nodurilor nivelului n formează nivelul $n+1$. Nivelul maxim al nodurilor unui arbore se numește înălțimea arborelui.

În general, înălțimea unui arbore nu este determinată de numărul nodurilor sale. Este simplu de observat că un arbore are înălțimea minimă dacă fiecare nivel conține numărul maxim de noduri posibile, cu excepția ultimului nivel.

Deoarece numărul maxim de noduri al nivelului i este 2^{i-1} , rezultă că înălțimea minimă a unui arbore binar cu n noduri este $\lceil \log_2 n \rceil + 1$.

Arborii binari de căutare, în general, nu respectă această cerință, deci o căutare necesită mai multe comparații, arborele având mai mult de $\lceil \log_2 n \rceil + 1$ niveluri.

Dacă ordinea în care sînt plasate nodurile în arbore nu contează, dar se dorește crearea unui arbore cu număr minim de niveluri, pentru construirea acestuia se poate alege o soluție ca în programul CONSTARBORE (P.VI.11) în care se creează un arbore perfect echilibrat, care are proprietatea că diferența dintre numărul nodurilor din subarborii stîng, respectiv drept, este cel mult 1.

```

B08A 20 ( CONSTRUIRE ARBORE PERFECT ECHILIBRAT )
B08A 30 PROGRAM CONSTARBORE;
B08A 40 TYPE NOD=RECORD
B08A 50     CHEIE:INTEGER;
B08A 60     STING,DREPT: ^NOD
B08A 70     END;
B08A 80     POINT=^NOD;
B08A 90 VAR RADACINA:POINT;
B093 100    N:INTEGER;
B093 110 FUNCTION ARBORE(N:INTEGER):POINT;
B096 120 VAR NODNOU:POINT;
B096 130     X,NS,ND:INTEGER;
B096 140 BEGIN (CONSTRUIRE ARBORE ECHILIBRAT CU N NODURI)
B0AE 150     IF N=0
B0B9 160     THEN
B0C3 170         ARBORE:=NIL
B0C7 180     ELSE
B0CF 190         BEGIN
B0CF 200             NS:=N DIV 2;
B0E3 210             ND:=N-NS-1;
B0FF 220             WRITE('DATI X:');
B111 230             READ(X);
B11A 240             NEW(NODNOU);
B127 250             WITH NODNOU DO
B135 260                 BEGIN
B135 270                     CHEIE:=X;
B144 280                     STING:=ARBORE(NS);
B161 290                     DREPT:=ARBORE(ND)
B171 300                 END;
B180 310                 ARBORE:=NODNOU
B180 320             END
B18C 330 END; ( ARBORE )
B198 340 PROCEDURE TIPARESTE(T:POINT;H:INTEGER);
B198 350 VAR I:INTEGER;
B198 360 BEGIN
B1B3 370     IF T<>NIL
B1BE 380     THEN
B1C8 390         WITH T DO
B1D6 400             BEGIN
B1D6 410                 TIPARESTE(STING,H-5);
B1F9 420                 FOR I:=1 TO H DO
B223 430                     WRITE(' ');
B22F 440                     WRITELN(CHEIE);
B23F 450                     TIPARESTE(DREPT,H-5)
B258 460                 END
B264 470 END; ( TIPARESTE )
B26F 480 BEGIN (MAIN)
B278 490     WRITE('DATI NR.DE NODURI:');
B295 500     READ(N);
B298 510     RADACINA:=ARBORE(N);
B2A9 520     TIPARESTE(RADACINA,20)
B2B1 530 END {$P}.

```

Probleme propuse

1. Se consideră polinoame de forma:

$$P(x) = c_1 x^{e_1} + c_2 x^{e_2} + \dots + c_n x^{e_n}, \text{ unde } e_i > 0, i = 1, \dots, n.$$

Un astfel de polinom poate fi reprezentat prin intermediul unei liste înlănțuite în care fiecare nod conține trei cimpuri: unul pentru coeficientul C_i , al doilea pentru exponentul e_i și al treilea de tip pointer la nodul următor.

Se cere să se scrie un program care determină:

- suma a două polinoame;
- diferența a două polinoame;
- produsul a două polinoame.

2. O coadă cu două capete (DEQUEUE) este o listă din care elementele pot fi șterse sau inserate la ambele capete. Se cere să se creeze astfel de cozi utilizând structuri de tip pointer.

3. Fiind dată o greutate totală GT și un set de greutăți reprezentate prin întregi pozitivi, a_1, a_2, \dots, a_n , se cere să se afle dacă greutățile se pot selecta astfel încât suma lor să fie exact GT (problema "sacului călătorului"). Se va utiliza structura de tip stivă.

4. Dându-se un text, să se afișeze cuvintele textului și frecvența lor cu ajutorul structurilor de tip arbore.

5. Un arbore binar poate fi definit ca o structură abstractă de date formată din structura de arbore propriu-zisă căreia i se asociază un set de operatori specifici: FIUSTING(N), FIUDREPT(N), PARINTE(N) și VID(N), unde N este cheia atașată nodului. Primii trei operatori returnează fiul stîng, fiul drept respectiv părintele nodului (sau 0 dacă vreunul nu există) iar VID(N) returnează valoarea adevărată dacă și numai dacă nu există nod cu cheia N .

Se cere să se implementeze aceste proceduri utilizînd reprezentarea arborilor cu ajutorul tipului pointer.

6. Catalogul unei biblioteci este organizat ca o structură de arbore binar de căutare. Fiecare nod se referă la o carte și conține titlul cărții, autorul, data intrării în bibliotecă și data ultimului împrumut. Se cere să se redacteze un program care traversează arborele și șterge toate cărțile împrumutate înaintea unei date precizate. Se precizează că datele se înregistrează ca numere întregi.

VII. ELEMENTE DE GRAFICA SI SUNET SPECIFICE LIMBAJULUI HP4TM

7.1. GENERALITATI

Este normal ca implementarea limbajului PASCAL pe calculatoare personale, care au facilități de grafică și sunet, așa cum sînt calculatoarele din familia Spectrum, să conțină și aceste extensii referitoare la grafică și sunet. Pentru a nu mări dimensiunea compilatorului și editorului, s-a adoptat ideea folosirii funcțiilor implementate și a rutinelor deja existente în memoria ROM a calculatorului. Intregul pachet de subprograme de grafică și sunet este scris în limbajul PASCAL, el putînd fi încărcat de pe casetă și adăugat oricărui program deja creat. Evident, setul de proceduri conținut în pachetul numit TURTLE, este un set minimal, dar suficient; el poate fi modificat, îmbunătățit, sau mărit prin adăugarea altor proceduri, de către utilizatori.

Ne propunem să discutăm fiecare procedură a pachetului inițial, oferit de firma Hisoft și să indicăm cîteva dezvoltări posibile.

Pachetul de subprograme inițial este conceput în stil LOGO, deci cursorul (BROASCA - TURTLE) poate fi deplasat cu ajutorul unor comenzi foarte simple. Cursorul poate lăsa urme, sau poate fi făcut invizibil, adică poate să se deplaseze fără a lăsa urme. Poziția și orientarea cursorului se păstrează în variabile globale, care sînt actualizate cînd cursorul este deplasat sau rotit; evident aceste variabile pot fi consultate sau modificate în orice moment.

7.2. VARIABILE GLOBALE

Variabilele globale utilizate de către pachetul TURTLE sînt: HEADING, XCOR, YCOR, PENSTATUS.

HEADING

Această variabilă este utilizată pentru a păstra valoarea unghiulară a orientării cursorului. Variabila poate lua orice valoare reală (în grade) și poate fi inițializată cu zero prin procedura TURTLE. Valoarea 0 corespunde direcției EST, deci după apelarea procedurii TURTLE, cursorul va fi orientat spre dreapta. Cînd variabila HEADING crește, cursorul se va roti în sens trigonometric, invers acelor de ceasornic.

XCOR, YCOR

Aceste variabile de tip real conțin coordonatele curente (X,Y - reale) ale cursorului pe ecran. Ecranul monitorului are 256x176 pixeli și cursorul poate fi poziționat în oricare punct al acestei suprafețe; dacă se încearcă scoaterea cursorului din limitele ecranului, va fi afișat mesajul 'Out of limits', iar programul va fi

oprit cu un mesaj 'HALT'. La începutul programului XCOR și YCOR sînt nedefinite, ele sînt inițializate doar prin apelarea procedurii TURTLE cu valorile 127, respectiv 87, coordonatele mijlocului ecranului, (altfel spus "BROASCA a fost palasată în mijlocul bălții sale").

PENSTATUS

Este o variabilă de tip întreg care păstrează starea curentă a "peniței" (adică a urmei lăsată de cursor). Poate lua valorile 0 sau 1. Valoarea 0 înseamnă "penița jos" - deci cursorul lasă urme în timpul deplasării; 1 înseamnă "penița sus" - deci cursorul nu lasă urme.

7.3. PROCEDURILE PACHETULUI TURTLE

În acest paragraf prezentăm textele sursă ale procedurilor pachetului inițial și efectul apelării lor precum și câteva aplicații.

7.3.1. Procedura SPOUT

```
PROCEDURE SPOUT(C:CHAR);
BEGIN
  INLINE(#FD,#21,#3A,#5C,#DO,#7E,#O2)
END;
```

Această procedură transmite parametrul C de tip caracter direct prin rutina RST #10 din ROM, evitînd interpretarea eronată de către HP4TM a parametrilor de ieșire ai procedurilor WRITE și WRITELN (în special cînd se folosesc caractere de control).

Rutina construită în INLINE este:

```
LD IY,#5C3A
LD A,(IX+2)
RST #10
```

7.3.2. Procedura CHECK

Această procedură are rolul de a verifica dacă coordonatele cursorului sînt în interiorul ecranului.

```
PROCEDURE CHECK(X,Y:INTEGER);
BEGIN
  IF (X>255) OR (X<0) OR (Y>175) OR (Y<0)
  THEN BEGIN
    WRITE('Out of limits');
    HALT
  END
END;
```

Evident în multe cazuri dorim ca programul să nu se oprească în cazul ieșirii din ecran, în vederea unor reveniri ale cursorului în ecran. Procedura se poate modifica astfel:

```
PROCEDURE CHECK(X,Y:INTEGER;VAR SW:BOOLEAN);
BEGIN
  SW:=TRUE;
  IF (X>255) OR (X<0) OR (Y>175) OR (Y<0)
  THEN SW:=FALSE
END;
```

unde SW este o variabilă globală testată înainte de a se realiza punerea unui punct pe ecran.

7.3.3. Procedura PLOT

```

PROCEDURE PLOT(ON: BOOLEAN; X, Y: INTEGER);
BEGIN
  IF ON
    THEN WRITE(CHR(21), CHR(0))
    ELSE WRITE(CHR(21), CHR(1));
  CHECK(X, Y);
  INLINE(#FD, #21, #3A, #5C, #DD, #46, #02, #DD, #4E, #04, #CD,
        #E5, #22)
END;

```

Dacă ON este TRUE atunci punctul de coordonate (X, Y) va fi plasat, oricare ar fi starea pixelului din acea poziție. Dacă ON este FALSE, atunci starea pixelului din poziția (X, Y) se inversează.

Rutina conținută în INLINE este:

```

LD IY, #5C3A
LD B, (IX+2)
LD C, (IX+4)
CALL #22E5      rutina PLOT din ROM

```

7.3.4. Procedura LINE

Are rolul de a trasa o dreaptă de la poziția curentă (XCOR, YCOR) la o nouă poziție (X+XCOR, Y+YCOR). Linia se trasează dacă ON este TRUE și se inversează starea pixelilor de pe linie dacă ON este FALSE.

```

PROCEDURE LINE(ON: BOOLEAN; X, Y: INTEGER);
VAR SGNX, SGN Y: INTEGER;
BEGIN
  CHECK(ROUND(X+XCOR), ROUND(Y+YCOR));
  IF ON
    THEN WRITE(CHR(21), CHR(0))
    ELSE WRITE(CHR(21), CHR(1));
  IF X<0
    THEN SGNX:=-1
    ELSE SGNX:=1;
  IF Y<0
    THEN SGN Y:=-1
    ELSE SGN Y:=1;
  LINE1(ABS(X), ABS(Y), SGNX, SGN Y)
END;

```

Se observă faptul că această procedură apelează la rîndul ei o altă procedură, LINE1, care nu face altceva decît să apeleze rutina DRAW din ROM.

```

PROCEDURE LINE1(X, Y, SX, XY: INTEGER);
BEGIN
  INLINE(#FD, #21, #3A, #5C, #DD, #56, #02, #DD, #5E, #04,
        #DD, #46, #06, #DD, #4E, #08, #CD, #BA, #24)
END;

```

adică:

```

LD IY, #5C3A
LD D, (IX+2)
LD E, (IX+4)
LD B, (IX+6)
LD C, (IX+8)
CALL #24BA      rutina DRAW din ROM

```

7.3.5. Procedura INK

```
PROCEDURE INK(C: INTEGER);
BEGIN
  IF (C>=0) AND (C<8)
  THEN
    SPOUT(CHR(16)); .
    SPOUT(CHR(C))
  END;
```

C este un parametru întreg cuprins între 0 și 7; procedura stabilește culoarea urmei lăsată de cursor (a "cernelei").

7.3.6. Procedura PAPER

```
PROCEDURE PAPER(C: INTEGER);
BEGIN
  IF (C>=0) AND (C<8)
  THEN
    INLINE(1,0,3,#21,0,#58,#DD,#7E,2,7,7,7,#5F,#7E,#E6,
           #C7,#B3,#77,#23,#0B,#78,#B1,#20,#FE);
    SPOUT(CHR(17));
    SPOUT(CHR(8))
  END;
```

Procedura stabilește culoarea fondului ecranului, "hîrtiei", conform culorii asociate parametrului C, un întreg între 0 și 7 inclusiv.

Rutina din INLINE este:

```
LD BC,#0300
LD HL,#5800
CICLU LD A,(IX+2)
RLCA
RLCA
RLCA
LD E,A
LD A,(HL)
AND #C
OR E
LD (HL),A
INC HL
DEC BC
LD A,B
OR C
JR NZ,CICLU
```

7.3.7. Procedura PENDOWN

Procedura modifică starea cursorului, astfel încît el să lase o urmă avînd culoarea asociată parametrului C, care este un întreg între 0 și 7 inclusiv. Procedura inițializează cu 0 variabila PENSTATUS:

```
PROCEDURE PENDOWN(C: INTEGER);
BEGIN
  PENSTATUS:=0;
  INK(C)
END;
```

7.3.8. Procedura PENUP

După apelarea procedurii, cursorul nu va mai lăsa urmă. Este utilă pentru a face deplasări dintr-o zonă în alta a ecranului. Procedura inițializează cu 1 variabila PENSTATUS.

```
PROCEDURE PENUP;
BEGIN
  PENSTATUS:=1
END;
```

7.3.9. Proceduri pentru fixare și mișcare cursor

A. Procedura SETHD

```
PROCEDURE SETHD(A: REAL);
BEGIN
  HEADING:=A
END;
```

A este un parametru real care este atribuit variabilei globale HEADING, stabilind astfel orientarea cursorului, pentru care se pot da următoarele valori:

```
0 - EST - dreapta
90 - NORD - sus
180 - VEST - stînga
270 - SUD - jos
```

B. Procedura SETXY

```
PROCEDURE SETXY(X,Y: REAL);
BEGIN
  XCOR:=X;
  YCOR:=Y
END;
```

Procedura stabilește poziția absolută a cursorului pe ecran în punctul de coordonate (X,Y). Remarcăm că nu se verifică dacă punctul aparține sau nu ecranului. Această verificare se face în procedura LINE, așa cum s-a arătat în subparagraful 7.3.4.

C. Procedura FWD

```
PROCEDURE FWD(L: REAL);
VAR NEWX,NEWY: REAL;
BEGIN
  PLOT(ROUND(XCOR),ROUND(YCOR));
  NEWX:=XCOR+L*COS(HEADING*3.1415926/180);
  NEWY:=YCOR+L*SIN(HEADING*3.1415926/180);
  LINE(TRUE,ROUND(NEWX)-ROUND(XCOR),ROUND(NEWY)-
    ROUND(YCOR));
  XCOR:=NEWX;
  YCOR:=NEWY
END;
```

Procedura deplasează cursorul înainte (FORWARD), cu L unități, în direcția în care este orientată (prin HEADING). O unitate corespunde unui pixel grafic, rotunjind, dacă este cazul, prin adaos sau prin lipsă, noile coordonate.

D. Procedura BACK

Deplasează cursorul cu L unități în direcția opusă orientării ei curente. Orientarea cursorului rămâne neschimbată.

```
PROCEDURE BACK(L: INTEGER);
BEGIN
  FWD(-L)
END;
```

E. Procedura TURN

Procedura are rolul de a modifica orientarea cursorului cu A grade, fără a-l deplasa. Orientarea este mărită în sens trigonometric, invers acelor de ceasornic.

```
PROCEDURE TURN(A: REAL);
BEGIN
  HEADING: =HEADING+A
END;
```

F. Procedura VECTOR

Procedura fixează orientarea, apoi deplasează cursorul cu L unități în direcția dată de A. După deplasare, cursorul rămâne cu aceeași orientare.

```
PROCEDURE VECTOR(A, L: REAL);
BEGIN
  SETHD(A);
  FWD(L)
END;
```

G. Procedurile RIGHT și LEFT

Procedura RIGHT este folosită ca alternativă a procedurii TURN, modificând orientarea cursorului, în sensul acelor de ceasornic, cu A grade. Procedura LEFT are același efect, sensul fiind invers.

```
PROCEDURE RIGHT(A: REAL);
BEGIN
  TURN(-A)
END;
PROCEDURE LEFT(A: REAL);
BEGIN
  TURN(A)
END;
```

H. Procedura ARCR

Această procedură realizează deplasarea cursorului pe un arc de cerc cu lungimea R. Lungimea arcului este determinată de A, unghiul la centru în sens orar. În mod normal R=0,5.

```
PROCEDURE ARCR(R: REAL; A: INTEGER);
VAR I: INTEGER;
BEGIN
  FOR I:=1 TO A DO
    BEGIN
      FWD(R);
      TURN(1)
    END
  END;
```

I. Procedura TURTLE

Procedura are rolul de a inițializa starea cursorului, plasându-l în mijlocul ecranului, orientat spre EST (dreapta). Fondul este albastru, iar urma cursorului galbenă. Această procedură trebuie apelată la începutul programului deoarece starea cursorului nu este definită în momentul lansării.

```
PROCEDURE TURTLE;
BEGIN
  PAGE;
  SETXY(127,87);
  SETHD(0);
  PAPER(1);
  PENDOWN(6)
END;
```

Observație:

Setul inițial TURTLE mai conține o procedură numită COPY care realizează o copie grafică a ecranului curent la o imprimantă ZX PRINTER. Sugerăm înlocuirea procedurii respective cu o procedură proprie, specifică imprimantei și calculatorului avute la dispoziție.

7.3.10. Proceduri pentru sunet

În pachetul TURTLE există două proceduri pentru generarea sunetelor.

```
PROCEDURE BEEP(FREQUENCY: INTEGER; LENGTH: REAL);
VAR I: INTEGER;
BEGIN
  IF FREQUENCY=0
  THEN
    FOR I:=0 TO ENTIER(12000*LENGTH) DO
    ELSE
      BEEPER(ENTIER(FREQUENCY*LENGTH),
            ENTIER(437500/FREQUENCY-30.128));
  FOR I:=1 TO 100 DO
END;
```

```
PROCEDURE BEEPER(A,B: INTEGER);
BEGIN
  INLINE(#DD,#6E,2,#DD,#66,3,#DD,#5E,4,#DD,#56,5,#CD
        #B5,3,#F3)
END;
```

adică:

```
LD L,(IX+2)
LD H,(IX+3)
LD E,(IX+4)
LD D,(IX+5)
CALL #03B5      rutina BEEPER din ROM
DI
```

7.3.11. Observații și exemple

Bineînțeles, procedurile prezentate anterior pot fi modificate, îmbunătățite, pentru a oferi utilizatorului un câmp cât mai larg de acțiune.

Cîteva exemple de astfel de modificări:

In programul DRAW (P.VII.1) a fost introdusă procedura OVER și au fost modificate corespunzător procedurile PLOT și LINE.

```

AEDB 10 PROGRAM DRAW;
AEDB 20 VAR ON:BOOLEAN;
AEE4 30
AEE4 40 PROCEDURE OVER(UNU:INTEGER);
AEE7 50 BEGIN
AEFF 60 WRITE(CHR(21),CHR(UNU))
AF10 70 END;
AF17 80
AF17 90 PROCEDURE PLOT(ON:BOOLEAN;X,Y:INTEGER);
AF1A 100 BEGIN
AF32 110 IF ON THEN OVER(0)
AF3D 120 ELSE OVER(1);
AF56 130 INLINE(#FD,#21,#3A,#5C,#DD,#46,2,#DD,#4E,4,#CD,#E5,#22)
AF63 140 END;
AF6D 150
AF6D 160 PROCEDURE LINE1(X,Y,SX,SY:INTEGER);
AF70 170 BEGIN
AF88 180 INLINE(#FD,#21,#3A,#5C,#DD,#56,2,#DD,#5E,4,#DD,#46);
AF94 185 INLINE(6,#DD,#4E,8,#CD,#BA,#24)
AF9B 190 END;
AFA5 200
AFA5 210 PROCEDURE LINE(ON:BOOLEAN;X,Y:INTEGER);
AFA8 220 VAR SGNX,SGNY:INTEGER;
AFA8 230 BEGIN
AFC0 240 IF ON THEN OVER(0)
AFCB 250 ELSE OVER(1);
AFE4 260 IF X<0 THEN SGNX:=-1 ELSE SGNX:=1;
B013 270 IF Y<0 THEN SGNY:=-1 ELSE SGNY:=1;
B042 280 LINE1(ABS(X),ABS(Y),SGNX,SGNY)
B064 290 END;
B079 300
B079 310 BEGIN
B082 320 PAGE;
B087 330 ON:=TRUE;
B08C 340 PLOT(ON,0,0);
B09E 350 LINE(ON,255,0);
B0B0 360 LINE(ON,0,175);
B0C2 370 LINE(ON,-255,0);
B0D7 380 LINE(ON,0,-175);
B0EC 390 LINE(ON,255,175);
B0FE 395 REPEAT UNTIL INCH<>CHR(0)
B109 400 END {$P}.

```

P. VII.1

In programul AT (P.VII.2) a fost introdusă procedura PRINTAT, (din acest moment putîndu-se lucra normal ca și în BASIC).

```

ACFD 10 PROGRAM AT;
ACFD 20 VAR L,C:INTEGER;
AD06 30
AD06 40 PROCEDURE SPOUT(C:CHAR);
AD09 50 BEGIN
AD21 60 INLINE(#FD,#21,#3A,#5C,#DD,#7E,2,#D7)
AD29 70 END;
AD30 80
AD30 90 PROCEDURE PRINTAT(L,C:INTEGER);
AD33 100 BEGIN
AD4B 110 SPOUT(CHR(22));SPOUT(CHR(L));SPOUT(CHR(C))
AD75 120 END;
AD88 130
AD88 140 BEGIN
AD91 150 READ(L,C);
AD9D 160 PAGE;
ADA2 170 PRINTAT(L,C);
ADAF 180 WRITE(' T E S T ');
ADC3 190 END {$P}.

```

P. VII.2

Pentru trasarea unei elipse propunem următoarea procedură:

```
PROCEDURE ELIPSOID(X,Y,R1,R2: INTEGER);
VAR NEWX,NEWY,I: INTEGER;
BEGIN
  FOR I:=0 TO 360 DO
    BEGIN
      NEWX:=+ROUND(R1*SIN(I));
      NEWY:=Y+ROUND(R2*COS(I));
      PLOT(NEWX,NEWY)
    END
  END;
END;
```

Pentru trasarea unui cerc se poate apela procedura anterioară cu R1=R2 în felul următor:

```
PROCEDURE CERC(X,Y,R: INTEGER);
BEGIN
  ELIPSOID(X,Y,R,R)
END;
```

Pentru trasarea unui cerc se poate alege și următoarea variantă:

```
FOR I:=1 TO 9 DO
BEGIN
  ARCR(0.5,360);
  RIGHT(40)
END;
```

Pentru transmiterea simultană a culorilor pentru INK și PAPER se poate folosi procedura:

```
PROCEDURE PAPERINK(P,I: INTEGER);
BEGIN
  SPOUT(CHR(17));
  SPOUT(CHR(P));
  SPOUT(CHR(16));
  SPOUT(CHR(I))
END;
```

Programul GRAFICRECURSIV (P.VII.3) conține toate rutinele din pachetul TURTLE precum și o aplicație privind trasarea unor curbe definite recursiv. Sugerăm execuția programului pentru următoarele date (perechi de numere), reprezentând valorile inițiale ale parametrilor SIZE și DIFF:

(40,15); (40,16); (40,17); (40,25); (50,32); (31,7); (31,12); (31,13); (29,6); (29,8); (20,3).

```
B78F 10 PROGRAM GRAFICRECURSIV;
B78F 20 VAR
B798 30 XCOR,YCOR,HEADING:REAL;
B798 40 A,B,PS:INTEGER;
B798 50
B798 60 PROCEDURE SPOUT(C:CHAR);
B798 70 BEGIN
B7B3 80 INLINE(#FD,#21,#3A,#5C, #DD,#7E,2.#D7)
B7B3 90 END;
B7C2 100
B7C2 110 PROCEDURE CHECK(X,Y:INTEGER);
B7C5 120 BEGIN
B7DD 130 IF (X>255) OR (X<0) OR (Y>175) OR (Y<0) THEN
```

```

140 BEGIN
150 WRITE('Out of limits');
160 HALT
170 END
180 END;
190
200 PROCEDURE PLOT(X,Y:INTEGER);
210 BEGIN
220 CHECK(X,Y);
230 SPOUT(CHR(20));SPOUT(CHR(PS));SPOUT(CHR(21));SPOUT(CHR(PS));
240 INLINE(#FD,#21,#3A,#5C, #DD,#46,2,#DD,#4E,4,#CD, #E5,#22)
250 END;
260
270 PROCEDURE LINE1(X,Y,SX,SY:INTEGER);
280 BEGIN
290 INLINE(#FD,#21,#3A,#5C, #DD,#56,2,#DD,#5E,4,#DD, #46,6,#DD,#4E,8,#CD,
300 #3A,#24)
310 END;
320
330 PROCEDURE LINE(ON:BOOLEAN;X,Y:INTEGER);
340 VAR SGNX,SGNY:INTEGER;
350 BEGIN
360 CHECK(ROUND(X+XCOR), ROUND(Y+YCOR));
370 SPOUT(CHR(20));SPOUT(CHR(PS));SPOUT(CHR(21));SPOUT(CHR(PS));
380 IF X<0 THEN SGNX:=-1 ELSE SGNX:=1;
390 IF Y<0 THEN SGNY:=-1 ELSE SGNY:=1;
400 LINE1(ABS(X),ABS(Y),SGNX, SGNY)
410 END;
420
430 PROCEDURE INK(C:INTEGER);
440 BEGIN
450 IF (C>=0) AND (C<8) THEN
460 SPOUT(CHR(16));SPOUT(CHR(C))
470 END;
480
490 PROCEDURE PAPER(C:INTEGER);
500 BEGIN
510 IF (C>=0) AND (C<8) THEN
520 INLINE(1,0,3,#21,0,#58, #DD,#7E,2,7,7,7,#5F,#7E,
530 #E6,#C7,#B3,#77,#23,#0B,#78,#B1,#20,#EE);
540 SPOUT(CHR(17));SPOUT(CHR(8));
550 END;
560
570 PROCEDURE COPY;
580 BEGIN
590 INLINE(#FD,#21,#3A,#5C,
600 #FD,#CB,#01,#CE,
610 #CD,#AC,#0E,#FD,
620 #CB,#01,#8E,#F3,
630 #C9)
640 END;
650
660 PROCEDURE PENDOWN(C:INTEGER);
670 BEGIN
680 PS:=0;
690 INK(C)
700 END;
710
720 PROCEDURE PENUP;
730 BEGIN
740 PS:=1
750 END;
760
770 PROCEDURE SETHD(A:REAL);
780 BEGIN
790 HEADING:=A
800 END;
810
820 PROCEDURE SETXY(X,Y:REAL);
830 BEGIN
840 XCOR:=X;
850 YCOR:=Y
860 END;
870
880 PROCEDURE FWD(L:REAL);
890 VAR NEWX,NEWY:REAL;
900 BEGIN
910 PLOT(ROUND(XCOR),ROUND (YCOR));
920 NEWX:=XCOR+L*COS(HEADING* 3.1415926/180);
930 NEWY:=YCOR+L*SIN(HEADING* 3.1415926/180);
940 LINE(TRUE,ROUND(NEWX)

```

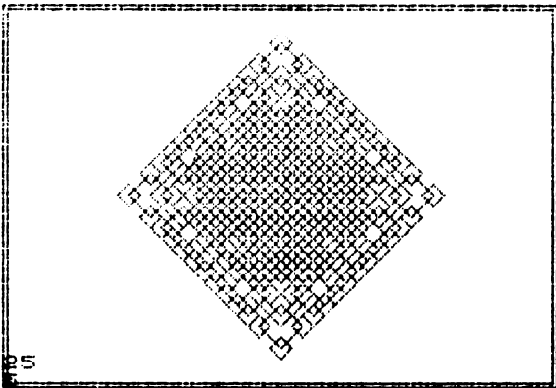
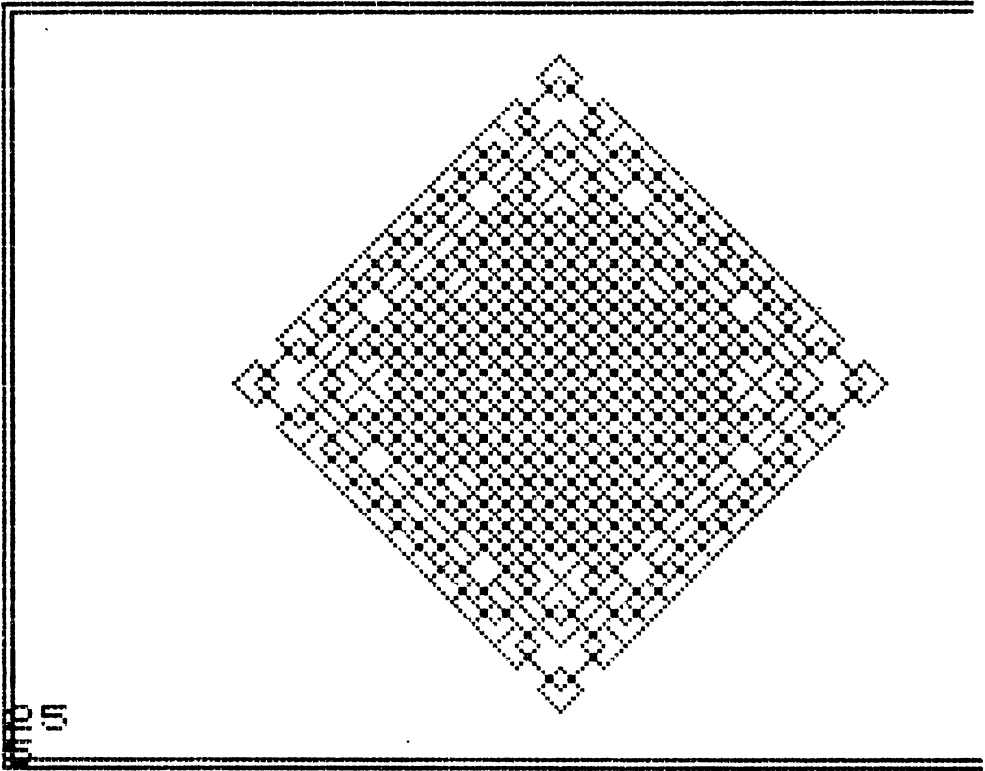
```

BD69 950 XCOR:=NEWX;
BD7C 960 YCOR:=NEWY
BD7C 970 END;
BD9E 980
BD9E 990 PROCEDURE BACK(L:REAL);
BDA1 1000 BEGIN
BDB9 1010 FWD(-L)
BDCF 1020 END;
BDE2 1030
BDE2 1040 PROCEDURE TURN(A:REAL);
BDE5 1050 BEGIN
BDFD 1060 HEADING:=HEADING+A;
BE1C 1070 END;
BE26 1080
BE26 1090 PROCEDURE VECTOR(A,L:REAL);
BE29 1100 BEGIN
BE41 1110 SETHD(A);
BE58 1120 FWD(L)
BE66 1130 END;
BE79 1140
BE79 1150 PROCEDURE RIGHT(A:REAL);
BE7C 1160 BEGIN
BE94 1170 TURN(-A)
BEAA 1180 END;
BEDD 1190
BEDD 1200 PROCEDURE LEFT(A:REAL);
BEC0 1210 BEGIN
BED8 1220 TURN(A)
BEE6 1230 END;
BEF9 1240
BEF9 1250 PROCEDURE ARCR(R:REAL; A:INTEGER);
BEFC 1260 VAR I:INTEGER;
BEFC 1270 BEGIN
BF14 1280 FOR I:=1 TO A DO
BF3E 1290 BEGIN
BF41 1300 FWD(R); TURN(1)
BF60 1310 END
BF69 1320 END;
BF78 1330
BF78 1340 PROCEDURE TURTLE;
BF7B 1350 BEGIN
BF93 1360 PAGE;
BF98 1370 SETXY(127,87);
BFB1 1380 SETHD(0);
BFC2 1390 PAPER(1);
BFCF 1400 PENDOWN(6)
BFD3 1410 END;
BFE2 1420 PROCEDURE EXIT;
BFES 1430 VAR A:INTEGER;
BFES 1440 BEGIN
BFFD 1450 READ(A);
C006 1460 IF A=1 THEN INLINE (#FD,#21,#3A,#5C,#C3,#3A,#5B) END;
C029 1470
C029 1480 PROCEDURE LINEX(X,Y:INTEGER); BEGIN
C044 1490 LINE(TRUE,X,Y);
C05F 1500 SETXY(X+XCOR,Y+YCOR) END;
C0A0 1510
C0A0 1520 PROCEDURE DIAMANT(X,Y,SIZE,DIFF:INTEGER);
C0A3 1530 BEGIN
C0BB 1540 PLOT(X,Y-SIZE);
C0E1 1550 SETXY(X,Y-SIZE);
C112 1560 LINEX(-SIZE,SIZE);
C12C 1570 LINEX(SIZE,SIZE);
C143 1580 LINEX(SIZE,-SIZE);
C15D 1590 LINEX(-SIZE,-SIZE);
C17A 1600 IF SIZE>4 THEN BEGIN
C190 1610 DIAMANT(X,Y+SIZE,SIZE-DIFF,DIFF);
C1D2 1620 DIAMANT(X,Y-SIZE,SIZE-DIFF,DIFF);
C215 1630 DIAMANT(X-SIZE,Y,SIZE-DIFF,DIFF);
C258 1640 DIAMANT(X+SIZE,Y,SIZE-DIFF,DIFF) END
C29A 1650 END;

C2A4 1660
C2A4 1670 BEGIN
C2AD 1680 TURTLE;
C2B2 1690 PAPER(0);PAGE;
C2C0 1700 REPEAT
C2C0 1710 PAGE;
C2C8 1720 READ(A,B);
C2D4 1730 SPOUT(CHR(22));SPOUT(CHR(0));SPOUT(CHR(0));

```

```
C2F5 1740 DIAMANT(128,88,A,B);  
C30A 1750 REPEAT UNTIL INCH<>CHR(0);  
C31F 1760 FOR A:=1 TO 300 DO B:=B+1;  
C346 1770 TOUT('DIAMANT ',16384,6912)  
C363 1780 UNTIL FALSE  
C365 1790 END {$P}.
```



Programul ELIPSA (P.VII.4) utilizează doar câteva din procedurile grafice, realizând un desen bazat pe trasarea unor elipse din care unele porțiuni sînt ascunse.

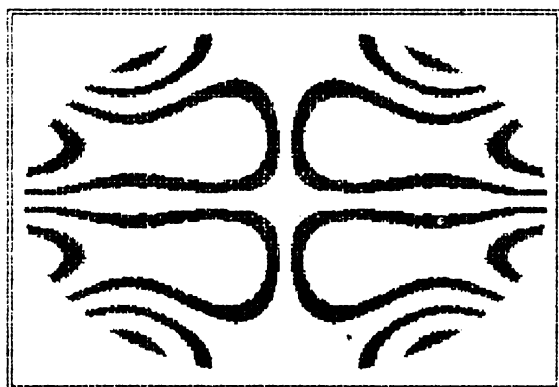
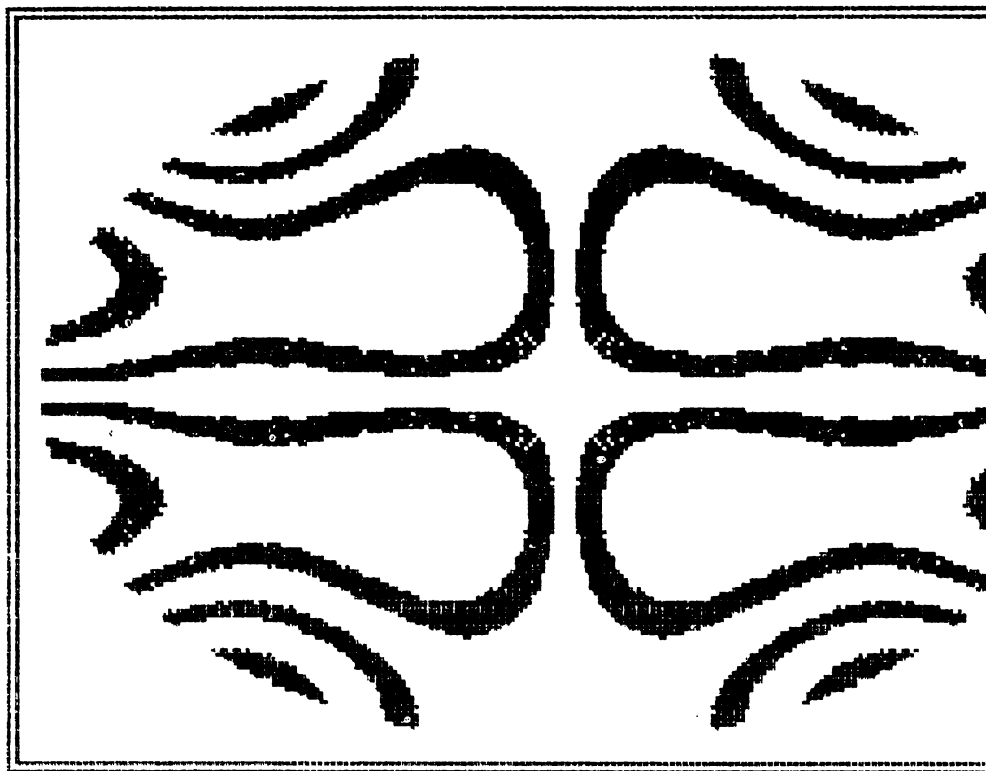
```

B1D3 10
B1D3 20 ( UTILIZARE RutINE GRAFICE
B1D3 30   TURTLE- initializare
B1D3 40   SPOUT - RST #10
B1D3 50   CHECK - out of screen$
B1D3 60   PLOT - PLOT
B1D3 70   PAPER - PAPER      )
B1D3 80
B1D3 90 PROGRAM ELIPSA;
B1D3 100 CONST PI=3.14159;
B1D3 110 VAR XCOR,YCOR,HEADING,RX,RY,X,Y,I,R1,R2:REAL;
B1DC 120   R,PS:INTEGER;
B1DC 130
B1DC 140 PROCEDURE SPOUT(C:CHAR);
B1DF 150 BEGIN
B1F7 160   INLINE(#FD,#21,#3A,#5C,#DD,#7E,2,#D7)
B1FF 170 END;
B206 180
B206 190 PROCEDURE CHECK(X,Y:INTEGER);
B209 200 BEGIN
B221 210   IF (X>255) OR (X<0) OR (Y>175) OR (Y<0) THEN
B278 220   BEGIN
B278 230     WRITE('Out of limits');
B290 240     HALT
B290 250   END
B293 260 END;
B29D 270
B29D 280 PROCEDURE PLOT(X,Y:INTEGER);
B2A0 290 BEGIN
B2B8 300   CHECK(X,Y);
B2CF 310   SPOUT(CHR(20));SPOUT(CHR(PS));SPOUT(CHR(21));SPOUT(CHR(PS));
B30B 320   INLINE(#FD,#21,#3A,#5C,#DD,#46,2,#DD,#4E,4,#CD,#E5,#22)
B318 330 END;
B322 340
B322 350
B322 360 PROCEDURE PAPER(C:INTEGER);
B325 370 BEGIN
B33D 380   IF (C)=0) AND (C<8) THEN
B36C 390   INLINE(1,0,3,#21,0,#58,#DD,#7E,2,7,7,7,#5F,#7E,#E6,
B37B 400   #C7,#B3,#77,#23,#0B,#78,#B1,#20,#EE);
B384 410   SPOUT(CHR(17));SPOUT(CHR(8));
B3A2 420 END;
B3A9 430
B3A9 440
B3A9 450 PROCEDURE TURTLE;
B3AC 460 BEGIN
B3C4 470   PAGE;
B3C9 480   PAPER(1);
B3D6 490 END;
B3DC 500
B3DC 510 BEGIN
B3E5 520   TURTLE;
B3EA 530   RX:=120;RY:=80;
B404 540   REPEAT
B404 550     I:=0;
B414 560     WHILE I<<(PI/2-1/RX) DO
B45B 570       BEGIN
B45B 580         I:=I+1/RX;
B480 590         X:=RX*COS(I);
B49D 600         Y:=RY*SIN(I);
B4BA 610         R1:=(SQRT(X)*SIN(Y/16))+SQRT(Y)*SIN(X/16)/2;
B519 620         R:=ROUND(1+COS(R1));
B534 630         IF R=2 THEN R:=1;
B54C 640         PS:=R;
B552 650         PLOT(128+ROUND(X),87+ROUND(Y));
B583 660         PLOT(128+ROUND(X),87-ROUND(Y));
B585 670         PLOT(128-ROUND(X),87+ROUND(Y));
B5E7 680         PLOT(128-ROUND(X),87-ROUND(Y))
B615 690       END;
B61D 700       RX:=RX-1;RY:=RY-0.6;
B657 710     UNTIL RX=0;
B678 720     SPOUT(CHR(22));SPOUT(CHR(0));SPOUT(CHR(0));
B699 730     REPEAT UNTIL INCH<>CHR(0);
B6AE 740     FOR R:=0 TO 255 DO
B6C8 750       BEGIN
B6CB 760         PLOT(R,0);

```



```
B6D8 770 PLOT(255-R,175)
B6EC 780 END;
B6F4 790 FOR R:=0 TO 175 DO
B70E 800 BEGIN
B711 810 PLOT(0,R);
B71E 820 PLOT(255,175-R)
B732 830 END;
B73A 840 TOUT('elipsa ',16384,6912)
B757 850 END {$P}.
```



P. VII. 4

VIII. MIC MANUAL DE OPERARE PASCAL HP4TM

8.1. INTRODUCERE

8.1.1. Punerea în funcțiune

Pascal HP4TM este o versiune rapidă, ușor de folosit și în același timp puternică a limbajului Pascal. Ea prezintă câteva diferențe față de PASCAL standard, de exemplu:

- FILE nu este implementată, totuși variabilele pot fi stocate pe bandă;
- tipul înregistrare nu poate avea variante;
- nu sînt admiși parametri procedură și parametri funcție.

În plus, PASCAL HP4TM pune la dispoziția utilizatorilor, pe lângă subprogramele standard care se regăsesc în orice implementare, câteva proceduri și funcții specifice, unele apelînd rutine din memoria ROM a calculatorului.

Pascal HP4TM utilizează diferite coduri de control, mai ales în cadrul editorului. Desigur, calculatoarele pot avea diferite configurații de tastatură și astfel diferite moduri de formare a codurilor de caractere. În acest manual, caracterele de control vor fi <CR> (RETURN), CC, CH, CI, CP, CS și CX.

În timpul editării unui program PASCAL HP4TM pot fi folosite următoarele caractere de control:

- <CR> (RETURN, ENTER)- se folosește pentru a termina linia;
- CC (CAPS SHIFT+1)-se revine în editor;
- CH (CAPS SHIFT+0)-șterge ultimul caracter introdus;
- CI (CAPS SHIFT+8)-deplasează la următoarea poziție TAB;
- CP (CAPS SHIFT+3)-schimbă ieșirea pe printer (dacă există), iar dacă ieșirea era pe printer, se revine pe monitor;
- CX (CAPS SHIFT+5)-șterge întreaga linie;
- CS (CAPS SHIFT+SPACE)-BREAK.

Încărcarea HP4TM se face cu comanda LOAD "". Mai întâi se încarcă un scurt program în BASIC, cu autostart, care va încărca efectiv blocul HP4TM. După ce HP4TM s-a încărcat, el va fi lansat automat în execuție și va afișa mesajul:

Top of RAM?

Trebuie să se introducă un număr zecimal pozitiv pînă la 65535 (acest număr reprezentînd noua valoare a variabilei de sistem RAMTOP), apoi <CR> (RETURN), fie direct <CR> (RETURN) (în acest caz variabila de sistem nemodificîndu-se). RAMTOP se află memorat pe doi octeți la adresa 23730.

Stiva compilatorului este plasată aici, deci rezervarea unei zone de memorie pentru rutine proprii în cod

mașină sau pentru memorarea altor informații se poate face prin tastarea unei valori mai mici decât RAMTOP-ul real. În versiunea pentru ZX SPECTRUM, RAMTOP-ul real este considerat a fi începutul zonei de caractere grafice definite de utilizator (UDG).

Se cere apoi: Top of RAM for 'T'.

Se poate introduce fie un număr zecimal, fie rămâne implicit valoarea anterioară pentru 'Top of RAM'. Valoarea introdusă va fi considerată adresa de început a stivei pentru cazul când se execută codul obiect rezultat prin comanda 'T' a editorului. Este necesar să se definească o stivă de rulare diferită de RAMTOP-ul real dacă, de exemplu, s-au scris extensii la modulele de execuție care se vor introduce în locațiile de deasupra RAMTOP-ului.

În fine, se cere: Table size?

Ceea ce se introduce va fi mărimea zonei de memorie care trebuie alocată tabelului de simboluri al compilatorului. Ca mai înainte, se poate introduce un număr zecimal urmat de <CR>, sau numai <CR>, în care caz pentru mărimea acestei zone de memorie va fi adoptată o valoare implicită (memoria RAM disponibilă împărțită la 16). În aproape toate cazurile, valoarea implicită asigură spațiu mai mult decât suficient pentru tabelul de simboluri. Tabelul de simboluri nu se poate extinde peste adresa #8000=32768 în zecimal. Dacă se menționează că va avea loc această depășire, se va cere din nou să se introducă 'Top of RAM' și celelalte opțiuni.

Opțional se poate include E înainte de numărul introdus, dacă se dorește ca editorul intern să nu fie reținut pentru a fi utilizat cu compilatorul, de exemplu dacă se folosește în acest scop un editor propriu.

În acest stadiu, compilatorul și editorul integral (dacă a fost reținut) vor fi relocalitate după tabelul de simboluri, iar execuția transferată editorului.

8.1.2. Compilarea și lansarea în execuție

Odată ce a fost invocat, compilatorul generează un listing de forma:

```
xxxx nnnn textul liniei sursă
```

unde:

xxxx este adresa la care începe codul generat de această linie,

nnnn este numărul liniei, cu zerourile de aliniere suprimate.

Dacă o linie conține mai mult de 80 de caractere, atunci compilatorul va introduce caractere 'linie nouă' <CR>, în așa fel încât lungimea unei linii să nu depășească 80 de caractere.

Listarea poate fi dirijată către printer prin folosirea opțiunii P, dacă aceasta este acceptată.

Listarea poate fi oprită în orice moment, apăsând CS (BREAK); în continuare, cu CC se revine în editor, iar cu orice altă tastă se continuă listarea.

Dacă pe parcursul compilării este detectată o eroare, atunci va fi afișat mesajul '*ERROR*', urmat de caracterul '^', plasat după simbolul care a generat eroarea și de un număr de eroare. Listarea se va opri; apăsând E, linia respectivă este adusă în zona de editare pentru efectuarea corecțiilor, iar apăsând P în linia de editare este adusă linia anterioară celei care a produs eroarea.

8.2. EDITORUL

8.2.1. Introducere în editor

Editorul furnizat cu toate versiunile Pascal 4 Hisoft este un editor simplu, la nivel de linie, ușor de exploatat și care editează programele rapid și eficient.

Editorul este lansat automat în execuție după ce HP4TM este încărcat de pe casetă, afișând mai întâi mesajul:

Copyright Hisoft 1983,84
All rights reserved

și apoi prompterul editorului: '>>'. Din acest moment se pot introduce linii de comandă cu formatul următor:

c n1, n2, s1, s2 <CR>

unde:

c - este comanda care urmează a fi executată;
n1, n2- sînt numere în intervalul [1,32767] inclusiv;
s1, s2- sînt șiruri cu maximum 20 de caractere.

Virgula este utilizată pentru a separa diferitele argumente (dar acest separator poate fi schimbat, vezi comanda S), iar spațiile sînt ignorate, cu excepția celor din șirurile de caractere.

Nici unul dintre aceste argumente nu este obligatoriu, deși unele comenzi (cum ar fi comanda D) nu acționează pînă nu sînt specificate n1 și n2. Editorul memorează numerele și șirurile introduse chiar dacă nu se specifică unul dintre argumentele liniei de comandă; va utiliza aceste valori introduse anterior, acolo unde este cazul.

Inițial n1 și n2 au valoarea 10, iar șirurile sînt vide. Dacă se introduce o linie de comandă incorectă, de exemplu F-1, 100, HELLO, ea va fi ignorată și se va afișa mesajul 'Pardon?'. Linia trebuie introdusă corect, adică F1, 100, HELLO. Același mesaj de eroare apare și dacă lungimea lui s2 depășește 20; dacă lungimea lui s1 depășește 20, caracterele în plus vor fi ignorate.

Comenzile pot fi introduse cu litere mari sau mici.

La introducerea unei linii de comandă pot fi folosite toate funcțiile de control descrise anterior, (de exemplu CX pentru a șterge toată linia).

Paragraful următor detaliază diferitele comenzi disponibile în editor.

Observație:

-dacă unele argumente sînt încadrate între simbolurile '<<' și '>>', atunci respectivele argumente trebuie să fie prezente, altfel comanda nu acționează.

8.2.2. Comenzile editorului

a) Inserarea textului

Textul poate fi introdus într-o filă de text, fie introducînd un număr de linie, un spațiu și apoi linia program, fie utilizînd comanda I. La introducerea unui text pot fi utilizate și funcțiile de control CX, CI, CC, CP și DELETE.

Comanda I n, m

Cu ajutorul acestei comenzi se obține intrarea în modul automat de inserare: vor fi afișate numerele de linie, începînd de la n , cu pasul m . După afișarea numărului de linie se introduce linia program. Aici se pot folosi funcțiile de control. Linia de text se termină cu <CR>. Pentru a ieși din acest mod se folosește funcția de control CC.

Dacă se introduce o linie a cărui număr există în text, atunci linia existentă va fi înlocuită cu cea nouă după tastarea lui <CR>. Dacă incrementarea automată, a numărului de linie produce un număr de linie mai mare decît 32767, se va ieși automat din modul inserare.

b) Listarea textului

Textul poate fi inspectat folosind comanda L; numărul de linii afișat deodată la executarea acestei comenzi este fixat inițial, dar poate fi modificat cu comanda K.

Comanda L n, m

Această comandă listează textul curent pe dispozitivul de afișare, de la linia cu numărul n , pînă la linia cu numărul m inclusiv. Valoarea implicită a lui n este întotdeauna 1, iar a lui m este 32767, adică valorile implicite nu sînt luate din argumentele introduse anterior. Pentru a lista întregul fișier de text, se utilizează simplu L, fără alte argumente. După listarea unui anumit număr de linii (fixat prin comanda K), listarea se va opri; cu funcția de control CC se revine în editor iar cu oricare altă tastă se continuă listarea.

Comanda K n

K stabilește numărul liniilor de ecran care urmează să fie listate înainte ca listarea să fie întreruptă, așa cum este descris la comanda L. Este calculată și stocată valoarea $n \text{ MOD } 256$.

c) Editarea textului

În editor există diverse comenzi pentru corectarea, ștergerea, mutarea sau renumerotarea liniilor.

Comanda D <n, m>

Toate liniile de la n la m inclusiv sînt șterse din fișierul de text. Dacă $m=n$ este ștersă o singură linie.

Comanda M n, m

Această comandă mută linia n la linia m ; vechea linie m , dacă exista, se va pierde, iar linia n rămîne neafectată. Dacă linia n nu există, comanda este inefectivă.

Comanda N <n, m>

Utilizarea comenzii N are ca efect renumerotarea liniilor programului sursă, prima linie primind numărul n , iar pasul fiind m . Dacă renumerotarea duce la numere de linii mai mari decît 32767, atunci va fi păstrată numerotarea inițială.

Comanda F n,m,f,s

În textul cuprins între liniile n (<m este căutat șirul f. Dacă un asemenea șir de caractere este identificat, atunci este afișată linia care îl conține și apoi se intră în modul editare. Se pot folosi subcomenzile F, S ale modului editare. Comanda F se termină cu <CR>.

Valorile numerelor de linii și cele două șiruri puteau fi specificate anterior prin oricare altă comandă și în acest caz este suficient să se introducă doar F.

Comanda E n

Editează linia cu numărul n. Dacă există, ea este copiată într-o zonă tampon și afișată pe ecran (împreună cu numărul de linie). Numărul de linie este apoi afișat din nou, imediat dedesubt și se trece în modul editare. În modul editare sînt disponibile următoarele subcomenzi:

- (spațiu) - deplasează cursorul cu o poziție spre dreapta, marcînd caracterul următor din linie;
- CH - deplasează cursorul cu o poziție spre stînga, marcînd caracterul anterior;
- CI - deplasează cursorul spre dreapta, pînă la următoarea poziție TAB;
- <CR> - termină editarea liniei curente, cu păstrarea tuturor modificărilor;
- Q - termină editarea liniei curente fără să rețină nici una din modificările făcute;
- R - reîncarcă linia în tamponul editorului, adică ignoră toate modificările făcute și reface linia originală;
- L - listează restul liniei care a mai rămas de corectat, adică partea care urmează după poziția curentă a cursorului; se rămîne în modul editare, cu cursorul re poziționat pe începutul liniei;
- K - șterge caracterul care se află la poziția curentă a cursorului;
- Z - șterge toate caracterele de la poziția curentă a cursorului (inclusiv) pînă la sfîrșitul liniei;
- F - găsește apariția următoare a șirului f de caractere definit anterior printr-o linie de comandă F; această subcomandă va termina automat editarea liniei curente (cu menținerea modificărilor) dacă aici nu este identificată o altă apariție a șirului f; dacă o apariție a șirului f este detectată într-o linie următoare (între limitele specificate anterior), atunci va fi introdus modul editare pentru linia în care a fost găsit șirul f; după o căutare încheiată cu succes, cursorul va fi poziționat la începutul liniei;
- S - înlocuiește apariția curentă a șirului f cu șirul s definit într-o linie de comandă F, după care se execută o subcomandă F, adică se caută următoarea apariție a șirului f;
- I - inserează caractere începînd de la poziția curentă a cursorului; începutul inserării este marcat prin transformarea cursorului în '*'; se rămîne în acest submod pînă cînd se apasă <CR>; atunci se revine în

modul editare, cu cursorul poziționat după ultimul caracter inserat; folosind DELETE în cadrul acestui submod caracterul de la stînga cursorului va fi șters, iar cu CI se va deplasa cursorul în poziția următoare, inserînd spații;

- X - avansează cursorul la sfîrșitul liniei și se trece în submodul de inserare;
- C - permite ca peste caracterul aflat la poziția curentă a cursorului să se scrie alt caracter; începutul modificării este marcat prin transformarea cursorului în '+'; se rămîne în submodul de modificare pînă cînd se apasă <CR> și atunci se revine în modul editare cu cursorul poziționat după ultimul caracter modificat; DELETE în cadrul acestui submod mută cursorul cu o poziție spre stînga, iar CI nu are nici un efect.

d) Comenzi pentru lucrul cu banda

Comanda P n,m,s

Liniiile din domeniul definit prin n<m sînt salvate pe casetă în format HP4TM, sub denumirea specificată prin șirul s. Aceste argumente pot fi specificate și printr-o comandă anterioară. În cazul în care se dorește realizarea mai multor copii, pentru copiile 2,3,..., este suficientă tastarea comenzii P fără argumente. (În timpul salvării filei text în fișierul cu numele s va fi afișat mesajul 'Busy...')

Comanda G,,s

Pe bandă este căutat un fișier în format HP4TM, cu denumirea s. Vor fi afișate denumirile fișierelor întîlnite în timpul derulării benzii, precedate de mesajul 'Found' Odată găsit fișierul căutat, se va afișa mesajul 'Using', urmat de numele fișierului, iar acesta va fi încărcat în memorie. Dacă pe timpul încărcării este detectată o eroare, va fi afișat mesajul 'Checksum error' sau 'Tape error', iar încărcarea este oprită.

Dacă șirul s este vid, va fi încărcat primul fișier HP4TM întîlnit pe bandă, indiferent de denumirea lui.

Căutarea și încărcarea de pe bandă se poate opri apăsînd CS; apăsînd apoi CC se revine în editare. Dacă există deja o filă text în memorie, fișierul citit de pe bandă va fi adăugat la aceasta și fila text rezultată va fi renumerotată, începînd de la numărul de linie 1 cu pasul 1.

e) Compilarea și lansarea în execuție din editor

Comanda C n

Se compilează textul care începe la linia cu numărul n. Dacă nu se specifică un număr de linie, textul va fi compilat de la prima linie existentă.

Comanda R

Codul obiect compilat anterior va fi executat, dar numai dacă sursa nu a fost modificată între timp.

Comanda T n

Textul sursă este compilat de la linia n sau de la început dacă n este omis, iar dacă compilarea se termină cu succes, apare mesajul 'Ok?'; dacă răspunsul este 'Y', atunci codul obiect produs prin compilare este deplasat la sfârșitul modulelor de execuție (distrugând compilatorul), apoi modulele de execuție și codul obiect vor fi salvate pe bandă, denumirea de fișier fiind cea specificată pentru fișierul f definit anterior. Ulterior acest fișier se poate încărca în memorie, folosind încărcătorul HP4TM, după care automat va fi executat codul obiect. Intrucît codul obiect este deplasat la sfârșitul modulelor de execuție, după comanda T compilatorul nu va mai fi în memorie și va trebui reîncărcat de pe bandă.

Dacă nu se dorește salvarea pe bandă, răspunsul la întrebarea 'Ok?' va fi orice caracter diferit de 'Y'; controlul va reveni la editor, acesta funcționînd normal, intrucît codul obiect nu a fost deplasat.

f) Alte comenzi**Comanda B**

Această comandă redă controlul sistemului de operare. În cazul calculatorului ZX SPECTRUM, se revine în BASIC. Pentru a reda controlul compilatorului se folosește

RANDOMIZE USR 24598,

caz în care textul existent se șterge, fie cu comanda

RANDOMIZE USR 24603

cînd textul Pascal sursă se păstrează.

Comanda O n,m

Se utilizează pentru a codifica un eventual text, obținut, de exemplu, cu ajutorul unui alt editor de texte. Textul este citit într-o zonă tampon în forma expandată și apoi dus înapoi în fișier sub formă codificată.

Comanda S,,d

Cu această comandă se poate schimba delimitatorul folosit pentru separarea argumentelor într-o linie de comandă. La pornirea editorului virgula (',') este luată ca delimitator; aceasta poate fi schimbată, folosind comanda S, cu primul caracter din șirul d specificat în comandă.

Menționăm că spațiul nu poate servi ca separator.

Comanda V

Afișează valorile curente pentru n1, n2, s1, s2.

Comanda X

Afișează în hexa adresa de sfârșit a compilatorului.

Comanda W n,m,s

Funcționează la fel ca P cu excepția faptului că fila text nu se salvează în format standard HP4TM. Textul cuprins între liniile m și n se va salva pe bandă într-un format

special sub numele *s*. Textul astfel salvat va fi folosit direct la compilare cu ajutorul opțiunii F a compilatorului.

8.3. Opțiuni ale compilatorului

Sintaxa specificării opțiunilor de compilare este dată în diagrama de sintaxă din fig.8.1.

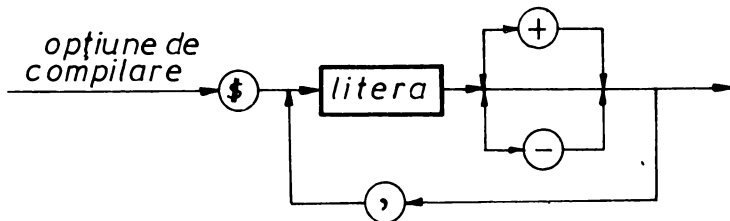


fig.8.1

Observație:

-opțiunile compilatorului care admit specificația '+', respectiv '-', pot fi anulate prin opțiunea complementară.

Opțiunea L

Controlează listarea programului text și a adreselor codului mașină generat de compilator.

Când se menționează L+, se listează textul programului.

Pentru L- sînt listate numai liniile în care este detectată o eroare.

Implicit: L+

Opțiunea O

Controlează dacă se fac unele verificări de depășire. Pentru înmulțirea și împărțirea numerelor întregi, ca și pentru toate operațiile cu numere reale, aceste verificări se fac întotdeauna. În cazul O+ se fac verificări la adunarea și scăderea numerelor întregi.

În cazul O-, aceste verificări nu se fac.

Implicit: O+.

Opțiunea C

Controlează testarea claviaturii în timpul executării programului în cod mașină. Dacă se specifică C+, atunci execuția se oprește când se apasă CC, cu apariția unui mesaj HALT.

Acest control se face înaintea tuturor ciclurilor, procedurilor și funcțiilor. Utilizatorul poate folosi această facilitate pentru a determina, în cursul unei depanări, ce structuri repetitive nu se termină corect. Această opțiune trebuie, evident, să fie dezactivată dacă vrem ca programul obiect să ruleze rapid.

În cazul C-, verificarea de mai sus nu se face.

Implicit: C+

Opțiunea S

Controlează testarea stivei.

În cazul S+, la apelarea fiecărei proceduri sau

funcții se face o verificare dacă există probabilitatea ca stiva să fie depășită în blocul care urmează. Dacă există pericolul ca în timpul execuției stiva să se suprapună peste zona afectată variabilelor dinamice sau peste program, execuția este oprită și se afișează mesajul 'Out of RAM at PC=XXXX'. Evident, nu este o certitudine; dacă o procedură folosește mult stiva, atunci programul se poate într-adevăr 'prăbuși'. Pe de altă parte, în cazul când o funcție folosește puțin stiva, oprirea execuției se poate dovedi inutilă.

In cazul S-, nu se execută testarea stivei.

Implicit: S+.

Opțiunea A:

Controlează verificarea încadrării indicilor tablourilor în limitele specificate la declararea tablourilor respective.

Dacă se specifică A+, iar indicele unui element de tablou este prea mare sau prea mic, execuția programului este oprită, fiind afișat mesajul 'Index too high', sau 'Index too low'.

In cazul A-, această verificare nu se mai face.

Implicit: A+.

Opțiunea I

Folosind aritmetica numerelor întregi pe 16 biți în cod complementar, are loc o depășire când se execută una din operațiile <, >, <=, >= dacă diferența dintre argumente este mai mare decât MAXINT (32767). În acest caz rezultatul comparării este incorect. În mod normal, aceasta nu supără; dacă, însă, utilizatorul dorește să compare asemenea numere, I+ va asigura corectitudinea rezultatelor. O situație analoagă poate apare în aritmetica numerelor reale, când va fi generată o eroare de depășire dacă argumentele diferă prin mai mult decât aproximativ 3.4E38.

In cazul I- nu se face verificarea rezultatelor comparărilor de mai sus.

Implicit: I-

Opțiunea P

Dacă se folosește opțiunea P, dispozitivul la care va fi transmis listingul după compilare va fi comutat, adică dacă înainte era utilizat monitorul, după P va fi utilizat printerul și invers. Observați că această opțiune nu este urmată de '+' sau de '-'.

Implicit: Este utilizat ecranul video.

Opțiunea F

Litera F trebuie să fie urmată de un spațiu și apoi denumirea unui fișier, formată din 8 caractere. Dacă denumirea fișierului are mai puțin de 8 caractere, se completează cu spații.

Prezența acestei opțiuni va determina includerea, la sfârșitul liniei curente, a textului sursă Pascal din fișierul specificat. Este utilă în cazul când programatorul dorește să alcătuiască pe bandă o 'bibliotecă' din multe proceduri și funcții proprii și apoi să o includă în diferite programe.

Programul trebuie să fi fost salvat cu comanda W din

editor. In majoritatea sistemelor trebuie folosită opțiunea L-, altfel viteza de compilare va fi mică.

Exemplu:

```
{ $L-, F MATRIX } include de pe bandă textul fișierului  
MATRIX
```

Cînd se editează programe foarte mari, este posibil să nu rămînă loc suficient în memoria calculatorului pentru ca programul sursă și codul obiect să fie prezente simultan. Este posibil, totuși să fie compilate asemenea programe, salvîndu-le pe bandă și folosind opțiunea F. In acest caz, în RAM vor fi - în orice moment - numai 128 octeți din sursă, rămînînd mult loc pentru codul obiect.

Această opțiune nu poate fi inclusă într-o structură repetitivă.

Observație:

Opțiunile compilatorului se pot utiliza și în mod selectiv.

Astfel, secțiunile deja depanate pot fi compactate și făcute să ruleze mai rapid, prin dezactivarea verificărilor inutile și reținînd aceste verificări numai pentru părțile netestate ale codului.

ANEXA 1. CUVINTE REZERVATE SI IDENTIFICATORI PREDEFINITI

A.1.1. Cuvinte rezervate

AND	ARRAY	BEGIN	CASE	CONST	DIV
DO	DOWNTO	ELSE	END	FOR	FORWARD
FUNCTION	GOTO	IF	IN	LABEL	MOD
NIL	NOT	OF	OR	PACKED	PROCEDURE
PROGRAM	RECORD	REPEAT	SET	THEN	TO
TYPE	UNTIL	VAR	WHILE	WITH	

A.1.2. Simboluri speciale

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

A.1.3. Identificatori predefiniți

ABS	ADDR	ARCTAN	BOOLEAN	CHAR	CHR
COS	ENTIER	EOLN	EXP	FALSE	FRAC
HALT	INCH	INLINE	INTEGER	INP	LN
MARK	MAXINT	NEW	ODD	ORD	OUT
PAGE	PEEK	POKE	PRED	RANDOM	READ
READLN	REAL	RELEASE	ROUND	SIN	SIZE
SQR	SQRT	SUCC	TAN	TIN	TOUT
TRUE	TRUNC	USER	WRITE	WRITELN	

unde avem:

-constante: MAXINT=32767;
 -tipuri: BOOLEAN=(FALSE,TRUE);
 CHAR (Setul de caractere ASCII extins);
 REAL (Submultime a numerelor reale)
 -proceduri: WRITE; WRITELN; READ; READLN; PAGE; HALT; USER;
 POKE; INLINE; OUT; NEW; MARK; RELEASE; TIN;
 TOUT;
 -functii: ABS; SQR; ODD; RANDOM; ORD; SUCC; PRED; INCH;
 EOLN; PEEK; CHR; SQRT; ENTIER; ROUND; TRUNC;
 FRAC; SIN; COS; TAN; ARCTAN; EXP; LN; ADDR;
 SIZE; INP;

ANEXA 2. HP4TM -COMPILATOR PASCAL ZX SPECTRUM- MEMORATOR

1. COMENZI DE EDITARE

- I n, m** -inserare de text sursă;
-n=numărul liniei;
-m=pasul de numerotare;
-după fiecare linie se tastează <CR>, în acest moment apărind numărul liniei următoare;
-ieșirea din înserare se face cu <caps-1>;
- L n, m** -listare program;
-n=numărul primei linii listate;
-m=numărul ultimei linii listate;
- D n, m** -ștergere linii;
-n=numărul ultimei linii șterse;
- N n, m** -renumerotare a liniilor programului;
-n=noua etichetă a primei linii;
-m=noul pas de numerotare;
- E n** -editarea liniei n;
- F x, y, string1, string2**
-între liniile x și y se caută șirul1 și eventual se înlocuiește cu șirul2.

2. MEMORAREA, INCARCAREA SI TIPARIREA PROGRAMELOR

- P n, m; s** -salvare program;
-textul sursă cuprins între liniile n și m va fi plasat pe casetă sub numele s;
- G , , s** -încărcare program de pe casetă;
- W n, m, s** -salvarea unor porțiuni de program;
-porțiunile astfel salvate pot fi inserate în program folosind opțiunea compilatorului
(\$F s);
- (\$F s) -apariția acestei opțiuni în textul programului are ca efect (în timpul compilării) introducerea în acel punct a unei proceduri salvate cu W sub numele s;
- (\$L-) -la întâlnirea acestei opțiuni în timpul compilării se sistează listarea textului sursă pe (rezultă creșterea vitezei de compilare);
- B** -reîntoarcere în BASIC;
-revenirea în PASCAL se face cu
-RANDOMIZE USR 24603 cu păstrarea textului;
-RANDOMIZE USR 24598 cu ștergerea textului;
- (\$P) -se comută canalul de ieșire de la monitor la imprimantă și invers.

3. ALTE COMENZI

- C n** -compilare începînd cu linia n;
- K n** -fixează numărul de linii afișate simultan pe

	ecran
M n, m	-copierea liniei n în linia m;
R	-lansare program (RUN);
T n	-compilarea programului și salvarea codului obiect;
V	-indică forma comenzii F n,m,f,s;
X	-furnizează adresa de sfârșit a compilatorului în hexa.

4. OPTIUNI DE EDITARE

space	-cursor dreapta;
caps o	-cursor stînga;
caps 5	-ștergerea liniei inclusiv eticheta;
caps 8	-salt la următorul TAB;
enter	-ieșire din editare cu validarea modificărilor;
C	-scrie peste caracterele existente; (ieșire cu <CR>);
F	-caută;
I	-inserează;
K	-șterge caracterul de sub cursor;
L	-listarea liniei cu modificările făcute;
Q	-ieșire din editare fără validarea modificărilor;
R	-reîncărcarea liniei fără modificări;
S	-înlocuiește;
X	-salt la sfârșitul liniei;
Z	-șterge toate caracterele de după cursor pînă la sfârșitul liniei.

5. MESAJE DE EROARE

1	-număr prea mare;
2	-lipsește ";";
3	-identificator nedeclarat;
4	-lipsește un identificator;
5	-s-a folosit ":"=" în loc de "=" în declarea constantelor;
6	-lipsește "=";
7	-identificator ce nu poate să apară în membrul stîng al unei atribuirii;
8	-lipsește ":"=";
9	-lipsește ")";
10	-tip eronat;
11	-lipsește ". ";
12	-lipsește un factor;
13	-lipsește o constantă;
14	-acest identificator nu este o constantă;
15	-lipsește THEN;
16	-lipsește DO;
17	-lipsește TO sau DOWNTO;
18	-lipsește "(";
19	-incompatibilitate de tip;
20	-lipsește OF;
21	-lipsește ", ";
22	-lipsește ":";
23	-lipsește PROGRAM;
24	-lipsă variabilă deoarece parametrul e de tip -variabilă;
25	-lipsește BEGIN;
26	-lipsește o variabilă la utilizarea lui READ;
27	-acest tip de expresii nu pot fi comparate;

- 28 -tipul utilizat trebuie să fie INTEGER sau REAL;
- 29 -acest tip de variabilă nu poate fi citit cu READ;
- 30 -acest identificator nu este un tip;
- 31 -exponentul trebuie să fie un număr real;
- 32 -lipsește o expresie scalară (nenumerică);
- 33 -nu sînt permise șiruri vide; folosiți CHR(0);
- 34 -lipsește [;
- 35 -lipsește];
- 36 -indicii de tablou trebuie să fie de tip scalar;
- 37 -lipsește "...";
- 38 -lipsește "," sau "]" într-o declarație de tablou;
- 39 -limita inferioară > limita superioară;
- 40 -mulțime mai mare de 256 de elemente;
- 41 -identificatorul de după FUNCTION () trebuie să fie identificator de tip;
- 42 -lipsește "," sau "]" într-o mulțime;
- 43 -lipsește "... sau "]" într-o mulțime;
- 44 -tipul parametrului trebuie să fie un identificator de tip;
- 45 -a nu se folosi o mulțime vidă ca prim factor în afara unei atribuirii;
- 46 -lipsește un tip scalar, inclusiv REAL;
- 47 -lipsește un tip scalar, exclusiv REAL;
- 48 -mulțimile nu sînt compatibile;
- 49 -"<" și ">" nu pot să apară în comparații de mulțimi;
- 50 -lipsește FORWARD, LABEL, CONST, VAR, TYPE, sau BEGIN;
- 51 -lipsește un număr hexa;
- 52 -comanda POKE nu poate fi folosită pentru mulțimi;
- 53 -matrice prea mare (>64);
- 54 -lipsește END sau ";" într-o declarație de articol;
- 55 -lipsește un identificator de câmp;
- 56 -lipsește variabila de după WITH;
- 57 -variabila de după WITH trebuie să fie de tip articol;
- 58 -identificatorul de câmp nu e precedat de WITH;
- 59 -lipsește valoarea întreagă după LABEL;
- 60 -lipsește valoarea întreagă după GOTO;
- 61 -pointer indicînd un câmp eronat;
- 62 -pointer nedefinit;
- 63 -parametrul pentru SIZE trebuie să fie variabilă;
- 64 -se pot face doar teste de egalitate pentru pointeri;
- 67 -singura formă de extragere a întregilor cu 2 caractere este e:m:H;
- 68 -șirurile de caractere nu trebuie să conțină EOL;
- 69 -parametrii instrucțiunilor NEW, MARK și RELEASE trebuie să fie variabile de tip pointer;
- 70 -parametrii instrucțiunii ADDR trebuie să fie variabile.

6. ERORI DE EXECUTIE

- 1 -oprire;
- 2 -dépășire;
- 4 -împărțire la zero;
- 5 -indice prea mare;
- 6 -indice prea mic;
- 7 -eroare într-o rutină matematică;
- 8 -număr prea mare;
- 9 -lipsește un număr;
- 10 -linie prea lungă;
- 11 -lipsește un exponent.

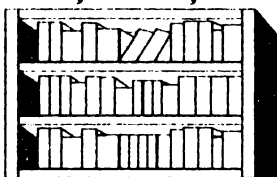
BIBLIOGRAFIE

1. Șerbănați, L-D., Iorga, V., Cristea, V., Moldoveanu, F., *Programarea sistematică în limbajele PASCAL și FORTRAN*, Ed. Tehnică, București, 1984.
2. Ciocârlie, H., Eleș, P., Balla, I., *Limbajele de programare PASCAL și PASCAL CONCURENT*, Ed. Facla, Timișoara, 1985.
3. Crețu, V., *Structuri de date*, I.P. Timișoara, 1987.
4. Wirth, N., *Algorithms and data structures*, New Jersey, 1986.
5. * * * *PASCAL-LANGUAGE HISOFT*, Cheddington, G.B. 1984.
6. * * * *Hisoft Pascal 4TM Implementation Note*.
7. Dupont, R., Rollke, K., Szeliga, M., *Pascal auf dem ZX Spectrum*, Düsseldorf, 1984.
8. Czerwinski, *Programmieren in Pascal*, München 1982.
9. * * * *Pascal - manual de utilizare*, CTGET, 1982.
10. Fussinger, M., *Programierkurs Turbo Pascal*, Düsseldorf, 1988.
11. Scharf, I., *PASCAL für anfangen*", R. Oldenburg Verlag Wien München, 1982.

In curs de apariție:

Colecția:

*Biblioteca pentru
elevi și studenți*



- *Algoritmi și programe BASIC*
- *Culegere de probleme în Turbo-Pascal
(la cerere cu floppy-disk)*
- *Limba jul C*
- *Culegere de programe în C
(la cerere cu floppy-disk)*
- *BASIC între teorie și practică*
- *Sisteme de operare interactive*

*Inițiere în
informatică*

A B C

Info

- *Primii pași în programarea
calculatoarelor*

*Pentru
specialiști*

HIGH

info

- *Limba jul PROLOG*
- *Metode și algoritmi de clasificare
și recunoașterea formelor*

ISBN 973-95118-0-5