Limbajul de programare Turbo Pascal si prelucrarea statistica a datelor experimentale

Prof.univ.dr. Anişoara Constantinescu
Facultatea de Fizică a Universităţii din Bucureşti

February 4, 2008

Cuprins

Capitolul 1

INTRODUCERE

Limbajul PASCAL este un limbaj de programare de nivel înalt, proiectat de Niklaus Wirth de la Universitatea Tehnică din Zurich în 1971 şi numit astfel în cinstea lui Blaise Pascal, celebrul matematician şi filozof francez.

TURBO PASCAL conţine o serie de facilităţi faţă de limbajul PASCAL: mediu integrat, ordonare liberă a secţiunilor în partea de declaraţie a programului, etc.

1.1 Editorul Turbo Pascal

Editorul Turbo Pascal se activează tastând **turbo** şi ENTER. Pe ecran apare următoarea figură:

FILE EDIT SEARCH RUN COMPILE ···
program conversie; ...
F1 HELP F2 SAVE F3 OPEN ALT-F9 COMPILE F10 MENU

Primul rând conţine un meniu. El este activat apăsând tasta F10 şi deplasându-ne cu tastele săgeţi la stânga sau la dreapta pe una din componente. Apăsând tasta ENTER se deschide o fereastră cu un submeniu în care

ne deplasăm cu tastele săgeţi în sus sau în jos până la comanda dorită.

Apăsarea tastei ENTER activează comanda selectată din submeniu. De exemplu ieşirea din editor se face tastând F10, deplasându-ne pe FILE, tastând ENTER, deplasându-ne pe EXIT şi tastând ENTER (sau direct cu ALT-X). În rândul de jos sunt date acţiunile unor taste (cel mai des folosite):
F3 OPEN deschide un fişier cu extensia .PAS din directorul curent, F2 SAVE salvează programul în fişierul curent, ALT-F9 COMPILE compilează programul, F10 MENU activează rândul de sus.

Alte taste de interes sunt:
ALT-F5 determină ieşirea temporară din editor pentru a vedea rezultatele unui calcul. Revenirea în editorul TP se face tastând ENTER.
ALT-n determină trecerea în fereastra n ($1 <= n <= 9$)
ALT-F3 determină închiderea ferestrei curente.
CTRL-F9 determină lansarea în execuţie a programului din fereastra curentă.

Între rândul de sus şi cel de jos se află ecranul ca o pagină albă de scris textul programului în Turbo Pascal. Rămânând în editorul TP putem edita mai multe programe cu F10, FILE, NEW; fiecare program este editat într-o pagină şi paginile sunt numerotate de la 1 la 9.

Încercaţi următoarele programe pentru a vă familiariza cu editorul TP.

1.Programul converteşte un unghi dat în radiani într-un unghi în grade, minute, secunde

```
EX1.PAS
 program  conversie;
 const pi=3.14159;
 var gfr, mfr, radian : real; grade, minute, secunde : integer;
 begin
 writeln(' introdu unghiul in radiani'); readln(radian);
while radian > 2*pi do radian:=radian - 2*pi;
gfr:=radian*180.0/pi; grade:=trunc(gfr);
mfr:=(gfr-grade)*60.0; minute:=trunc(mfr);
secunde:=trunc((mfr-minute)*60.0);
writeln(radian,' radiani=',grade,' grade ',minute,' minute ',
secunde,' secunde'); readln;
end.
```

2. Fiind date 10 valori x, programul calculează valoarea medie şi abaterea lor standard.

EX2.PAS

```
program valmedie;
const n=10;
var x : array[1..n] of real; s, xm, sigx : real; i : integer;
begin
writeln(' introdu ', n, ' valori x');
for i:=1 to n do read(x[i]);
s:=0; for i:=1 to n do s:=s+x[i];
xm:=s/n;
s:=0; for i:=1 to n do s:=s+(xm-x[i])*(xm-x[i]);
sigx:=sqrt(s/n); {pentru valori n mari}
writeln(' x=', xm, ' +/-', sigx);
end.
```

3. Programul calculează o rădăcină a ecuaţiei $x^4-9x^3-2x^2+120x-130=0$, în intervalul [a,b] citit, prin metoda înjumătăţirii intervalului.

EX3.PAS

```
program rezec4;
label 10, 20, 30;
const eps=1.E-05;
var a, b, c : real;
function f(x:real):real;
   begin
    f:=sqr(x)*sqr(x)-9*sqr(x)*x-2*sqr(x)+120*x-130.
   end;
BEGIN
20: write(' introdu limitele intrevalului, [a, b] '); readln(a, b);
if a=b then goto 30; {pentru terminarea calculului trebuie introdus a=b}
if f(a)*f(b) > 0 then
   begin
   writeln(' intervalul nu contine o radacina '); goto 20;
   end;
10: c:=(a+b)/2;  if f(c)=0 then begin writeln(' x= ',c); goto 20; end;
if f(a)*f(c) < 0 then b:=c else a:=c;
```

```
 if abs(b-a) < eps then
    begin
    writeln(' x= ',(a+b)/2); goto 20;
    end
    else goto 10;
    readln;
30: end.
```

4. Acest program simplifică o fracţie, a/b, (a şi b sunt numere întregi) prin împărţirea cu cel mai mare divizor comun al numerelor a şi b. Cel mai mare divizor comun al numerelor întregi a şi b se calculează folosind algoritmul lui Euclid.

EX4.PAS

```
 program fractie;
 var a, b, x, y, z : integer;
 procedure cmmdc;
   begin
   if x < y then
    begin
    z:=x; x:=y; y:=z;
    end;
   while y <> 0 do  {impartiri repetate}
    begin
    z:=x mod y; x:=y; y:=z;
    end;
    end;  {end cmmdc}
{programul principal}
 BEGIN
 writeln(' introdu a si b ca numere intregi, pozitive'); read(a,b);
 writeln(' fractia nesimplificata: ', a , '/', b);
 x:=a; y:=b;  cmmdc;  { apelare procedura cmmdc }
 if x > 1 then  { simplificare }
  begin
    a:=a div x; b:=b div x;
    writeln(' fractia simplificata: ', a, '/',b);
  end
 else writeln(' fractia nu se poate simplifica');
 end.
```

1.2 Unităţi lexicale

Un program în TP poate fi considerat ca un text care specifică acţiunile executate de către un procesor. Acest text este format din caractere grupate în unităţi lexicale; acestea constituie cele mai mici unităţi ale programului. Caracterele (simboluri de bază) folosite în TP sunt:

- literele alfabetului latin: A...Z, a...z şi _(subliniere)
- cifrele arabe: 0...9
- simboluri speciale: + - * / = < > { } [ ] . , : ; # $ blanc
- perechi de caractere: <=, >=, :=, ...

În Turbo Pascal, în scrierea programului, se pot folosi litere mici sau mari; compilatorul nu face distincţie între ele.

Unităţile lexicale ale unui program în Turbo Pascal sunt:

- identificatori
- numere
- şiruri de caractere
- delimitatori
- comentarii

i) identificatori şi cuvinte rezervate

Identificatorii notează etichete, constante, tipuri de date, variabile, funcţii, proceduri, unit-uri, programe şi câmpuri în înregistrări. Un identificator poate avea orice lungime dar sunt semnificative doar primele 63 caractere (primele 8 în Pascal standard). Deşi putem alege oricum identificatorii, e bine ca ei să fie aleşi astfel încât să sugereze semnificaţia mărimilor pe care le desemnează; de exemplu **t** sau **timp** pentru timp, **v** sau **viteza** pentru viteză, **l** sau **lungime** pentru lungime.

Un identificator trebuie să înceapă cu o literă sau cu caracterul _ (subliniere) şi nu poate conţine blancuri. După primul caracter sunt permise litere, cifre şi caractere _ .

Anumiţi identificatori, cunoscuţi ca identificatori standard, sunt pre–declaraţi. Astfel sunt: ABS, ARCTAN, BOOLEAN, CHAR, CHR, COS, EOF, EOLN, EXP, FALSE, GET, INPUT, INTEGER, IN, MAXINT, NEW, ODD, ORD, OUTPUT, PACK, PAGE, PRED, PUT, READ, READLN, REAL, RESET, REWRITE, ROUND, SIN, SQR, SQRT, SUCC, TEXT, TRUE, TRUNC, UNPACK, WRITE, WRITELN, etc. Programatorul poate redefini funcţia lor.

Limbajul Turbo Pascal interzice însă ca anumite cuvinte, numite **cuvinte rezervate**, să fie folosite în alt context decât cel prevăzut la definirea limbajului. Cuvintele rezervate nu pot fi folosite ca identificatori. Ele sunt:
ABSOLUTE, AND, ARRAY, BEGIN, CASE, CONST, DIV, DO, DOWNTO, ELSE, END, EXTERNAL, FILE, FOR, FORWARD, FUNCTION, GOTO, IF, IMPLEMENTATION, IN, INLINE, INTERFACE, INTERRUPT, LABEL, MOD, NIL, NOT, OF, OR, PACKED, PROCEDURE, PROGRAM, RECORD, REPEAT, SET, SHL, SHR, STRING, THEN, TO, TYPE, UNIT, UNTIL, USES, VAR, WHILE, WITH, XOR

ii) numere

Pentru numerele care sunt constante de tip întreg sau real se foloseşte notaţia zecimală obişnuită. Constantele de tip întreg trebuie să fie în domeniul -2147483648...2147483547 (adică $-2^{31}..2^{31}-1$). Constantele zecimale pot fi scrise cu exponent; ele trebuie să fie în intervalul 2.9 E-38..1.7E38. O constantă întreagă hexazecimală foloseşte semnul $ ca prefix. Ea trebuie să fie în domeniul $00000000..$FFFFFFFF (4 octeţi).

iii) şiruri de caractere.

Un şir de caractere este o secvenţă de zero (şirul vid) sau mai multe caractere scrisă pe o linie de program şi inclusă între apostrofuri. Două apostrofuri succesive într-un şir de caractere semnifică un singur caracter, apostroful, ca în exemplul: 'you"ll see'.

iv) delimitatori.

Delimitatorii servesc la separarea unităţilor lexicale. Ei pot fi: unul sau mai multe blancuri, comentarii, semne de punctuaţie, operatori, cuvinte rezervate. Între perechi de identificatori sau numere trebuie să existe cel puţin un separator, spaţiile libere (blancuri) nu pot apare în interiorul unităţilor lexicale cu excepţia şirurilor de caractere.

v) comentarii.

Următoarele construcţii sunt comentarii şi sunt ignorate de compilator:
{ Orice text intre acolade este un comentariu }
(*Orice text ce contine * in dreapta si stanga parantezelor ronde este un comentariu*)

ATENŢIE Un comentariu care începe cu semnul dolar($) imediat după deschiderea cu { sau cu (* este o directivă de compilare. Astfel {$N-} este o directivă de compilare cu generare cod software floating point iar {$N+} este o directivă de compilare cu generare cod 8087 floating point (pentru variabile reale cu precizie înaltă).

vi)etichete.

O etichetă este o succesiune de cifre în domeniul 0..9999 sau un identificator. Zerourile din faţă nu sunt semnificative. Etichetele sunt folosite cu instrucţiuni GOTO.

1.3 Structura programelor în Turbo Pascal

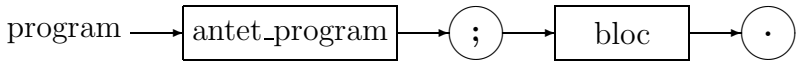
Diagrame de sintaxă.

Reprezentarea sintaxei prin diagrame de sintaxă utilizează ca grafică:

- dreptunghiuri pentru desemnarea categoriilor sintactice
- ovaluri sau cercuri pentru cuvintele cheie sau simbolurile terminale

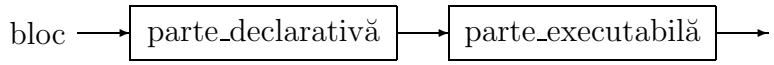
Ca exemplu, structura unui program în Turbo Pascal se poate reprezenta

astfel:

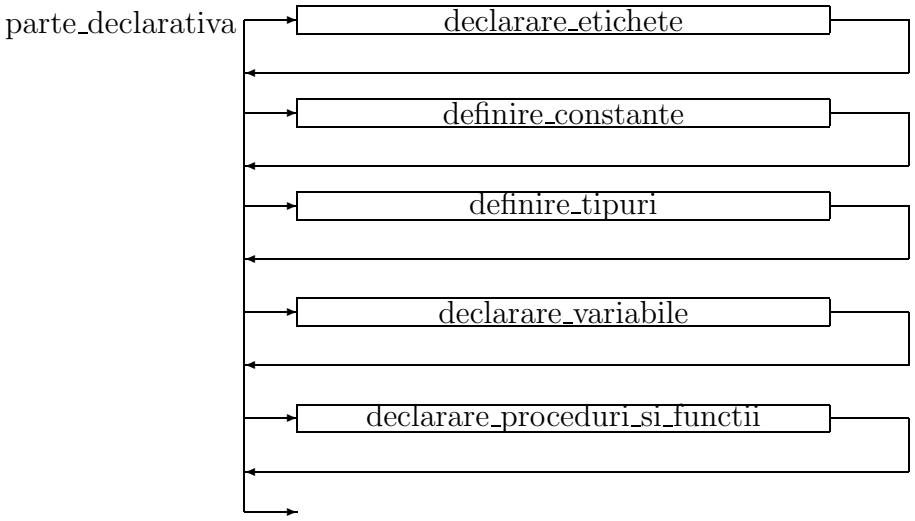


Un program este compus dintr-un bloc precedat de un antet.

Un bloc conţine o parte declarativă care defineşte proprietăţile datelor folosite de program şi o parte executabilă care specifică acţiunile asupra datelor conform algoritmului de calcul.

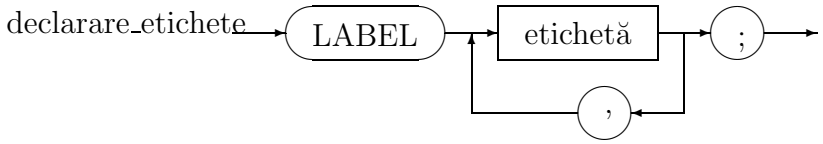


Obiectele (etichete, constante, tipuri de date, variabile, proceduri, funcţii) declarate în partea declarativă a blocului sunt locale acestui bloc.
Dau mai jos diagrama de sintaxă a părţii declarative a unui program în Pascal.



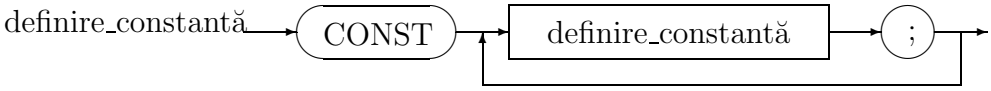
În Turbo Pascal declaraţiile sunt scrise în orice ordine (spre deosebire de Pascal standard unde ordinea declaraţiilor este strictă şi precizată de diagrama de sintaxă de mai sus).

Diagramele de sintaxă pentru fiecare din secţiunile de mai sus sunt:

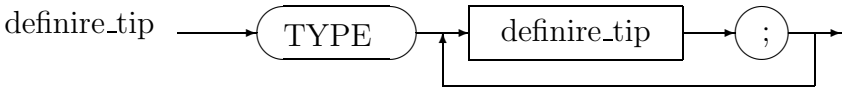


Exemplu: LABEL 1, 5, unu, cinci; { singurele etichete permise in program sunt 1, 5, unu, cinci }

Fiecare etichetă trebuie să marcheze doar o singură instrucţiune.

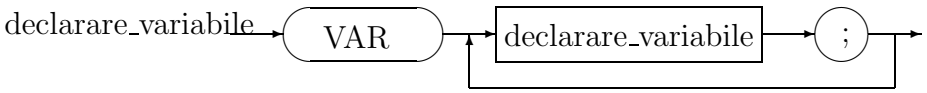


Exemplu: CONST n=10; eps=0.1E-05;



Exemplu:

```
TYPE vara=(iunie, iulie, august);
     litera=''..'Z';
     matricep=ARRAY[1..n, 1..n] OF REAL;
```

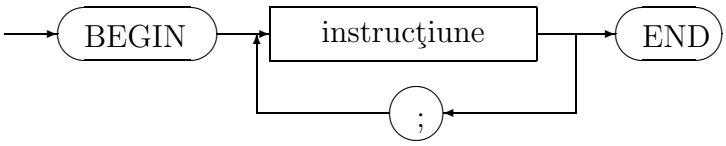


Exemplu:

```
VAR i, j : INTEGER; sezon :vara;
   lit : litera; a, b, c :REAL;
```

Diagrama de sintaxă a părţii executabile a unui program în Turbo Pascal

este:



Exemplu de program simplu în Turbo Pascal:

EX5.PAS

```
PROGRAM cerc;  { antetul programului }
   { calculeaza aria cercului de diametru dat }

   {partea declarativa}
 CONST pi = 3.14158;              { definirea constantei pi }
 VAR diam, arie : REAL;           { declararea variabilelor
                                    diam si arie de tip real }
  {  partea executabila }
BEGIN
 write ( ' introdu diametrul '); { scrie mesaj pe ecranul
                                          de lucru}
 readln ( diam );                 { citire valoare diametrul
                           cercului introdusa de la tastatura }
 arie := pi * SQR(diam)/4.0       { calcul arie }
 writeln(' arie=',arie);          { afisare rezultat pe monitor }
 readln;                          { ramane in ecranul de lucru }

 { tasteaza ENTER pentru a reveni in ecranul de editare }
END.
```

Capitolul 2

TIPURI DE DATE (1)

2.1 Tipuri de date nestructurate predefinite: INTEGER, REAL, CHAR, BOOLEAN

În memoria calculatorului, la nivel de cod maşină, datele se reprezintă ca şiruri de cifre binare. Trecerea de la datele de intrare la această reprezentare binară şi invers, trecerea de la reprezentarea internă a datelor la cea a datelor de ieşire, nu ne interesează în detaliu; pentru calculator informaţia asupra acestei transformări e dată de tipul de date.

Un tip de date defineşte o mulţime finită de valori şi o mulţime finită de operaţii asociate.

Fiecărei date i se asociază un tip unic.

În limbajul PASCAL se pot defini tipuri de date structurate pe baza unor tipuri de date nestructurate şi a unor tipuri definite de programator. Există patru tipuri de date nestructurate reprezentate prin identificatorii de tip predefiniţi: INTEGER, REAL, CHAR, BOOLEAN.

2.1.1 tipul INTEGER

Tipul INTEGER reprezintă o submulţime a mulţimii numerelor întregi dependentă de implementare. În Turbo Pascal există 5 tipuri întregi predefinite şi anume:

Tip	Domeniu	memorie
shortint	-128..127	8-bit cu semn
integer	-32768..32767	16-bit cu semn
longint	-2147483648..2147483647	32-bit cu semn
byte	0..255	8-bit fără semn
word	0..65535	16-bit fără semn

Operaţiile aritmetice cu operanzi de tip întreg folosesc precizia 8-bit, 16-bit sau 32-bit conform următoarelor reguli:

- tipul unei constante de tip întreg este predefinit tip întreg cu domeniul cel mai mic care include valoarea constantei întregi.

- pentru un operator binar, ambii operanzi sunt convertiţi la tipul lor comun înainte de operaţie. Tipul comun este predefinit ca tipul întreg cu cel mai mic domeniu care include toate valorile posibile ale ambelor tipuri. Operaţia este realizată folosind precizia tipului comun şi tipul rezultatului este tipul comun.

- expresia din partea dreaptă a unei instrucţiuni de atribuire este evaluată independent de domeniul şi tipul variabilei din stânga.

- dacă rezultatul operaţiei între valori întregi se situează în afara domeniului reprezentând tipul întreg apare o eroare în faza de execuţie a programului.

- operatorii binari pentru operanzi întregi sunt: +, -, *, DIV, MOD

- funcţii pentru operanzi întregi sunt: ABS, SQR

- operatorii relaţionali sunt: =, <>, >=, <=, >, <

- tipul întreg defineşte o succesiune ordonată de valori deci pentru fiecare valoare (cu excepţia capetelor intervalului) se pot defini un succesor şi un predecesor cu funcţiile:
  SUCC(x)=x+1
  PRED(x)=x-1

2.1.2 tipul REAL

Tipul REAL reprezintă o submulţime finită a numerelor reale. În Turbo Pascal există 5 tipuri reale predefinite dar în directiva de compilare {$N-} selectată implicit, vom lucra doar cu variabile de tip real cărora li se alocă 6 octeţi/variabilă, au domeniul de valori 2.9 E-38..1.7E38 şi 11-12 cifre semnificative.

- operatorii binari pentru tipul REAL sunt: +, -, *, /
- funcţiile standard sunt: ABS, SQR, LN, EXP, SQRT, SIN, COS, ARCTAN
- funcţiile de transfer sunt:
  TRUNC pentru conversia în întreg cu trunchierea părţii fracţionare a argumentului şi
  ROUND pentru conversia în întreg cu rotunjirea părţii fracţionare a argumentului
  ROUND(x) = TRUNC(x+0.5) pentru x>=0
  ROUND(x) = TRUNC(x-0.5) pentru x<0
- pe mulţimea valorilor reale nu este definită o relaţie de ordonare deci nu pot fi folosite funcţiile SUCC şi PRED

2.1.3 tipul CHAR

Tipul CHAR reprezintă o mulţime finită şi ordonată de caractere din setul ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) extins.
Încercaţi programul:

EX6.PAS

```
program caractere ASCII;
 var i, j:integer;
begin
 for i:=1 to 6 do write(i:2,':',chr(i):2,';'); writeln;
 for i:=14 to 23 do write(i:3,':',chr(i):2,';');
 writeln; readln;
 for i:=2 to 24 do
   begin
```

```
   for j:=10*i+4 to 10*i+13 do write(j:3,':',chr(j):2,';');
   writeln;
   end;
 for i:=254 to 255 do write(i:3,':',chr(i):2,';');
 writeln;readln;
end.
```

- Fiecare caracter are o anumită reprezentare internă - valoarea caracterului - şi o anumită poziţie. Din punct de vedere extern o valoare tip caracter se reprezintă prin caracterul respectiv inclus între apostrofuri. Poziţia sau numărul de ordine al unui caracter se obţine cu funcţia ORD. Funcţia inversă este CHR. De exemplu ORD('A') = 65; CHR(65) = 'A'
- Funcţiile standard PRED şi SUCC se definesc astfel:
  PRED(c) = CHR(ORD(c)-1)
  SUCC(c) = CHR(ORD(c)+1)
- Cei 6 operatori relaţionali se pot aplica pentru a compara operanzi de tip caracter obţinându-se rezultate booleene.

2.1.4 tipul BOOLEAN

Tipul BOOLEAN este un tip enumerare cu 2 elemente FALSE <TRUE ceea ce permite aplicarea asupra variabilelor de acest tip a

- operatorilor logici: NOT, AND, OR
- operatorilor relaţionali: =, <>, <=, >=, <, >
- relaţiilor de ordine: ORD(FALSE) = 0; ORD(TRUE) = 1;
  SUCC(FALSE) = TRUE; PRED(TRUE) = FALSE

2.2 Tipuri de date definite de programator: enumerare, interval

Considerând că tipurile standard nu îi sunt suficiente pentru a descrie datele programului, programatorul poate să-şi definească propriile tipuri de date. Dintre acestea prezint aici doar două: tipul enumerare şi tipul interval.

2.2.1 tipul enumerare

Tipul enumerare este descris de enumerarea componentelor sub forma unei liste ordonate de valori pe care le poate lua o variabilă de tipul respectiv.
Exemplu: TYPE culoare=(alb, rosu, galben, verde, albastru);

- Ordinea în care sunt enumeraţi identificatorii în listă defineşte relaţia între componente permiţând aplicarea operatorilor relaţionali precum şi a funcţiilor PRED, SUCC şi ORD (Atenţie: numărul de ordine al primei componente este 0)
  ORD(alb) = 0; ORD(rosu) = 1
  PRED(rosu) = alb; SUCC(alb) = rosu,...

Tipul standard BOOLEAN este echivalent cu tipul enumerare (FALSE, TRUE)

2.2.2 tipul interval

Tipul interval este un subdomeniu de valori dintr-un tip primitiv (cu excepţia celui REAL) sau al unui tip enumerare, denumit tip de bază.

- Definiţia unui interval (sau subdomeniu) specifică valorile cea mai mică şi cea mai mare în interval separate prin simbolul **..** Ambele constante trebuie să fie de acelaşi tip ordinal.
  Exemple:
  0..99 subdomeniu al tipului de bază BYTE
  -128..127 subdomeniu al tipului de bază SHORTINT
  'C'..'R' subdomeniu al tipului de bază CHAR
  rosu..albastru subdomeniu al tipului de bază enumerare definit mai sus

- O variabilă de tip interval are toate proprietăţile variabilelor tipului de bază dar valoarea sa în timpul execuţiei programului trebuie să fie în intervalul specificat.

- Atenţie: o ambiguitate sintactică apare când pentru capetele intervalului se folosesc expresii constante ca în următorul exemplu:
  CONST X=50; Y=10;
  TYPE scala=(x-y)*2..(x+y)*2;

Sintaxa Turbo Pascal spune că dacă o definiţie de tip începe cu o paranteză, tipul este enumerare. Ambiguitatea sintactică de mai sus se poate evita fie scriind
TYPE scala=2*(x-y)..2*(x+y);
fie, când o astfel de variantă nu este posibilă, prin
CONST x=50; y = 10; u = 2*(x-y); v = 2*(x+y);
TYPE scala = u..v;

Tipurile predefinite INTEGER, CHAR şi BOOLEAN şi cele definite, enumerare şi interval, definesc mulţimi finite şi ordonate, motiv pentru care se mai numesc şi tipuri scalare sau ordinale.

Capitolul 3

EXPRESII

O expresie reprezintă o formulă ce defineşte calculul unei valori prin aplicarea unor operatori asupra unor operanzi: constante, variabile, funcţii, mulţimi.

Evaluarea unei expresii se face de la stânga la dreapta respectând nişte reguli de prioritate ale operatorilor. În Turbo Pascal există 4 nivele de prioritate şi anume:

3.1 Operatori, nivele de prioritate

Operatori	Prioritate
@ , not	prima(cea mai înaltă)
$*$, / , div, mod, and, shl, shr	a doua
+, -, or, xor	a treia
=, <>, <, >, <=, >=, in	a patra

Există trei reguli de bază pentru prioritate într-o expresie şi anume:

- un operand aflat între doi operatori de priorităţi diferite este legat de operatorul de prioritate mai înaltă.
- un operand aflat între doi operatori de priorităţi egale este legat de operatorul din stâga sa.
- expresiile din paranteze se evaluează prioritar fiind tratate ca un singur operand.
- intr-o expresie se folosesc numai parantezele ronde, ( ), pentru a schimba prioritatea operaţiilor.

La scrierea expresiilor se iau următoarele precauţii:

- să nu se omită operatorul înmulţirii, $*$, între doi operanzi.
- să nu apară doi operatori consecutivi
  x$*-$y e scrisă greşit
  x$*$(-y) e scrisă corect
- să fie sigur că toţi operanzii reprezentaţi prin variabile au valori definite anterior evaluării expresiei.

OPERATORI ARITMETICI

operator	operaţie	tip operanzi	tip rezultat
+	adunare	integer, real	integer, real
-	scădere	integer, real	integer, real
$*$	înmulţire	integer, real	integer, real
/	împărţire	integer, real	real
div	împărţire întreagă	integer	integer
mod	restul împărţirii	integer	integer

OPERATORI LOGICI

operator	operaţie	tip operanzi	tip rezultat
not	negaţie bit cu bit	integer	integer
and	şi logic bit cu bit	integer	integer
or	sau logic bit cu bit	integer	integer
xor	sau exclusiv bit cu bit	integer	integer
shl	deplasare stânga	integer	integer
shr	deplasare dreapta	integer	integer

- dacă operandul operatorului **not** este de un tip întreg, rezultatul este de acelaşi tip întreg
- dacă ambii operanzi ai operatorilor **and**, **or**, **xor** sunt de câte un tip întreg, tipul rezultatului este tipul comun al celor doi operanzi
- operaţiile **i shl j** şi **i shr j** deplasează valoarea lui **i** la stânga şi, respectiv, la dreapta cu **j** biţi.

  Tipul rezultatului este acelaşi cu tipul lui **i**.

OPERATORI BOOLEENI

operator	operaţie	tip operanzi	tip rezultat
not	negaţie	boolean	boolean
and	şi logic	boolean	boolean
or	sau logic	boolean	boolean
xor	sau exclusiv	boolean	boolean

OPERATORUL STRING

operator	operaţie	tip operanzi	tip rezultat
+	concatenare	char	şir
		şir	şir

OPERATORI DE RELAŢIE

operator	operaţie	tip operanzi	tip rezultat
=	egal	tipuri simple compatibile, set, string	boolean
<>	neegal	-"-	boolean
<	mai mic	tipuri simple compatibile, string	boolean
>	mai mare	-"-	boolean
<=	mai mic sau egal	-"-	boolean
>=	mai mare sau egal	-"-	boolean
<=	subset al	tipuri set compatibile	boolean
>=	superset al	-"-	boolean
in	membru al	operand stânga:orice tip ordinal,t operand dreapta:setul a cărui bază este compatibilă cu t	boolean

- la compararea tipurilor simple e necesar ca tipurile să fie compatibile; totuşi dacă un operand este de tip real, celălalt poate fi de un tip întreg

- compararea şirurilor se face conform cu ordonarea setului de caractere ASCII. Oricare două valori string pot fi comparate deoarece toate valorile string sunt compatibile. O valoare tip CHAR este compatibilă cu o

valoare tip STRING; când ele sunt comparate valoarea tip CHAR este tratată ca o valoare tip STRING de lungime 1.

FUNCŢII STANDARD: $f : A - - > B$

- funcţii matematice:
  sin, cos, arctan, exp, ln, sqrt, A = {real, integer}, B = {real}
  abs, sqr, A = {real, integr }, B = A
- funcţii de transfer:
  trunc, round, A = {real}, B = {integer}
- funcţii scalare:
  ord, A = {integer, char, enumerare, interval}, B = {integer}
  pred, succ, A = {integer, char, enumerare, interval }, B = A
  chr, A = {integer }, B = {char }

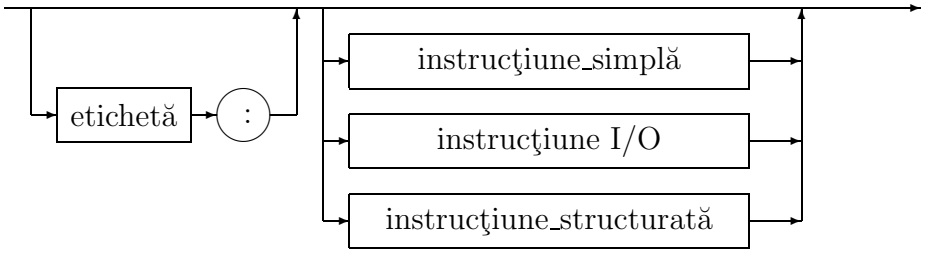
EX7.PAS

```
program operatii;
 var i, j, k, l, m, sl, sr : integer; n, o :byte;
begin
 i := $123; j :=$ff;
 k:= i and j; writeln (i,' and ', j, ' = ', k);
 l:= i or j;  writeln(i,' or ', j,' = ', l);
 m:= i xor j; writeln(i, ' xor ', j, ' = ', m);
 n:=$34; o:= not n; writeln(' not ', n, ' = ', o);
 sr:= $34 shr 4; writeln('shr $34 = ', sr);
 sl:= $34 shl 4; writeln('shl $34 = ', sl);
end.
```

Capitolul 4

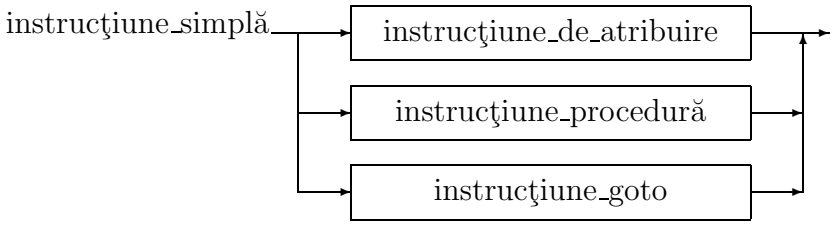
INSTRUCŢIUNI

Diagrama de sintaxă pentru o instrucţiune este următoarea:



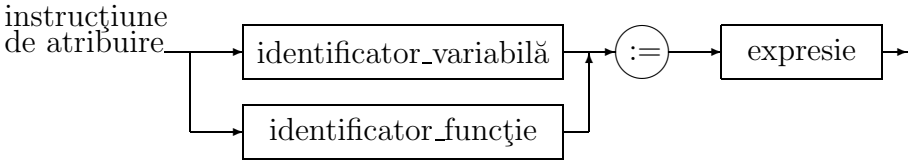
4.1 Instrucţiuni simple

Diagramele de sintaxă pentru instrucţiunile simple sunt următoarele:



4.1.1 instrucţiuni de atribuire

Diagrama de sintaxă pentru instrucţiunea de atribuire este următoarea:

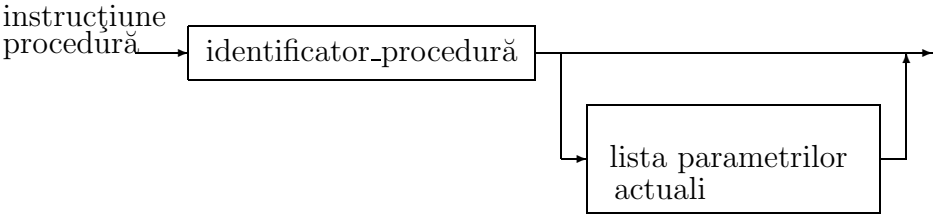


Exemple:
radian := 1.5;
gfr := radian∗180.0/pi; grade := trunc(gfr); (din EX1.PAS)
sau
f := sqr(x)∗ sqr(x) - 9∗ sqr(x)∗ x - 2∗ sqr(x) + 120 ∗x -130.;
(din EX3.PAS)

Instrucţiunea de atribuire înlocuieşte valoarea curentă a unei variabile cu o valoare specificată ca o expresie. Valoarea expresiei trebuie să fie de acelaşi tip sau de un tip compatibil cu tipul variabilei.

4.1.2 instrucţiuni de apelare a unei proceduri

Diagrama de sintaxă a instrucţiunii procedură este următoarea:

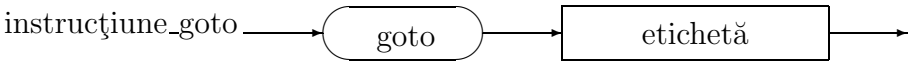


Exemplu **cmmdc** din EX4.PAS

Instrucţiunea procedură specifică activarea unei proceduri notate prin identificatorul procedurii. Dacă declararea procedurii conţine o listă de parametrii formali atunci instrucţiunea procedură trebuie să aibe aceeaşi listă de parametrii actuali.

4.1.3 instrucţiunea goto

Diagrama de sintaxă a instrucţiunii goto este următoarea:



Exemple: **goto 10; goto 20; goto 30** din EX3.PAS

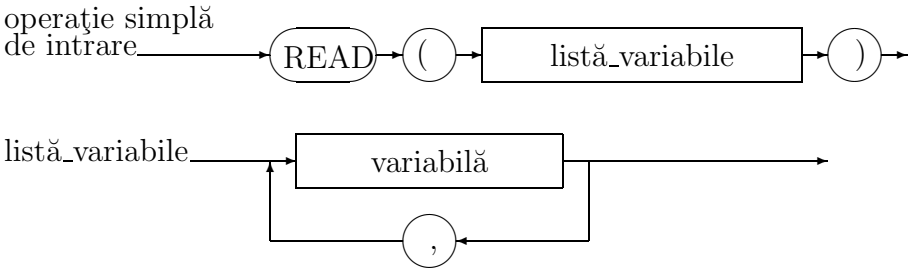
Instrucţiunea **goto** transferă execuţia programului la instrucţiunea ce are ca prefix eticheta referenţiată în instrucţiunea goto. La folosirea instrucţiunii goto se urmăreşte regula ca eticheta referită în instrucţiunea goto să fie obligator în acelaşi bloc ca şi instrucţiunea goto. Cu alte cuvinte nu este posibil a face transfer în sau în afara unei proceduri sau funcţii.

4.2 Instrucţiunile de transfer de date: READ, READLN, WRITE, WRITELN

Într-un program în Turbo Pascal se consideră predeclarate fişierul standard de intrare (INPUT, tastatura) şi fişierul standard de ieşire (OUTPUT, ecranul monitorului).

4.2.1 READ, READLN

Transferul datelor din fişierul standard de intrare în memorie se face prin apelarea procedurii READ conform diagramei de sintaxă:



Variabilele din listă pot aparţine doar tipurilor simple INTEGER, REAL, CHAR. Prin execuţia operaţiei de intrare se preiau valori din fişierul standard de intrare şi se atribuie variabilelor în ordinea dată de lista de variabile.

Datele de intrare sunt considerate de program ca un flux continuu; o operaţie de intrare preia o valoare de pe mediul de intrare din punctul imediat următor ultimei valori preluate prin operaţia de intrare precedentă.

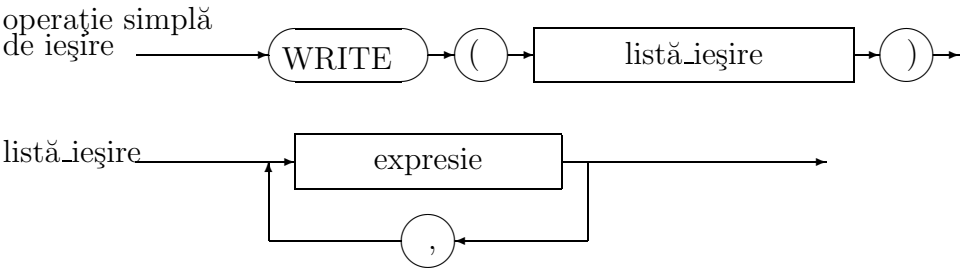
O formă deosebită a operaţiei de intrare are sintaxa:

şi se realizează, după transferul de valori din fişierul de intrare la lista de variabile, o poziţionare la începutul liniei următoare ignorându-se prin aceasta informaţia rămasă în linia curentă.

Procedura READLN fără listă_de_variabile nu realizează un transfer de informaţie ci doar poziţionarea la începutul liniei următoare.

4.2.2 WRITE, WRITELN

Operaţia de ieşire se realizează prin apelarea procedurii WRITE conform diagramei de sintaxă:



Spre deosebire de lista de intrare (care este o listă de variabile) în lista de ieşire pot apare expresii. Ca şi la intrare datele de ieşire sunt transferate în flux continuu. Ele pot fi structurate pe linii folosind procedura WRITELN cu diagrama de sintaxă:

Forma WRITELN (fără listă_ieşire) realizează doar trecerea la linia următoare în timp ce WRITELN(lisă_ieşire) scrie în linia curentă valorile

expresiilor din listă_ieşire după care face trecerea la linia următoare.

- o valoare booleană apare la ieşire sub forma şirului de caractere TRUE sau FALSE

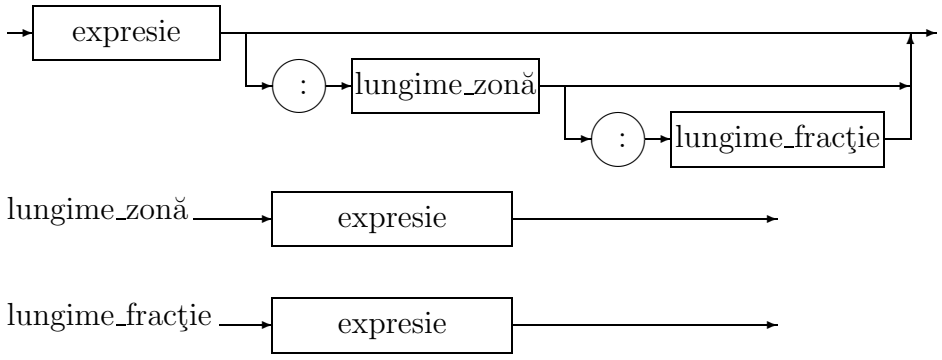
- o valoare întreagă apare ca un şir de cifre precedat, eventual, de semnul - (minus)

- o valoare reală apare la ieşire în virgulă mobilă normalizată conţinând o mantisă şi un exponent.

Operaţia de ieşire permite specificarea numărului de poziţii în care se transferă valoarea de ieşire.

- în cazul în care valoarea de ieşire este de tip INTEGER, CHAR sau BOOLEAN se poate specifica un singur parametru şi anume lungimea totală a zonei ca o expresie întreagă. În acest caz valoarea respectivă va fi plasată în linia de ieşire aliniată la dreapta în zona de lungime specificată. Dacă lungimea zonei este mai mică decât lungimea valorii de tipărit, zona va fi extinsă pentru a afişa întreaga valoare.

- când valoarea de ieşire este de tip REAL sunt necesari doi parametri: lungimea totală a zonei şi lungimea fracţiei. Dacă sunt specificaţi ambii parametri, valoarea reală e scrisă fără exponent, cu punct zecimal şi semn în zona rezervată, aliniată la dreapta, având partea fracţionară rotunjită la numărul de cifre precizat ca lungime a fracţiei. Exemplu: valoarea din memorie x=-15.864 va apare cu WRITE(x:6:1) ca -15.9
  Dacă se specifică doar lungimea zonei, valoarea de tip REAL va fi afişată normalizat având mantisa rotunjită să încapă în lungimea totală a zonei minus 6. Exemplu: WRITE(x:8) afişează, pentru aceeaşi valoare x de mai sus, -1.6 E+01 (unde - este semnul numărului, 1.6 este mantisa iar 01 puterea lui 10 cu care trebuie înmulţită mantisa pentru a avea valoarea din memorie; deci $-1.6 * 10^1$)
  Diagrama de sintaxă este:



EX8.PAS

```
program racheta;
 var ore, min, sec, tzbor : 0..maxint;
begin
 write(' dati ora lansarii rachetei in ore/minute/secunde');
 readln( ore, min, sec );
 writeln('lansare ', ore:2, '/',min:2,'/',sec:2);
 write('dati in secunde durata de zbor a rachetei');
 readln(tzbor);
 writeln('durata de zbor: ',tzbor:6,' secunde');
 sec := sec + tzbor; min := min + sec div 60; sec := sec mod 60;
 ore := ore + min div 60; min := min mod 60; ore := ore mod 24;
{ afisare moment sosire }
 writeln(' sosire: ',ore:2, '/',min:2,'/',sec:2);
end.
```

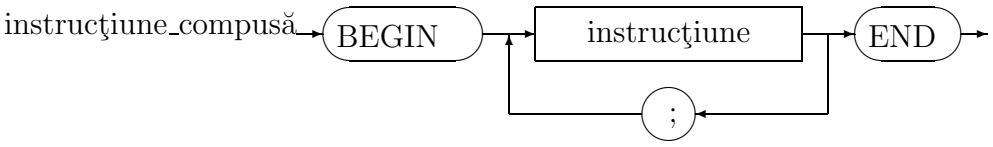
Datele, întregi, cerute de acest program se introduc fie câte una pe un rând (valori numerice pentru oră ENTER, minut ENTER, secundă ENTER) fie toate valorile numerice separate cu blanc pe un singur rând.

4.3 Instrucţiuni structurate

Instrucţiunile structurate sunt construcţii formate din alte instrucţiuni executate fie secvenţial (în instrucţiunea compusă) fie condiţional (în instrucţiunile condiţionale) fie iterativ (în instrucţiunile repetitive).

4.3.1 instrucţiunea compusă

Instrucţiunea compusă reprezintă o secvenţă de instrucţiuni considerată ca un tot şi executată în ordinea în care sunt scrise instrucţiunile în secvenţă. Instrucţiunile din secvenţă sunt separate prin **;** şi încadrate în parantezele de instrucţiune BEGIN şi END conform diagramei de sintaxă:



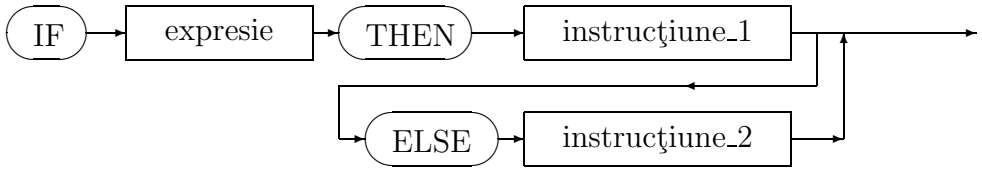
Instrucţiunea compusă este tratată sintactic ca o singură instrucţiune. Partea executabilă a unui program poate fi astfel considerată ca o singură instrucţiune compusă.

4.3.2 instrucţiunea condiţională

O instrucţiune condiţională selectează pentru execuţie o singură instrucţiune (sau nici o instrucţiune) din instrucţiunile sale componente.

instrucţiunea IF

Diagrama de sintaxă a instrucţiunii IF este următoarea:



în care **expresie** este o expresie booleană ce trebuie să dea un rezultat de tip boolean. Dacă rezultatul evaluării expresiei este **TRUE** se execută **instrucţiune1**. Dacă rezultatul evaluării expresiei este **FALSE** şi este prezent **ELSE** se execută **instrucţiune2**. Dacă rezultatul evaluării expresiei este **FALSE** şi nu este prezent **ELSE** atunci se trece la instrucţiunea următoare instrucţiunii **IF**.

Atenţie: delimitatorul **;** nu trebuie pus înaintea lui ELSE deoarece ar duce la terminarea deciziei fără a considera şi cea de a doua alternativă.

- dacă **instrucţiune1** sau/şi **instrucţiune2** sunt formate din mai multe instrucţiuni, ele se reprezintă ca instrucţiuni compuse
- în general un **ELSE** este asociat cu cel mai apropiat **IF** neasociat deja cu un alt **ELSE**. Ambiguitatea sintactică ce apare din construcţii ca: **if expresie1 then if expresie2 then instrucţiune1 else instrucţiune2** se rezolvă interpretând construcţia astfel:

```
if expresie1 then
    begin
      if expresie2 then
        instructiune1
      else
        instructiune2
    end
```

(nu era necesară paranteza begin..end) sau, în cazul următor necesitând pentru interpretarea corectă paranteza begin..end, :

```
if expresie1 then
    begin
        if expresie2 then instructiune1
    end
else
    instructiune2
```

Exemple de folosire a instrucţiuni IF

1. se consideră trei valori reale pozitive a<=b<=c. Să se stabilească dacă ele pot reprezenta laturile unui triunghi şi, în caz afirmativ, să se precizeze natura triunghiului (oarecare, dreptunghi, isoscel sau echilateral).
Condiţia ca numerele a, b, c, care satisfac condiţia de mai sus, să formeze laturile unui triunghi este ca ($c < a + b$)

EX9.PAS

```
PROGRAM triunghi;
  {stabileste natura triunghiului format cu trei
   numere date ca laturi a<=b<=c}
  VAR a, b, c : REAL;
```

```
      echl, isos, drept : BOOLEAN;
BEGIN
   WRITE (' introdu 3 valori reale si pozitive, a<=b<=c');
   READLN(a, b, c);
   WRITELN(a:6:2,' ', b:6:2,' ', c:6:2);
   IF c < a + b  THEN
    begin
    WRITE( ' numerele date formeaza un triunghi ');
        echl := (a=b) AND (a=c);
        isos := (a=b) OR (a=c) OR (b=c);
        drept := c*c = a*a + b*b;
    IF echl THEN WRITELN(' echilateral')
     ELSE  IF isos THEN WRITELN('isoscel')
            ELSE  IF drept THEN WRITELN ('dreptunghi')
                   ELSE  WRITELN ('oarecare')
    end
   ELSE WRITELN('numerele date nu formeaza un triunghi');
  READLN;
END.
```

2. fiind citită o literă să se scrie dacă ea este o vocală sau o consoană.

EX10.PAS

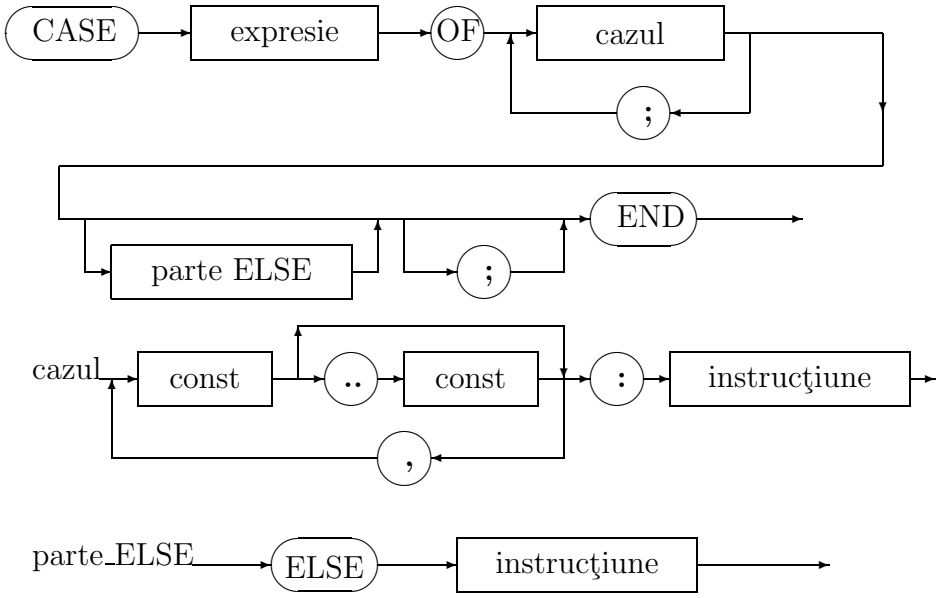
```
PROGRAM vocala_consoana;
  VAR c:CHAR;
BEGIN
    WRITE('introdu o litera'); READLN(c);
     IF(c='a') OR (c='A') OR (upcase(c)='E') OR (upcase(c)='I')
       OR (upcase(c)='O') OR (upcase(c)='U')
     THEN WRITELN(c, ' este o vocala')
     ELSE WRITELN(c, 'este o consoana');
END.
```

3. fiind citit un şir de cifre reprezentând un număr întreg şi pozitiv, să se scrie şirul de cifre ce reprezintă numărul răsturnat. De exemplu, fiind introdus numărul 5413, rezultatul va fi 3145.
EX11.PAS

```
PROGRAM calcul_inversare_cifre;
  LABEL 1;
  VAR m: INTEGER;
BEGIN
WRITE(' introdu un numar intreg si pozitiv'); READLN(m);
    1:WRITE(m mod 10 :2); m := m div 10; IF m <> 0 THEN goto 1;
WRITELN; READLN;
END.
```

instrucţiunea CASE

Diagrama de sintaxă a instrucţiunii CASE este următoarea:



Instrucţiune CASE permite alegerea şi execuţia unei singure instrucţiuni dintre mai multe instrucţiuni componente în funcţie de valoarea unei expresii scalare (**expresie** din diagrama de sintaxă) numită **expresie selector**.

Instrucţiunile componente sunt precedate de una sau mai multe constante distincte ce au valori de acelaşi tip simplu (INTEGER, CHAR, BOOLEAN, enumerare, interval) ca şi tipul expresiei selector. Ordinea acestor 'etichete' este arbitrară.

Expresia selector, de tip ordinal, poate lua valori între -32768 şi 32767 şi deci nu poate fi de tip LONGINT, WORD sau STRING.

Instrucţiunea CASE execută instrucţiunea prefixată de o constantă CASE egală cu valoarea expresiei selector sau prefixată de un domeniu CASE ce conţine valoarea expresiei selector. Dacă nu există o astfel de constantă CASE şi este prezentă partea_ELSE se execută instrucţiunea ce urmează pe ELSE. Dacă nu există partea_ELSE nu se execuă nici-o instrucţiune.

Exemple
1. Fiind introdus de la tastatură un caracter să se scrie dacă este o literă mare, o literă mică, un blanc sau un alt caracter.

EX12.PAS

```
PROGRAM testcase1;
  VAR c : CHAR;
  BEGIN
  WRITE(' introdu un caracter oarecare: '); READLN(c);
  CASE c OF
    'A'..'Z' : WRITELN(' litera mare');
    'a'..'z' : WRITELN(' litera mica');
    ' '      : WRITELN(' blanc');
{aici poate apare ; spre deosebire de instructiunea IF cu ELSE}
  ELSE WRITELN(' alt caracter ');
  END;  {end CASE}
  READLN;
  END.
```

2. Fiind citită sub forma zi, lună, an (în cifre) data zilei de azi să se scrie data zilei de mâine.

EX13.PAS

```
PROGRAM maine;
  {calculeaza data zilei de maine din data zilei de azi}
VAR an : 2000..2100;
```

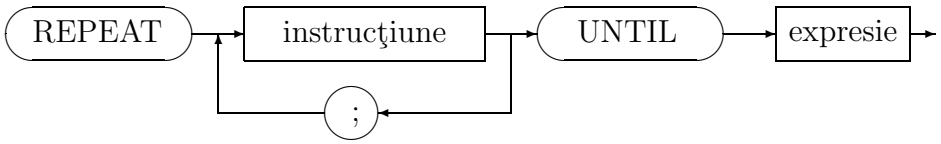
```
    luna : 1..12; zi : 1..31; ultima_zi : 29..31;
    bisect : boolean;
BEGIN
  WRITE( ' introdu in cifre zi, luna, an, data zilei de azi ');
  READLN(zi, luna, an); WRITELN('azi: ', zi, '/',luna,'/',an);
  {bisect ia valoarea TRUE daca anul este bisect}
bisect := (an MOD 4 = 0) AND ( an MOD 100 <> 0) OR (an MOD 400
  = 0);
  { gasirea ultimei zile din luna respectiva}
CASE luna OF
1, 3, 5, 7, 8, 10, 12 : ultima_zi := 31;
4, 6, 9, 11 : ultima_zi := 30;
2 : IF bisect THEN ultima_zi := 29 ELSE ultima_zi := 28;
END;  {end CASE}
{ gasirea datei zilei de maine}
IF zi < ultima_zi THEN zi := zi + 1 ELSE
  IF zi > ultima_zi THEN WRITELN ('data incorecta') ELSE
      begin
      zi := 1; {luna urmatoare}
      IF luna = 12 THEN
          begin
          luna := 1; {anul urmator}
          an := an + 1;
          end
      ELSE luna := luna + 1;
      end;
WRITELN(' maine: ', zi, '/', luna,'/', an);
END.
```

4.3.3 instrucţiuni repetitive

Instrucţiunea repetitivă specifică faptul că anumite instrucţiuni urmează a se executa repetat. Dacă numărul de repetări e cunoscut dinainte este folosită instrucţiunea FOR altfel vor fi folosite instrucţiunile WHILE sau REPEAT.

instrucţiunea REPEAT...UNTIL

Diagrama de sintaxă a instrucţiunii REPEAT este următoarea:



în care:
expresie controlează repetarea execuţiei succesiunii de instrucţiuni din cadrul instrucţiunii REPEAT. Expresie trebuie să dea un rezultat de tip BOOLEAN.

Instrucţiunile între cuvintele rezervate ale limbajului Pascal, REPEAT şi UNTIL, se execută în succesiune repetat până când **expresie** evaluată dă rezultatul TRUE.

Succesiunea de instrucţiuni este executată cel puţin o dată deoarece **expresie** este evaluată după execuţia succesiunii de instrucţiuni.

Exemple:
1. secvenţa de program ce calculează cel mai mare divizor comun între două numere întregi, i şi j, cu algoritmul lui Euclid:

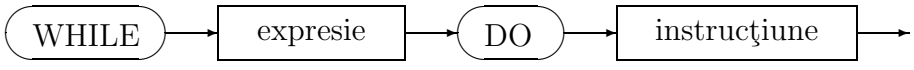
```
REPEAT
  k := i MOD j; i := j; j := k;
UNTIL j=0;
```

2. secvenţa de program ce cere introducerea unui număr între 0 şi 9:

```
REPEAT
  WRITE(' introdu o valoare in intervalul 0..9');
  READLN(i);
UNTIL (i >= 0) AND (i <= 9);
```

instrucţiunea WHILE...

Diagrama de sintaxă a instrucţiunii WHILE este următoarea:



şi conţine o **expresie** care controlează repetarea execuţiei unei **instrucţiuni** (care poate fi o instrucţiune compusă). **Expresie** trebuie să fie de tip BOOLEAN şi este evaluată înainte ca **instrucţiune** să fie executată. **Instrucţiune** este executată repetat cât timp **expresie** dă rezultatul TRUE; dacă **expresie** dă FALSE la început , **instrucţiune** nu se execută nici o dată.

Exemple:

1. o secvenţă de program ce contorizează în variabila, **nr**, numărul de blancuri alăturate dintr-un şir introdus de la tastatură;

```
nr := 0; READ(c); WHILE c = ' ' DO
                        begin
                        nr := nr + 1;
                        READ(c);
                        end;
```

2. secvenţa de program ce calculează cel mai mare divizor comun al numerelor x si y prin algoritmul lui Euclid (EX4.PAS)

```
WHILE y <> 0 DO  {impartiri repetate}
  begin
    z := x mod y; x:=y; y:=z;
  end;
```

Exemple de programe ce folosesc şi instrucţiunile repetitive REPEAT şi WHILE

1. Fiind citit un număr întreg, **n** să se stabilească dacă el reprezintă un palindrom adică dacă el este egal cu numărul obţinut citindu-l pe **n** de la dreapta la stânga. De exemplu n=121 este un palindrom dar n=122 nu este.

Algoritm: se repetă compararea cifrelor cea mai semnificativă şi cea mai puţin semnificativă ale numărului fie până când cele două cifre diferă fie până când s-a efectuat numărul suficient de comparaţii; în primul caz numărul nu este un palindrom.

EX14.PAS

```
PROGRAM palindrom;
```

```
TYPE cifra=0..9;
VAR n, r, mare : INTEGER; p, q :cifra;
BEGIN
WRITE (' introdu un numar intreg, n='); READLN(n); WRITE(n);
r := n; mare := 1;
WHILE r >= 10 DO
  begin
      mare := mare * 10;
  { mare = 10 la puterea (m-1), m = numar de cifre in n}
      r := r DIV 10;
  end;
IF mare = 1 THEN WRITE (' este ')
{ orice numar dintr-o cifra este palindrom}
ELSE
  begin
    REPEAT
      p := n DIV mare; { separa cifra cea mai semnificativa }
      q := n MOD 10; { separa cifra cea mai putin semnificativa}
      n := n MOD mare; { suprima cifra cea mai semnificativa }
      n := n DIV 10; { suprima cifra cea mai putin semnificativa}
      mare := mare DIV 100;
    UNTIL (p <> q) OR ( mare < 10);
    IF p = q THEN WRITE (' este ') ELSE WRITE(' nu este ');
  end;
WRITELN(' palindrom ');
END.
```

2. Să se calculeze valoarea integralei:

$$I(a,b) = \frac{2}{\pi}\int_0^{\pi/2} \frac{dx}{\sqrt{a^2cos^2(x) + b^2sin^2(x)}}$$

pentru **a** şi **b** date ştiind că această integrală reprezintă limita comună a şirurilor $\{u_n\}$ şi $\{v_n\}$ generate de relaţiile de recurenţă:

$$u_n = (u_{n-1} + v_{n-1})/2,\ v_n = \sqrt{u_{n-1} * v_{n-1}}$$

pornind cu:

$$u_0 = \frac{1}{|a|},\ v_0 = \frac{1}{|b|}$$

Calculul se opreşte când

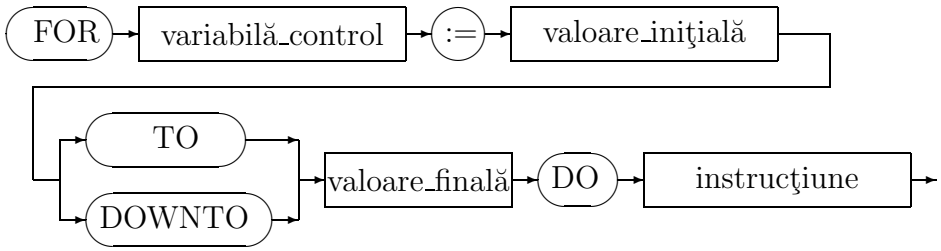
$$|u_n - v_n| < \epsilon \qquad \text{cu} \qquad \epsilon = 1.0E - 05$$

EX15.PAS

```
PROGRAM val_integrala_sir;
  VAR a, b, u, v, u1 :REAL; i : INTEGER;
  CONST eps=1.E-05;
BEGIN
  WRITE(' introdu a, b: '); READLN(a, b);
  u := 1/ABS(a); v := 1/ABS(b);
  REPEAT
  u1 := u; u := (u + v) / 2; v := SQRT(u1 * v);
  inc(i);
  UNTIL ABS( u - v ) < eps;
  WRITELN(' ulim =', u, ' vlim=', v, ' la ',i,' iteratii');
  END.
```

instrucţiunea FOR...

Diagrama de sintaxă a instrucţiunii FOR este următoarea:



în care:
variabilă_control este un identificator de variabilă iar **valoare_iniţială** şi **valoare_finală** sunt câte o expresie.

- Instrucţiunea FOR face ca **instrucţiune** (care poate fi o instrucţiune compusă) să se execute repetat cât timp variabilei de control i se atribuie o progresie de valori.

- Variabila de control trebuie să fie de unul din tipurile ordinale şi **valoare_iniţială** şi **valoare_finală** trebuie să fie de un tip compatibil din punct de vedere atributiv cu tipul variabilei de control.

- Când se intră într-o instrucţiune FOR valorile iniţială şi finală sunt determinate o dată pentru tot restul execuţiei instrucţiunii FOR. Instrucţiunea se execută o dată pentru fiecare valoare a variabilei de control.

- Variabila de control începe întotdeauna cu valoarea iniţială. Când instrucţiunea FOR foloseşte TO valoarea variabilei de control creşte cu unu la fiecare repetare. Dacă valoarea iniţială este mai mare decât valoarea finală, **instrucţiune** nu se execută. Când instrucţiunea FOR foloseşte DOWNTO, valoarea variabilei de control scade cu unu la fiecare repetare. Dacă valoarea iniţială este mai mică decât valoarea finală, **instrucţiune** nu se execută.

Notă: este eroare dacă **instrucţiune** schimbă valoarea variabilei de control.

Exemple de programe ce folosesc instrucţiunea FOR

1. programul val_medie din EX2.PAS
2. fiind citit un număr întreg n să se calculeze suma $s = 1+1/2+1/3+...+1/n$

EX16.PAS

```
PROGRAM eval_sir;
  { evaluare suma 1/1 + 1/2 +...+ 1/n, n fiind citit}
VAR i, n : INTEGER; s : REAL;
BEGIN
  WRITE(' introdu n '); READLN( n );
  s := 0; FOR i:= n DOWNTO 1 DO s := s + 1/i;
  WRITELN(' suma 1/1 + 1/2 + ... + 1/',n,' = ',s);
END.
```

Alte exemple

1. Fiind dat un număr sub forma unui şir de caractere hexazecimale, să se găsească reprezentarea sa zecimală.

EX17.PAS

```
PROGRAM conv_hex_zec;
  { calculeaza valoarea zecimala a unui sir de caractere
  hexazecimale, numar intreg}
VAR c:CHAR; d:INTEGER; val:0..15;
  BEGIN
WRITELN(' introdu numarul in forma hexazecimala, litere mari');
d:=0; READ(c);
WRITE(c); WHILE ( c >= '0') AND ( c <= '9' ) OR
                (c>= 'A') AND (c <= 'F') DO
                begin
                    IF (c >= '0') AND ( c <= '9') THEN
                        val := ORD(c) - ORD('0')
                    ELSE
                        CASE c OF
                        'A': val := 10;
                        'B': val := 11;
                        'C': val := 12;
                        'D': val := 13;
                        'E': val := 14;
                        'F': val := 15;
                        end;  {end CASE}
                d := 16 * d + val; { contributia cifrei hexa }
                READ(c); {citeste urmatorul caracter}
                WRITE(c);
                end; { end instructiunea compusa din WHILE}
WRITELN; { trece la linia urmatoare }
  WRITELN(d) { scrie numarul in reprezentarea zecimala}
END.
```

2. Un program care afişează toate numerele pitagorice până la un număr dat, n, (n $<$ 40) excluzând dublele.

EX18.PAS

```
PROGRAM numere_pitagorice;
  VAR i, j, n : BYTE; v, z : REAL:
  BEGIN
```

```
  i := 1; WRITE(' introdu n < 40, n='); READLN(n);
  WHILE i <= n DO
  begin
 j :=i; {pentru a elimina numerele duble, de exemplu 3, 4, 5 si 4, 3, 5}
      WHILE j <= n DO
          begin
          v := SQR(i) + SQR(j); z := SQRT(v);
           IF z > 100.0 THEN j := 100 ELSE
            IF FRAC(z) = 0.0 THEN WRITELN(i:5, j:5, '-->', z:5:0);
            j := j + 1;
           end;
   i := i + 1;
   end;
END.
```

3. Un program care afişează divizorii unui număr şi numărul acestora. Dacă numărul este prim se afişează un mesaj corespunzător.

EX19.PAS

```
PROGRAM divizori;
  VAR contor : BYTE;
      i, nr : WORD;
BEGIN
  WRITELN; contor := 0;
  WRITE(' introdu numarul: '); READLN(nr);
  FOR i := 2 TO nr DIV 2 DO { s-au exclus 1 si numarul insusi}
      IF nr/i = nr DIV i THEN { se verifica daca i divide nr}
          begin
          inc(contor); WRITELN(i);
          end;
  IF contor = 0 THEN WRITELN (nr, ' este numar prim')
  ELSE WRITELN( nr, ' are', contor, ' divizori');
  END.
```

4. Un program care afişează numerele prime cuprinse între a şi b precum şi numărul lor.

EX20.PAS

```
 PROGRAM numere_prime;
   LABEL salt;
   CONST nr : WORD = 0;
   VAR a, b, i, j :WORD;
 BEGIN
   WRITE(' a='); READLN(a); WRITE(' b='); READLN(b);
   WRITELN;
   WRITELN(' numere prime cuprinse intre ', a, ' si ',b);
   FOR i := a TO b DO
     begin
     FOR j := 2 TO i DIV 2 DO
       IF i/j = i DIV j THEN goto salt;
       WRITELN(i); inc(nr);
salt: end;
WRITELN;
WRITELN(' avem ', nr, ' numere prime intre ', a, ' si ',b);
END.
```

5. Un program care scrie anotimpurile

```
EX1300.PAS
program ex_case;
type an=(ian,feb,mar,apr,mai,iun,iul,aug,sep,oct,nov,dec);
var luna:an;
begin
for luna:=ian to dec do
case luna of
ian,feb,dec: writeln('iarna ');
mar,apr,mai: writeln('primavara ');
iun,iul,aug: writeln('vara ');
sep,oct,nov: writeln('toamna ');
end;
readln;
end.
```

Capitolul 5

TIPURI DE DATE (2)

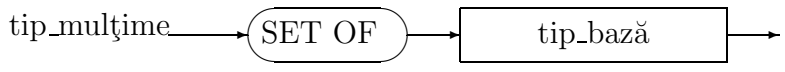
Tipuri structurate de date sunt : **set, array, string, record, file**

Un tip structurat de date este caracterizat prin tipul (tipurile) componentelor şi prin metoda structurării sale.

5.1 Tipul set(mulţime)

Tipul **set** se defineşte în raport cu un tip de bază. Tipul de bază trebuie să fie un tip ordinal: CHAR, BYTE, BOOLEAN, enumerare, interval. Cu toate că tipurile întregi sunt ordinale nu este permis ca tip de bază decât BYTE. Dându-se un asemenea tip de bază, valorile posibile ale tipului set sunt formate din mulţimea tuturor submulţimilor posibile ale tipului de bază, inclusiv mulţimea vidă.
Notă: dacă tipul de bază are **n** valori, tipul set va avea 2^n valori cu restricţia n<=255. Numărul maxim de elemente pentru o mulţime este 256 iar valorile ordinale asociate aparţin domeniului 0..255. De aici restricţia de a lua dintre tipurile întregi doar tipul BYTE.

Diagrama de sintaxă a definirii acestui tip este următoarea:



Exemple:

```
TYPE litera=(A, B, C);
        lit= SET OF litera;
VAR m : lit;
```

m va putea lua ca valori una din următoarele mulţimi:

```
[ ], [A], [B], [C], [A, B], [A, C], [B, C], [A, B, C]
```

```
 TYPE examen=(analiza, mecanica, fizica, chimie);
      exprom=SET OF examen;
 VAR stud1, stud2 : exprom;
```

valorile variabilelor stud1, stud2 pot fi mulţimea vidă [ ] (nici-un examen promovat), un examen promovat [analiza] sau [mecanica] sau [fizica] sau [chimie], două examene promovate [analiza, mecanica] sau orice altă combinaţie de 2 materii din cele din tipul de bază, trei examene promovate [analiza, mecanica, fizica] sau orice altă combinaţie de 3 materii sau 4 examene promovate [analiza, mecanica, fizica, chimie]. De exemplu:
stud1 := [analiza, fizica];
stud2 := [ ];

- o valoare tip mulţime poate fi specificată printr-un **constructor** (generator) de mulţime. Un constructor conţine specificarea elementelor separate prin virgulă şi închise între paranteze pătrate. Un element poate fi o **valoare** precizată sau un **interval** de forma inf..sup, unde valorile inf şi sup precizează valorile inferioară şi superioară. Atât valoare cât şi limitele de interval pot fi expresii. Dacă sup < inf nu se generează nici-un element.
- operaţiile cu valori tip **set** sunt:

```
+  reuniune      o valoare de tip ordinal c este in a + b
                    daca c este in a sau in b
```

```
-  diferenta     o valoare de tip ordinal c este in a - b
                    daca ea este in a si nu este in b
*  intersectie   o valoare de tip ordinal c este in a * b
                    daca ea este in a si in b
```

- relaţiile referitoare la mulţimi:
  dacă a şi b sunt operanzi de tip set atunci

```
a = b        este adevarata numai daca a si b contin exact
                  aceleasi elemente; altfel a <> b.
a <= b       este adevarata daca fiecare element al lui a
                  este de asemeni un element al lui b
a >= b       este adevarata daca fiecare element al lui b
                  este de asemeni un element al lui a
x in a       este adevarata daca x apartine lui a. Daca x
                  este de tipul ordinal t, a este de tip
                  multime cu tipul de baza compatibil cu t
```

Exemplu Fiind citite elementele a două mulţimi **m1** şi **m2** (ce au tipul de bază BYTE) să se scrie elementele mulţimilor **m1+m2, m1*m2, m1-m2**.

EX21.PAS

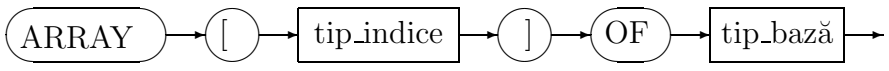
```
PROGRAM setop;
  TYPE mult=SET OF BYTE;
  VAR m1, m2, int, un, dif : mult;
      nrm1, nrm2, x, i:BYTE;
 BEGIN
 WRITE(' introdu numarul de elemente din multimea m1, nrm1=');
 READLN(nrm1); m1 := [ ];
 FOR i := 1 TO nrm1 DO
    begin
      write(' elementul tip BYTE ',i,' este:'); readln(x);
      m1 := m1 + [x];
    end;
WRITE(' introdu numarul de elemente din multimea m2, nrm2=');
 READLN(nrm2); m2 := [ ];
 FOR i := 1 TO nrm2 DO
```

```
    begin
      write(' elementul tip BYTE ',i,' este:'); readln(x);
      m2 := m2 + [x];
    end;
{calculul elementelor multimilor intersectie,
                                  reuniune, diferenta}
int := m1 * m2;  un := m1 + m2;  dif := m1 - m2;
    {scrierea rezultatelor}
WRITE(' Operandul stang este: [');
  FOR i := 0 TO 255 DO IF i in m1 THEN write(i,',');
  WRITELN(']');
WRITE(' Operandul din dreapta este: [');
  FOR i := 0 TO 255 DO IF i in m2 THEN write(i,',');
  WRITELN(']');
WRITE(' m1 ',#239, ' m2 = [');
  FOR i := 0 TO 255 DO IF i in int THEN write(i,',');
  WRITELN(']');
WRITE(' m1 U m2 = [');
  FOR i := 0 TO 255 DO IF i in un THEN write(i,',');
  WRITELN(']');
WRITE(' m1 - m2 = [');
  FOR i := 0 TO 255 DO IF i in dif THEN write(i,',');
  WRITELN(']');
END.
```

5.2 Tipul array (tablou)

5.2.1 Tablouri cu o dimensiune

Tipul tablou este o structură omogenă formată dintr-un număr fix de componente de acelaşi tip numit tip de bază.
Diagrama de sintaxă ce defineşte acest tip este:



în care tip_indice este un tip ordinal iar tip_bază este orice tip.

Tabloul este o structură cu acces direct, timpul de acces la orice componentă a tabloului fiind acelaşi.

Fiecare componentă a unui tablou este selectată printr-un indice care ia valori într-o mulţime finită numită tipul indicelui. Tipuri_indice valabile sunt toate tipurile ordinale cu excepţia lui LONGINT şi a subdomeniilor de LONGINT; tip_indice nu poate fi un tip structurat. **Exemple:**

```
TYPE vector=array[1..10] of real;  {defineste tipul structurat
             vector reprezentand multimea tablourilor ce au 10
             componente de tip real}
VAR u, v : vector;
```

Tipul de bază poate fi orice tip nestructurat sau structurat.

Accesarea unei componente a tabloului se face prin identificatorul tabloului şi valoarea indicelui sau o expresie al cărui rezultat trebuie să se afle în domeniul definit de tip_indice.

5.2.2 Tablouri cu mai multe dimensiuni

Componentele unui tablou pot fi de orice tip, inclusiv de tip tablou. Astfel dacă tip_bază este un tip tablou se poate trata rezultatul ca un tablou de tablouri sau ca un tablou cu mai multe dimensiuni. Astfel
ARRAY [BOOLEAN] OF ARRAY[1..10] OF ARRAY[1..2] OF REAL;
este interpretat de compilator ca
ARRAY [BOOLEAN, 1..10, 1..2] OF REAL;
Accesarea unui element al tabloului multidimensional se face specificând valorile indicilor din parantezele drepte. Astfel
VAR a:ARRAY[1..10] OF ARRAY[1..20] OF REAL;
defineşte o matrice cu 10 linii şi 20 coloane. Elementul din linia i şi coloana j se identifică prin a[i][j] sau a[i, j].

5.2.3 Constantele tablou multidimensionale

Constantele tablou multidimensionale sunt definite prin includerea constantelor fiecărei dimensiuni în seturi separate de paranteze, separate prin virgulă. Constantele cele mai interne corespund dimensiunii din extrema dreaptă.

Astfel:

```
TYPE cub=array[0..1, 0..1, 0..1] of integer;
  CONST tabel : cub = (((0, 1),(2,3)),((4, 5), (6, 7)));
```

dă pentru tabel iniţializarea la:
tabel [0, 0, 0] = 0 tabel [0, 0, 1] = 1
tabel [0, 1, 0] = 2 tabel [0, 1, 1] = 3
tabel [1, 0, 0] = 4 tabel [1, 0, 1] = 5
tabel [1, 1, 0] = 6 tabel [1, 1, 1] = 7
sau
CONST cifra : array[0..9] OF char = ['0','1','2','3','4','5','6','7','8','9'];
sau
CONST cifra : array[0..9] OF char='0123456789';
dă pentru cifra iniţializarea la
cifra [0] = '0'; cifra [1] = '1'; cifra [2] = '2';...

Exemple de programe ce folosesc tipul **ARRAY**:

1. Fiind dat un şir de valori întregi x[i], i=1,n, $n <= 30$, să se găsească xmin şi xmax şi poziţia lor în şir

EX22.PAS

```
program minmax;
 const n=30;
 var x:array[1..n] of integer;
     xmin, xmax, imin, imax, i:integer;
begin
 writeln(' introdu ', n, ' valori intregi ');
 for i:=1 to n do read(x[i]);
 xmax:=x[1]; xmin:=x[1]; imin:=1; imax:=1;
 for i:=2 to n do
  begin
    if xmax < x[i] then
     begin
      xmax:=x[i]; imax:=i;
     end;
```

```
    if xmin>x[i] then
     begin
      xmin:=x[i]; imin:=i;
     end;
  end;
 writeln(' i   x[i]   ');
 for i:=1 to n do  writeln(i:3,x[i]:10);
 writeln(' xmin= ', xmin:10, ' in pozitia ', imin:3);
 writeln(' xmax= ', xmax:10, ' in pozitia ', imax:3);
 readln;
end.
```

2. Fiind dat un şir de valori întregi x[i], i=1,n, $n <= 100$, să se ordoneze crescător şirul.
Ca algoritm se foloseşte compararea elementelor x[i] cu x[i+1] (cu i mergând de la 1 la n-1) şi trecerea pe locul cu indice superior a valorii mai mari. La sfârşitul unei parcurgeri a şirului în x[n] se va afla valoarea cea mai mare din şir. Reluăm compararea elementelor x[i] cu x[i+1] cu i mergând de la 1 la n-2. În x[n-1] se va afla valoarea cea mai mare dintre valorile rămase ($x[n-1] < x[n]$); continuăm până la compararea lui x[1] cu x[2] cu aşezarea în x[2] a celei mari mari valori dintre x[1] şi x[2]. Acum şirul va fi ordonat.

EX23.PAS

```
program ordonare;
var x:array[1..100] of integer;
    i, j, n:byte;  a:integer;
begin
write(' introdu numarul de elemente:'); readln(n);
for i:=1 to n do
  begin
   write(' introdu elementul intreg x[',i:3,' ]=');
   readln(x[i]);
  end;
for j:=n-1 downto 1 do  {asigura scaderea dimensiunii sirului}
  for i:=1 to j do
     if x[i]> x[i+1] then
       begin
        a:=x[i];  x[i]:=x[i+1]; x[i+1]:=a;
```

```
       end;
 writeln(' sirul ordonat este:');
 for i:=1 to n do writeln(x[i]:5);
 readln;
 end.
```

3. Fiind dată o matrice, să se scrie programul ce calculează puterea **p** a matricii (**p** este un număr natural).

EX24.PAS

```
PROGRAM matrice; { ridicarea unei matrice la o putere naturala}
  CONST n=3;       {dimensiunea fixata pentru matrice}
  TYPE matricep=ARRAY[1..n, 1..n] of real;
  LABEL 1;
  VAR a, b, c : matricep;
      s : real;
      i, j, k : 1..n;
      L, p, o :BYTE;
BEGIN
 {citire/afisare matrice}
 for i := 1 to n do
    begin
     writeln(' introdu linia',i:2);
     for j :=1 to n do read(a[i, j]);
    end;
 writeln(' matricea initiala este:');
 for i :=1 to n do
  begin
    for j := 1 to n do write(a[i,j]:10);
    writeln;
  end;
1:  write(' introdu puterea '); readln(p);
 {initializare b cu matricea unitate}
 for i :=1 to n do
  for j := 1 to n do
    if i<>j then b[i, j] :=0 else b[i, j] :=1;
 {inmultire repetata}
```

```
 for L := 1 to p do
  begin
    for i :=1 to n do
     for j := 1 to n do
      begin
       s := 0;
        for k :=1 to n do s := s + a[i,k] * b[k, j];
        c[i, j] := s;
      end;
      b := c;
      end;    { end inmultire repetata}
  {afisare rezultat}
writeln(' matricea la puterea ',p,' este:');
  for i := 1 to n do
    begin
     for j :=1 to n do write(b[i, j]:10);
     writeln; {trecerea la randul urmator}
    end;
 writeln(' doriti ridicarea matricii la alta putere?);
 writeln(' introdu 1 pentru da '); readln(o); if o=1 then goto 1;
END.
```

4. Fiind dată o sumă în lei să se afişeze numărul minim de bancnote şi monezi pentru achitarea ei. Bancnotele sunt de 100, 50, 10, 1 lei iar monezile sunt de 0.50 si 0.10 lei. Pentru a pastra valori intregi cerute de instructiunea FOR din program introducem valorile inmultite cu 1000.

EX25.PAS

```
PROGRAM retributie;
  {exprima o suma cu numar minim de bancnote si monezi}
TYPE bancmon=(osutamii, cincizecimii, zecimii, omie, cincisute, osuta);
     bani=array[bancmon] of longint;
CONST
 valori:bani=(100000, 50000, 10000, 1000,500,100);
VAR retrib, totretrib : bani;
    v : bancmon;
    numesal:string;
    suma, total : longint;
```

```
    nsal : 1..1000; { numar salariati}
       i : 1..1000;
    numesal:string;
BEGIN
for v:=osutamii to osuta do totretrib[v]:=0; {initializare}
 total := 0; writeln;
  write(' introdu numar de salariati, nsal='); readln(nsal);
  for i := 1 to nsal do
    begin
    write(' introdu nume salariat: '); readln(nsal);
    write(' introdu suma de plata pentru salariat:');
    readln(suma); total := total + suma;
    for v := cincisutemii to osuta do
        begin
        retrib[v] := suma DIV valori[v];
        suma := suma MOD valori[v];
        write(retrib[v]:10);
        totretrib[v] := totretrib[v] + retrib[v];
        end; writeln;
    end;
    writeln;
    writeln(' totaluri '); write(total:10);
    for v := cincisutemii to osuta do write(totretrib[v]:10);
 END.
```

5. Programul calculează numărul studenţilor ce au promovat fiecare examen dintr-o sesiune si pentru fiecare student câte examene a promovat.

EX26.PAS

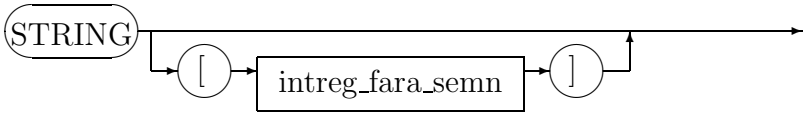
```
PROGRAM student_examen_promovat;
  CONST n=5   { numarul studentilor considerati aici}
  TYPE
examen=(analiza, mecanica, molecula, programare);
 exprom= set of examen;
VAR stud : ARRAY[1..n] of exprom;
    nexprom : ARRAY[1..n] of BYTE;
    i, anal, mec, mol, prog : BYTE;
```

```
    j : examen;
BEGIN
  stud[1] := [analiza, programare];
  stud[2] := [mecanica, molecula, programare];
  stud[3] := [];
  stud[4] := [analiza, mecanica, molecula, programare];
  stud[5] := [molecula, programare];
  {initializare contori la zero}
  anal := 0; mec := 0; mol := 0; prog :=0;
  { calculul numarului de studenti ce au promovat fiecare examen}
  for i := 1 to n do
     begin
     if analiza in stud[i] then anal := anal + 1;
     if mecanica in stud[i] then mec := mec + 1;
     if molecula in stud[i] then mol := mol + 1;
     if programare in stud[i] then prog := prog + 1;
     end;
  writeln(' din ', n, ' studenti ');
  writeln(' au promovat analiza : 'anal:2);
  writeln(' au promovat fizica moleculei: ', mol:2);
  writeln(' au promovat mecanica: ',mec:2);
  writeln(' au promovat Limbaje de programare: ',prog:2);
  writeln(' studenti ');
   for i := 1 to n do nexprom[i] := 0;
   for i := 1 to n do
     begin
     for j := analiza to programare do
       if j in stud[i] then nexprom[i] := nexprom[i] + 1;
       writeln('studentul ', i, ' a promovat ', nexprom[i],
       ' examene ');
     end;
   readln;
 END.
```

5.3 Tipul string (şir de caractere)

Tipul string este specific limbajului Pascal. Valoarea unei variabile de acest tip este formată dintr-un număr de caractere.

Tipurile string sunt tipuri structurate într-un mod asemănător cu tipul ARRAY cu diferenţa majoră că numărul de caractere într-un şir poate varia dinamic între zero şi limita superioară specificată (între 1 şi 255). Diagrama de sintaxă ce defineşte acest tip este următoarea:



întreg_fără_semn este o constantă întreagă în domeniul 1..255 şi dă lungimea maximă a şirului de caractere. Dacă nu e specificat se ia 255.
Exemple:

```
TYPE nume = STRING[14];
VAR s1, s2 : nume;
```

sau direct
VAR s1, s2 : STRING[14] {tip anonim }

Variabilele de tip STRING ocupă în octeţi lungimea maximă plus un octet (primul) ce conţine lungimea curentă a variabilei. Caracterele individuale într-un şir sunt indiciate de la 1 la lungimea şirului.

Expresiile STRING constau din constante şiruri, variabile tip STRING, nume de funcţii şi operatori

Operaţia de concatenare se realizează cu operatorul + sau cu funcţia **concat**. Dacă lungimea şirului rezultat depăşeşte 255 apare o eroare la execuţie.

Aplicarea operatorilor de relaţie (prioritate mai mică decât +) se face caracter cu caracter de la stânga la dreapta conform valorilor ASCII asociate. Dacă şirurile au lungimi diferite dar sunt egale până la ultimul caracter din şirul mai scurt, atunci şirul mai scurt este considerat mai mic. Şirurile sunt egale numai dacă lungimile şi conţinutul şirurilor sunt identice.

O valoare tip CHAR este compatibilă cu o valoare tip STRING considerându-se un string de lungime 1.

Asignarea şirurilor:
age := 'optsprezece';

Notă: dacă prin asignare se depăşeşte lungimea maximă a variabilei de tip STRING, caracterele în exces sunt trunchiate,

Proceduri pentru şiruri:

DELETE
sintaxa: **delete (St, Pos, Num)**
scop: şterge un substring de Num caractere începând cu poziţia Pos din şirul St
St - variabilă tip STRING
Pos, Num - expresii de tip întreg
- dacă Pos > Length(St) nu se şterge nici-un caracter
- dacă Pos + Num > Length(St) sunt şterse doar caracterele din şir
- dacă Pos > 255 apare o eroare în timpul execuţiei
exemplu: delete('abcde',3,2) dă şirul 'abe'

INSERT
sintaxa: **insert (Obj, Target, Pos)**
scop: inserează şirul Obj în şirul Target începând cu Poziţia Pos
Obj, Target - variabile tip STRING
Pos - expresie tip întreg
- dacă Pos > Length(Target) atunci Obj este concatenat la Target
- dacă rezultatul > lungimea maximă a lui Target, caracterele în exces vor fi trunchiate
exemplu: dacă St='abcdef' atunci insert('xx', St, 3) dă pentru St valoarea 'abxxcdef' dacă lungimea lui St permite.

STR
sintaxa: **Str(value, St)**
scop: converteşte valoarea numerică value într-un şir şi rezultatul este memorat în St.
value - variabilă de tip întreg sau real

st - variabilă tip STRING
exemplu: Str(5.25,st) dă st='5.25' **VAL**
sintaxa: **Val(St, Var, Code)**
scop: converteşte expresia de tip string St într-o valoare întreagă/reală în funcţie de tipul lui Var) şi o memorează în var.
St - tip STRING
Var - tip întreg sau real
Code - tip întreg; dacă nu sunt detectate erori, code are valoarea zero altfel este poziţionat pe primul caracter ce produce eroarea şi valoarea lui var este nedefinită.
exemplu: Val('12.25',var, code) dă var=12.25 şi code=0

Funţii pentru şiruri:

Copy
sintaxa: **copy (St, Pos, Num)**
scop: dă un subşir conţinând Num caractere din şirul St începând cu poziţia Pos.
St - tip STRING
Pos, Num - tip întreg
- dacă Pos > Length(St) funcţia dă un şir vid
- dacă Pos + Num > Length(St) funcţia dă doar caracterele din şir
- dacă Pos este în afara domeniului 1..255 funcţia dă eroare în timpul execuţiei
exemplu: dacă St='abcdefg' atunci
copy(St,4,2) dă subşirul 'de' iar
copy(St, 4, 10) dă subşirul 'defg'

Length
sintaxa: **length(St)**
scop: dă lungimea reală a lui St
St - tip STRING
rezultat - tip întreg
exemplu: length('abcd')=4

Pos
sintaxa: **Pos(obj, target)**
scop: scanează şirul target pentru a găsi şirul obj în target
obj, target - tip STRING

rezultatul este un întreg şi arată poziţia în target a primului caracter din obj; dacă obj nu e găsit, Pos dă valoarea zero
exemplu: dacă St este 'abcdefg' atunci Pos('de', St) dă valoarea 4.
EX27.PAS

```
PROGRAM sir;
 VAR s1 : STRING[10]; s2 : STRING[20]; l : INTEGER;
 BEGIN
 s1 := 'turbo';
 s2 := 's1 + ' pascal';
 l := length(s2);
 writeln(s2); writeln(' lungimea reala = ', l);
 END.
```

EX28.PAS

```
PROGRAM reversie_sir;
  VAR str : STRING; i : BYTE;
BEGIN
  write(' introdu un sir de caractere:'); readln(str);
  writeln(' sirul inversat este:');
  for i := length(str) downto 1 do write(str[i]);
  readln;
END.
```

EX1311.PAS

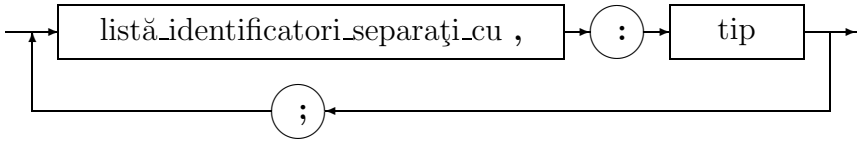
```
program test_string;
uses crt;
var  s:string[11];
     s1,s2,s5:string[10];
     s3,s4:string;
     v1,v2:string;
     v3,code:integer; v4:real;
     c1,c2:string;
 begin
 clrscr;
 s:='optsprezece';s1:=s;s2:=s1;s3:=s; s4:=s;s5:=s1;
 writeln(' var. string[11], s=',s:11);
```

```
writeln(' var. string[10], s1=',s1:10);
readln; delete(s1,2,7); writeln(' delete(s1,2,7)=',s1:10);
delete(s2,12,7);writeln('delete(s2,12,7)=',s2:10);
writeln;
insert('abc',s3,2);writeln(' insert(''abc'',s3,2)=',s3:10);
insert('abc',s5,2);writeln(' insert(''abc'',s5,2)=',s5:10);
insert('abc',s4,15);writeln(' insert(''abc'',s4,15)=',s4:10);
writeln;
v3:=14;str(v3,v1);
writeln('v3=',v3:10,'; str(v3,v1)=',v1:10);
v4:=2.15;str(v4,v2);
writeln('v4=',v4:10:3,'; str(v4,v2)=',v2:10);
writeln;
val(v1,v3,code);writeln('code=',code:3);
writeln(' val(v1,v3,code)=',v3:10);
val(v2,v4,code);writeln('code=',code:3);
writeln(' val(v2,v4,code)=',v4:10:3);
writeln;
v1:='12bc';v2:='2.1a';
val(v1,v3,code);writeln('code=',code:3);
writeln(' val(v1,v3,code)=',v3:10);
val(v2,v4,code);writeln('code=',code:3);
writeln(' val(v2,v4,code)=',v4:10:3);
writeln;readln;
c1:=copy(s,2,4); writeln('copy(s,2,4)=',c1:15);
c1:=copy(s,length(s)+2,4);
writeln('copy(s,length(s)+2,4)=',c1:15);
c1:=copy(s,2,18); writeln('copy(s,2,18)=',c1:15);
writeln;
writeln(' pos(''opt'',s)=',POS('opt',s):15);
writeln(' pos(''opr'',s)=',POS('opr',s):15); writeln;
writeln('concat(''eu am '',s,''ani'')=',concat('eu am ',s,' ani'));
writeln(' ''eu am + s + ani='' ','eu am ' + s + ' ani');
readln;
end.
```

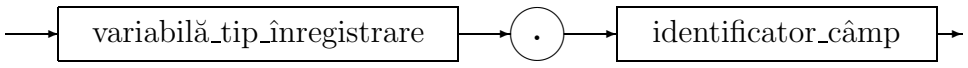
5.4 Tipul record(înregistrare)

Tipul înregistrare este un tip compus format dintr-un număr de componente, numite câmpuri. Spre deosebire de variabilele de tip ARRAY, câmpurile, elemente ale variabilelor de tip RECORD, pot fi de tipuri diferite. Fiecare câmp are un nume, identificatorul de câmp. Numărul componentelor poate fi fix sau variabil. În primul caz se spune că avem o structură fixă, în cel de al doilea caz avem o structură cu variante.
Diagrama de sintaxă a definirii tipultui RECORD este următoarea:

unde lista_câmpurilor în structura fixă are diagrama de sintaxă:



Referirea la o componentă din înregistrarea cu structură fixă se face conform următoarei diagrame de sintaxă:



Exemple:

```
TYPE data=RECORD
          an : 1900..2100;   {tip interval}
          luna : (ian,feb, mar, apr, mai, iun, iul, aug, sep, oct,
          nov,dec);           {tip enumerare}
          zi : 1..31;         {tip interval}
  end;
VAR astazi : data;
```

Variabila astazi este o variabilă de tipul înregistrare, definit ca data, componentele ei pot fi selectate astfel: astazi.an, astazi.luna, astazi.zi

EX29.PAS

```
PROGRAM exrecord;
 TYPE data=RECORD
           an : 1900..2100;
           luna : (ian, feb, mar, apr, mai, iun, iul, aug, sep,
           oct, nov, dec);
           zi : 1..31;
   end;
 VAR astazi : data;
     i, j : BYTE;
 BEGIN
 astazi.an := 2003;
 astazi.luna := nov;
 astazi.zi :=3;
 for i := 0 to 11 do
 if ORD(astazi.luna)=i then j := i + 1;
 writeln(astazi.zi:2,'/',j:2,'/',astazi.an:4);
 END.
```

Notă: dacă două variabile de tip înregistrare sunt de acelaşi tip (ca în exemplul VAR data1, data2 : data;) atunci copierea unei înregistrări în cealaltă se poate face printr-o singură instrucţiune de atribuire (data1 := data2).

Alte exemple, folosite în grafică, sunt:

```
TYPE viewporttype=RECORD
                  x1, y1, x2, y2 : integer;
                  clip : boolean;
                  END;

TYPE pointtype=RECORD
               x, y : integer;
               END;
```

Componentele unei înregistrări pot fi la rândul lor înregistrări (tipuri îmbricate) ca în exemplul:

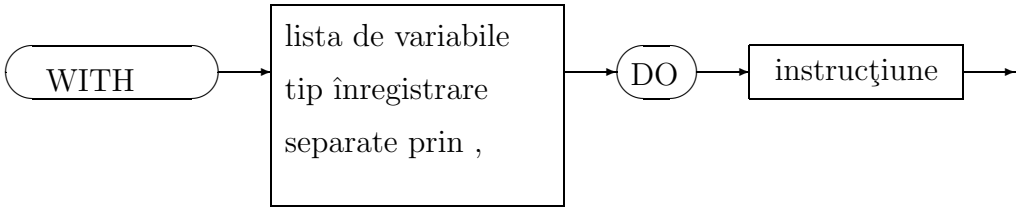
```
TYPE data=RECORD
          an : 1900..2100;
          luna:(ian, feb, mar, apr, mai, iun, iul, aug, sep, oct,
```

```
          nov, dec);
          zi:1..31;
          END;
     carte=RECORD
           titlu:ARRAY[1..20] of char;
           autor:ARRAY[1..15] of char;
           dateautor:data;
           END;
```

Declaraţia de variabile **VAR c:carte;** permite referirea la câmpurile variabilei de tip înregistrare, **c**, astfel:
c.autor[3] reprezintă a 3-a literă a numelui autorului cărţii **c**
c.dateautor.zi reprezintă ziua naşterii autorului cărţii **c**.

5.4.1 Instrucţiunea WITH

Instrucţiune WITH permite o referire prescurtată la câmpurile unei variabile de tip înregistrare. Diagrama de sintaxă a instrucţiunii este următoarea:



În cadrul unei instrucţiuni WITH, la întâlnirea unui nume de variabilă, prima dată se testează dacă variabila poate fi interpretată ca un nume de câmp al unei înregistrări. Dacă da, variabila va fi interpretată aşa chiar dacă o variabilă cu acelaşi nume este de asemeni accesibilă. Iată un exemplu:

```
TYPE punct=RECORD
           x, y : integer;
           END;
VAR x: punct; y : integer;
...
WITH x DO
     begin
```

```
     x := 10; y :=25;
     end;
...
```

x între WITH şi DO se referă la variabila de tip **punct** iar în instrucţiunea compusă x şi y reprezintă câmpurile x.x şi x.y
Dacă selectarea unei variabile de tip înregistrare necesită efectuarea indicierii unei variabile de tip tablou atunci operaţia este efectuată înainte de executarea instrucţiunii situate după cuvântul cheie DO. Iată un exemplu:

```
TYPE punct=RECORD
           x,y : Integer;
           END;
     tabel=ARRAY[1..10] of punct;
VAR t : tabel;

... WITH t[5] DO
        begin
         x := 1;   y := 2;
        end;
...
```

este echivalentă cu t[5].x := 1; t[5].y :=2;

5.4.2 Constante de tip RECORD

Declararea unei constante de tip înregistrare specifică identificatorul şi valoarea fiecărui câmp, incluse în paranteze şi separate prin ;
Iată un exemplu:

```
TYPE punct=RECORD
           x, y : real;
           END;
      data=RECORD
           zi:1..31;
           luna:1..12;
```

```
            an:1900..2100;
            END;
     vector=array[0..1] of punct;
CONST
    origine : punct = (x:0.0; y:0.0);
    linie:vector = ((x:-3.1; y:1.5),(x:5.8; y:3.0));
    azi:data=(zi:3; luna:11; an:2003);
```

Câmpurile trebuie să fie specificate în aceeaşi ordine în care ele apar în definiţia tipului înregistrare.

5.5 Tipul file

În paragraful 4.2 am văzut că există două fişiere standard predeclarate: fişierul standard de intrare (tastatura) şi fişierul standard de ieşire (ecranul monitorului). Dacă vrem să citim sau să scriem date folosind un fişier de pe discul magnetic atunci trebuie să definim o variabilă de tip FILE pe care să o punem în corespondenţă cu fişierul dorit.

5.5.1 Conceptul de fişier

Fişierele constau dintr-o secvenţă de componente de acelaşi tip. Numărul de componente într-un fişier (dimensiunea fişierului) nu este determinat prin definirea fişierului. Tipul unui fişier este definit de cuvântul rezervat **FILE OF** şi urmat de tipul componentelor fişierului. Componentele unui fişier pot fi de orice tip exceptând tipul file.

Diagrama de sintaxă este următoarea:

Exemple:

```
TYPE f=file of byte;
     g=file of real;

VAR a,b:f;
    c:g;
```

Corespondenţa între variabila tip fişier şi numele fişierului se face apelând procedura **ASSIGN(FilVar, Str)** unde FilVar este numele variabilei tip fişier iar Str este numele fişierului.

Exemple:

```
ASSIGN(f,'date.in');
ASSIGN(g,'xdat.out');
```

După asignarea numelui fişierului variabilei tip fişier în instrucţiunile READ sau WRITE va interveni doar numele variabilei tip fişier.

5.5.2 Scrierea fişierelor

Un fişier este pregătit pentru scriere prin apelarea procedurii standard **REWRITE(g)**. Scrierea valorilor în fişier se face cu procedura **WRITE(g,** $x_1, ..., x_n$**)** unde în prima poziţie apare numele variabilei tip fişier iar valorile $x_1, ..., x_n$ trebuie să fie de acelaşi tip cu cel al componentelor fişierului g.

EX30.PAS

```
program testfile1;
 TYPE f=FILE OF real;
 VAR a:f;
     x:array[1..10,1..5] of real;
     i,j:byte;
BEGIN
 assign(a,'testfile.dat'); rewrite(a);
 for i:= 1 to 10 do
    for j:= 1 to 5 do
     begin
        x[i,j]:=1.*i*j; { 1. transforma i*j intr-o valoare reala}
        write(a,x[i,j]);
     end;
close(a);
END.
```

5.5.3 Citirea fişierelor

Pregătirea unui fişier pentru citire se face apelând procedura **RESET(f)**. Citirea datelor dintr-un fişier se face cu procedura standard **READ(f,a,b,c)** unde variabilele a,b,c trebuie să fie de acelaşi tip cu cel al componentelor fişierului f.

EX31.PAS

```
program testfile2;
 TYPE f=FILE OF real;
 VAR a:f;
     x:array[1..10,1..5] of real;
     i,j:byte;
 BEGIN
 assign(a,'testfile.dat'); reset(a);
 {citeste datele din fisierul 'testfile.dat'}
  for i:=1 to 10 do
     for j:=1 to 5 do read(a, x[i, j]);
 { scrie datele citite pe ecran, o linie a matricii x pe un rand}
  for i := 1 to 10 do
   begin
    for j := 1 to 5 do write(x[i,j]:6:1);
    writeln;
   end;
 close(a);   readln;
 END.
```

5.5.4 Câteva proceduri pentru manipularea fişierelor

SEEK

Sintaxa: **Seek(FilVar, n)**

Seek mută pointerul de fişier la a n-a componentă a fişierului asignat lui FilVar. n este o expresie de tip întreg. Prima componentă a fişierului are n=0. Extinderea fişierului asignat lui FilVar se poate face cu procedura SEEK în care n are valoarea numărul de componente din fişier plus unu sau cu Seek(FilVar, FileSize(FilVar))

CLOSE
Sintaxa: **Close(FilVar)**
Fişierul asignat lui FilVar este închis. Close este necesar şi asupra fişierelor deschise pentru citire.

5.5.5 Funcţii standard asupra fişierelor

Eof
Sintaxa: **Eof(FilVar)**
Funcţia întoarce valoarea TRUE dacă pointerul de fişier este poziţionat la sfârşitul fişierului, peste ultima componentă a fişierului. Altfel întoarce FALSE.

FilePos
Sintaxa:**FilePos(FilVar)**
Funcţia întoarce poziţia curentă a pointerului de fişier. Prima componentă are numărul zero.

FileSize
Sintaxa:**FileSize(FilVar)**
Funcţia întoarce numărul de înregistrări din fişier.

EX32.PAS

```
program testfile3;
 var f : file of byte;
     nume : string;   i, j : byte;
BEGIN
write('introdu nume fisier: '); readln(nume); assign(f, nume);
rewrite(f);
for i := 1 to 15 do
  begin
  j := i * i; write(f, i, j);
  end;
close(f); reset(f);
while not eof(f) do
  begin
  read(f, i, j); writeln(i : 3, j :4);
```

```
  end;
write('filesize(f)=',filesize(f)); writeln; close(f);
reset(f); seek(f, 10);
{pozitionare pointer la sfarsitul inregistrarii 10}
writeln('filepos(f)=', filepos(f));
{scrie pozitia pointerului in fisier}
read(f, i, j); writeln(i, ':', j);
{ scrie inregistrarile 11 si 12}
close(f)
END.
```

EX210.PAS

```
program test_truncate;
  var   f:file of INTEGER;
        i,j:integer;
BEGIN
 assign(f,'test.int'); rewrite(f);
 for i:=1 to 6 do write(f,i);
 writeln(' fisier inainte de trunchiere'); reset(f);
 while not eof(f) do
       begin
      read(f,i); writeln(i);
       end;
 reset(f); for i:=1 to 3 do  read(f,i);
{citeste primele 3 inregistrari}
 truncate(f);
{ sterge restul fisierului din pozitia curenta
a pointerului de fisier}
 writeln;  writeln(' fisierul dupa trunchiere');
 reset(f);
 while not eof(f) do
     begin
    read(f,i); writeln(i);
     end;
 close(f);
 erase(f);
 {sterge fisierul test.int din directorul curent}
 end.
```

5.5.6 Fişiere de tip Text

Fişierele descrise mai sus pot fi scrise sau citite doar într-un program în Turbo Pascal nu cu un editor DOS. Fişierele de tip TEXT pot fi însă scrise sau citite şi în DOS.

Fişierele de tip TEXT sunt structurate în linii, fiecare linie se termină cu End-Of-Line (EOL) iar fişierul se termină cu Enf-Of-File (EOF). Fişierele de tip TEXT sunt fişiere secvenţiale. Operaţiile cu caractere se fac cu procedurile READ şi WRITE. Liniile din fişier sunt procesate cu procedurile READLN şi WRITELN. Sintaxa de declarare a unei variabile de tip fişier text este:
VAR f:TEXT;
EX1306.PAS

```
program test_text;
var f:text;
    s:array[1..20] of string;
    i:byte;
    nume:string[15];
begin
 writeln('introdu nume fisier output'); readln(nume);
 assign(f,nume);rewrite(f);
 writeln(f,'textul de inceput');
 close(f); append(f);writeln(f,'text adaugat');writeln(f,'text 2');
 close(f); reset(f); i:=0;
 while not eof(f) do
 begin
 i:=i+1;
 readln(f,s[i]); writeln(s[i]); {scrie pe ecran randurile din fisier}
 end;
 close(f);
 writeln('i=',i);  {scrie pe ecran numarul de randuri din fisier}
readln; end.
```

EX33.PAS

```
program testfile4;
const n=10;
 var f : text;  nume:string;
```

```
     x, y, z, m : array [1..n] of real;
     xcm, ycm, zcm, mtot, sxm, sym, szm, sm : real;
BEGIN
write('introdu nume fisier cu x, y, z, m ale celor n
       puncte materiale')
readln(nume); assign(f,nume); reset(f);
 for i:= 1 to n do readln(f, x[i], y[i], z[i], m[i]);
close(f);   {scrie pe ecran datele citite}
writeln('  i     x        y          z          m    ');
for i:= 1 to n do
writeln(i:5,x[i]:10:2,y[i]:10:2,z[i]:10:2,m[i]:10:2);
sxm := 0; sym :=0; szm := 0; sm := 0; {initializare sume}
 for i := 1 to n do
   begin
   sxm := sxm + x[i] * m[i];    sym := sym + y[i] * m[i];
   szm := szm + z[i] * m[i];    sm := sm + m[i];
   end;
xcm:=sxm/sm; ycm:=sym/sm; zcm:=szm/sm; mtot:=sm;
writeln('coordonatele centrului de masa sunt:);
writeln('xcm=', xcm, 'ycm=', ycm, 'zcm=',zcm);
writeln('masa totala=', mtot); readln;
append(f); {deschide fisierul si adauga la sfarsitul sau}
writeln(f,'xcm=', xcm, ' ycm=', ycm, ' zcm=', zcm);
writeln(f, ' mtot=', mtot); close(f);
END.
```

Următorul program citeste din fisierul text lista.in numarul de elevi dintr-o clasa, numarul de teste si elevii cu notele la teste. Ordoneaza alfabetic elevii pastrand informatia despre notele lor si face media Scrie pe ecran informatia citita din lista.in si scrie rezultatele in lista.out

```
program ordstring;
uses crt;
label unu;
var nume:array[1..30] of string[20];
    prenume:array[1..30] of string[20];
    note:array[1..30,1..10] of byte;
    aux:string[20];
    ivechi:array[1..30] of byte;
```

```
    media:array[1..30] of real;
    i, j, k, m, n, iaux:byte;
    s:real;  f,g:text;
begin
clrscr;
{introducerea datelor}
assign(f,'lista.in'); reset(f);
readln(f,n); readln(f,m);
writeln(n:4,m:4);
for i:=1 to n do
  begin
   readln(f,nume[i],prenume[i]);
   for j:=1 to m do read(f,note[i,j]);
   readln(f);
  end;
 {scrie datele introduse}
  writeln('datele initiale sunt:');
  for i:=1 to n do
      begin
     writeln(nume[i], prenume[i]);
     for j:=1 to m do write(note[i,j]:5);
     writeln;
      end;
{ordonare alfabetica crescator}
  for i:=1 to n do ivechi[i]:=i;
 unu: k:=0;
  for i:=1 to n-1 do
    if nume[i]>nume[i+1] then
         begin
     aux:=nume[i]; iaux:=ivechi[i];
     nume[i]:=nume[i+1]; ivechi[i]:=ivechi[i+1];
     nume[i+1]:=aux; ivechi[i+1]:=iaux;
     k:=k+1;
          end
    else if nume[i]=nume[i+1] then
      if prenume[i]>prenume[i+1] then
          begin
        aux:=prenume[i]; iaux:=ivechi[i];
```

```
      prenume[i]:=prenume[i+1]; ivechi[i]:=ivechi[i+1];
    prenume[i+1]:=aux; ivechi[i+1]:=iaux;
      k:=k+1;
          end;
    if k<>0 then goto unu;
  for i:=1 to n do
    begin
   s:=0; for j:=1 to m do s:=s+note[ivechi[i],j];
   media[i]:=s/m;
    end;
    assign(g,'lista.out'); rewrite(g);
  writeln(g,'               elev              note         media');
  for i:=1 to n do
       begin
      write(g,nume[i]+prenume[i]:40);
     for j:=1 to m do write(g,note[ivechi[i],j]:3);
     writeln(g,media[i]:6:2);
       end;
close(g);    {altfel nu scrie ultimul rand in fisier!}
 readln;
 end.
```

Fisierul lista.in poate arata asa:

```
10
 5
 ionescu alexandra
 10 10 10 10 10
 adam ion
 10 9 10 8 9
 adam alexandru
 5 6 8 4 5
 andrei violeta
 8 8 9 7 8
 ionescu ioana
 10 9 6 9 10
 popescu mihai
 6 7 8 5 1
 constantinescu ana
```

```
10 8 9 10 8
constantinescu ioan
10 10 10 10 10
matei alexandru
8 8 7 8 9
matei marin
9 6 5 1 8
```

Atunci lista.out va arata asa:

```
       elev              note          media
       adam alexandru  5  6  8  4  5  5.60
       adam ion       10  9 10  8  9  9.20
       andrei violeta  8  8  9  7  8  8.00
  constantinescu ana 10  8  9 10  8  9.00
 constantinescu ioan 10 10 10 10 10 10.00
   ionescu alexandra 10 10 10 10 10 10.00
       ionescu ioana 10  9  6  9 10  8.80
    matei alexandrua  8  8  7  8  9  8.00
         matei marin  9  6  5  1  8  5.80
    popescu mihai     6  7  8  5  1  5.40
```

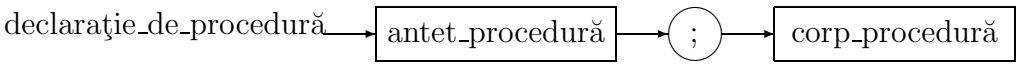
Capitolul 6

PROCEDURI ŞI FUNCŢII

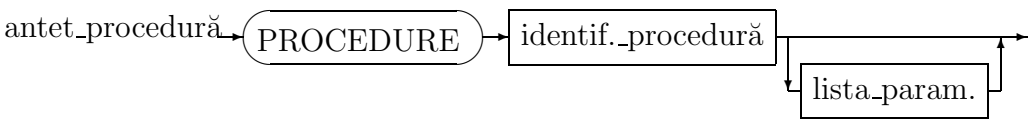
Procedurile şi funcţiile permit structurarea programelor complexe, fiecare procedură sau funcţie realizând complet o sarcină concretă în cadrul programului în care apare. Fiecare declaraţie de procedură sau funcţie, ce apare în partea declarativă a unui program, are un antet urmat de un bloc (ca în EX3.PAS şi EX4.PAS). O procedură este activată printr-o instrucţiune de procedură (cmmdc din EX4.PAS); o funcţie este activată ca orice funcţie standard (f(a) sau f(b) în EX3.PAS).

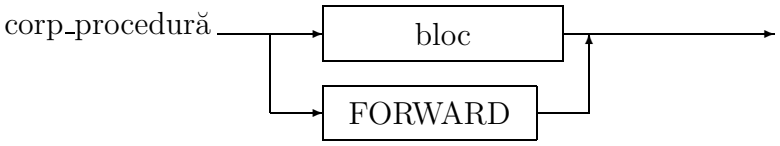
6.1 Proceduri

O declaraţie de procedură asociază un identificator cu un bloc de procedură. Diagrama de sintaxă pentru cazurile simple, pe care le vom studia, este următoarea:



în care:





- În antetul procedurii scriem identificatorul procedurii şi, dacă există, parametri formali. Procedura este apelată în cadrul programului prin numele ei urmat, dacă există, de parametri actuali.

- În locul blocului poate să apară declaraţia FORWARD. Se ştie că una din regulile de bază ale limbajului este că locul de definiţie al unui nume trebuie să preceadă textual utilizările numelui respectiv. Respectarea acestei reguli întâmpină greutăţi în unele cazuri. De exemplu un program declară două proceduri P şi Q însă P apelează procedura Q şi Q apelează procedura P. Este clar că în acest caz oricare ar fi forma programului utilizarea numelui unei proceduri precede locul ei de definiţie. Pentru a rezolva această problemă s-a introdus directiva FORWARD prin care se pot separa fizic cele două componente de bază ale declaraţiei unei proceduri sau funcţii: antetul şi corpul procedurii sau funcţiei. Vom avea:

```
PROCEDURE Q(x, y, z : INTEGER); FORWARD;
PROCEDURE P(u, v, w : INTEGER);
BEGIN
...
Q(1, 2, 3)
...
END;
PROCEDURE Q; { nu se repetă declararea parametrilor lui Q}
...
BEGIN
...
P(5, 7, 9);
...
END;
...
```

6.1.1 Variabile locale şi globale

Să amintim programul fracţie din EX4.PAS ce utiliza procedura de calcul al celui mai mare divizor comun (**cmmdc**) pentru a simplifica o fracţie raţională exprimată ca un raport de două numere întregi, a/b.

Deoarece algoritmul lui Euclid, folosit pentru găsirea celui mai mare divizor comun al numerelor a şi b, modifică numerele a şi b, ele vor fi copiate în x şi y. Etapele programului vor fi:

- citirea şi afişarea numărătorului **a** şi numitorului **b**
- copierea lui **a** în **x** şi a lui **b** în **y**
- algoritmul lui Euclid cere $x > y$
- calculul **cmmdc** între **x** şi **y**
- dacă **cmmdc** > 1 atunci **a** şi **b** se împart la el
- afişarea numărătorului şi numitorului simplificate

```
EX4P.PAS
PROGRAM fractie;
 {simplificare fractie a/b prin impartire cu cmmdc}
 VAR a, b, x, y, z : INTEGER;
 PROCEDURE cmmdc;    {calcul cmmdc cu algoritmul lui Euclid}
   begin
    if x < y then         { asigurare x > y }
      begin
      z:=x; x:=y; y:=z;
      end;
 while y<>0 do        {impartiri repetate}
       begin
       z:=x mod y;  x:=y; y:=z;
       end;
  end;    {end procedura cmmdc}
 BEGIN    {programul principal}
 writeln('introdu numaratorul si numitorul ca numere intregi
 pozitive'); read(a,b);
 writeln('fractie nesimplificata: ',a,'/',b);
```

```
x:=a; y:=b; cmmdc;   {apelare procedura cmmdc}
if x>1 then   {simplificare fractie}
  begin
  a:=a div x; b:=b div x;
  writeln('fractie simplificata: ',a,'/',b);
  end else writeln('fractia nu se poate simplifica');
END.
```

În exemplul dat mai sus declaraţia de procedură nu conţine partea de declaraţie de aceea toate variabilele, declarate în programul principal, sunt variabile globale.

Există posibilitatea de a scrie o parte de declaraţie în interiorul procedurii la fel ca în orice program în Turbo Pascal. Identificatorii introduşi în partea de declaraţie a procedurii sunt locali, ei pot fi referiţi şi cunoscuţi numai în blocul în care au fost declaraţi, acesta reprezentând domeniul acestor identificatori. Iată cum arată programul de mai sus cu variabile locale.

EX4S.PAS

```
PROGRAM fractie2;
 VAR a, b, c : integer;       {variabile globale}
 PROCEDURE cmmdc;
   VAR x, y, z : integer;     {variabile locale}
   begin
   x:=a; y:=b; if x < y then
         begin
      z:=x; x:=y; y:=z;
         end;
   while y <> 0 do
     begin
     z:= x mod y; x:=y; y:=z;
     end;
   c:=x;  { cmmdc e transmis intr-o variabila globala}
   end; {end cmmdc}

 BEGIN    {programul principal}
 writeln('introdu numaratorul si numitorul ca numere intregi
 pozitive'); read(a,b);
 writeln('fractie nesimplificata: ',a,'/',b);
```

```
cmmdc;   {apelare procedura cmmdc}
if c>1 then   {simplificare fractie}
  begin
  a:=a div c; b:=b div c;
  writeln('fractie simplificata: ',a,'/',b);
  end
else writeln('fractia nu se poate simplifica');
END.
```

6.1.2 Domeniul de valabilitate al obiectelor

Fiecare corp de procedură poate conţine în partea sa de declaraţie o declaraţie de procedură sau funcţie (numită procedură sau funcţie inclusă sau locală). Prin obiecte înţelegem constante, tipuri, variabile, proceduri, funcţii identificate prin identificatorul asociat.

Există câteva reguli ce determină domeniul de valabilitate şi durata de viaţă ale unui identificator. Ele sunt următoarele:

1. domeniul unui identificator îl constituie blocul în care a fost declarat şi toate blocurile incluse în el

2. dacă un identificator **a**, declarat într-un bloc **x**, este redeclarat într-un bloc **y** atunci blocul **y** şi blocurile incluse lui se exclud din domeniul de valabilitate al identificatorului **a** declarat în **x**

   EX34.PAS

   ```
   program test_var;
    var x, y:real;
    procedure citit_scris;
      var x, u:real;
      begin
      write(' introdu x si u, numere reale: '); readln(x, u);
      writeln(' x=',x:5:1; 'u=',u:5:1);
      end;
    begin
    write(' introdu x si y, numere reale: '); readln(x, y);
     citit_scris;
    writeln('x=',x:5:1, 'y=',y:5:1);
    end.
   ```

Dacă am introdus x=1.5, y=2.5 şi apoi x=10.5 şi u=20.5 pe ecran va apare:

```
x= 10.5 u= 20.5
   valorile citite si scrise in procedura citit_scris
x=  1.5 y=  2.5
    valorile citite inainte de apelarea procedurii si
    scrise dupa apelarea ei
```

3. identificatorii de proceduri se supun aceloraşi reguli de domeniu ca şi ceilalţi identificatori, deci o procedură poate fi folosită doar în blocul în care ea a fost declarată şi în blocurile incluse în acesta

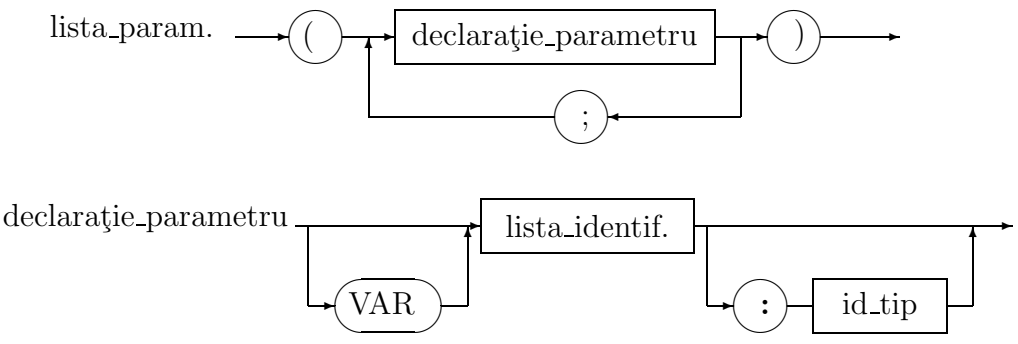
4. o procedură se poate referi la ea însăşi (apelare recursivă)

Aceste reguli determină şi durata de viaţă a identificatorilor; o variabilă declarată ca locală într-o procedură există numai în timpul execuţiei procedurii fiind creată la activarea procedurii prin alocarea de memorie şi distrusă la ieşirea din procedură prin eliberarea memoriei ocupate.

6.1.3 Parametri

Folosirea parametrilor formali permite apelarea aceloraşi proceduri în puncte diferite cu valori diferite ale variabilelor.

Un parametru formal reprezintă un obiect local al procedurii. Lista parametrilor formali are diagrama de sintaxă următoare:



Există trei feluri de parametri (valoare, variabile, variabile fără tip) caracterizaţi astfel:

1. un grup de parametri separaţi prin virgulă, fără a fi precedaţi de cuvântul cheie **VAR** dar urmaţi de identificatorul de tip este o listă de parametri valoare

2. un grup de parametri separaţi prin virgulă, precedaţi de cuvântul cheie **VAR** şi urmat de identificatorul de tip este o listă de parametri variabile

3. un grup de parametri separaţi prin virgulă, precedaţi de cuvântul cheie **VAR** dar neurmat de identificatorul de tip este o listă de parametri variabile fără tip

Acţiunea lor este următoarea:

1. Un parametru formal valoare acţionează ca o variabilă locală pentru procedură cu diferenţa că la activarea procedurii îşi ia valoarea iniţială din parametrul actual corespunzător. Din această cauză se mai numeşte şi **parametru de intrare**. Modificările făcute asupra parametrului formal valoare în procedură nu afectează valoarea parametrului actual corespunzător. Parametrul actual trebuie să fie de un tip compatibil atributiv cu tipul parametrului formal valoare.

2. Un parametru formal variabilă este folosit când valoarea trebuie transferată de la procedură la programul apelant. Parametrul actual corespunzător în instrucţiunea procedură (care activează procedura) trebuie să fie o referire de variabilă. Orice schimbări ale parametrului formal variabilă sunt reflectate în parametrul actual. Tipul parametrului actual trebuie să fie identic cu tipul parametrului formal variabilă (această restricţie se poate evita prin folosirea unor parametri formali fără tip).

3. Când un parametru formal este o variabilă fără tip, parametrul actual corespunzător poate fi orice referire de variabilă indiferent de tipul ei.

Să rescriem programul fracţie2 cu procedura cmmdc cu parametri valoare şi apoi cu parametri valoare şi variabile (EX35.PAS şi, respectiv, EX36.PAS).

```
EX4T.PAS
PROGRAM fractie3;
 VAR a, b, c : integer;
 PROCEDURE cmmdc(x, y:integer);{x si y sunt parametrii valoare}
  VAR z:integer;
  begin
   if x < y then
      begin
      z:=x; x:=y; y:=z;
      end;
   while y < > 0 do
      begin
      z:=x mod y; x:=y; y:=z;
      end;
   c:=x;   { cmmdc e transmis intr-o variabila globala c}
   end;   { end cmmdc}
 BEGIN   {program principal}
 writeln(' introdu a si b, numere intregi, pozitive');
 readln(a,b);
 writeln('fractie nesimplificata: ',a,'/',b);  cmmdc(a,b);
 if c > 1 then
   begin
   a:= a div c; b:=b div c;
   writeln('fractia simplificata: ',a,'/',b);
   end
 else writeln('cmmdc=',c);  readln;
 END.

EX4Q.PAS
PROGRAM fractie3prim;
 VAR x, y, c : integer;
 PROCEDURE cmmdc(x, y:integer);{x si y sunt parametrii valoare}
  VAR z:integer;
  begin
   if x < y then
      begin
      z:=x; x:=y; y:=z;
      end;
```

```
   while y < > 0 do
      begin
      z:=x mod y; x:=y; y:=z;
      end;
   c:=x;   { cmmdc e transmis intr-o variabila globala c}
   end;   { end cmmdc}
 BEGIN   {program principal}
 writeln(' introdu x si y, numere intregi, pozitive');
 readln(x,y);
 writeln('fractie nesimplificata: ',x,'/',y);  cmmdc(x,y);
{x si y nu sunt modificati in procedura}
 if c > 1 then
   begin
   x:= x div c; y:=y div c;
   writeln('fractia simplificata: ',x,'/',y);
   end
 else writeln('cmmdc=',c);  readln;
 END.
```

EX4C.PAS

```
PROGRAM fractie4;
  VAR a, b, c :integer;
  PROCEDURE cmmdc(x, y :integer; VAR w:integer);
  VAR z:integer;
  PROCEDURE swap;
    begin
    z:=x; x:=y; y:=z;
    end;  {end swap}
  begin  {begin cmmdc}
  if x < y then swap;
  while y < > 0 do
     begin
     z:=x mod y; x:=y; y:=z;
     end;
  w:=x
  end;  {end cmmdc}
BEGIN  {program principal}
```

```
 write(' introdu a, b numere intregi, pozitive');
 readln(a, b); writeln( 'fractia nesimplificata: ',a,'/',b);
 cmmdc(a, b, c);
 if c > 1 then
   begin
    a:= a div c; b:=b div c;
    writeln('fractie simplificata:', a, '/',b);
   end
 else writeln('cmmdc=',c);  readln;
END.
```

Dar adevărata folosire a unei proceduri cu parametri valoare (apelarea ei în diverse locuri din program cu valori de intrare diferite) este ilustrată de EX37.PAS. Programul converteşte o sumă (între 0 şi 999) din cifre în cuvinte.

EX35.PAS

```
PROGRAM lei_cuvinte;
 TYPE suma=0..999;
 VAR lei:suma;
 PROCEDURE convincuv(x:suma);
 {converteste un numar >0 si <999 in cuvinte}
 TYPE cifra=0..9;
 VAR sute, zeci, unitate:cifra; rang:(unu, zece, suta);
 PROCEDURE unitati(c:cifra);{scrie in cuvinte o singura cifra}
    begin
     case c of
     0: ;
     1: if rang=suta then write('o') else write('un');
     2: if rang=unu then write ('doi') else write('doua');
     3: write('trei');
     4: write('patru');
     5: write('cinci');
     6: write('sase');
     7: write('sapte');
     8: write('opt');
     9: write('noua');
     end;  {end case}
    end; {end unitati}
```

```
  begin {begin convincuv}
  {scrie numarul complet in cuvinte}
  {separararea cifrelor numarului}
  sute:= x div 100; zeci:= x mod 100 div 10; unitate:= x mod 10;
  {prelucrarea cifrei sutelor}
  if sute > 0 then
    begin
    rang:=suta; unitati(sute);
    if sute = 1 then write(' suta ') else write(' sute ');
    end;   {prelucrarea cifrei zecilor}
  if zeci > 0 then
    begin
    rang:=zece;
    if zeci < > 1 then
      begin
      unitati(zeci); write(' zeci ');
      end;
    end;   {prelucrarea cifrei unitatilor}
  if unitate > 0 then
    begin
    rang:=unu;
    if zeci > 1 then write(' si '); unitati(unitate);
    if zeci - 1 then write('sprezece ');
    end
  else
    if zeci=1 then write(' zece ');
 end {end convincuv}
 BEGIN {programul principal}
 write(' introdu suma in cifre < 999:'); readln(lei);
 convincuv(lei);  writeln(' lei'); readln;
 END.
```

Alt exemplu de proceduri cu parametri valoare sunt cele din programul ce realizează trecerea unui număr real din baza 10 intr-o altă bază dată de utilizator.

EX36.PAS

```
program baza;
```

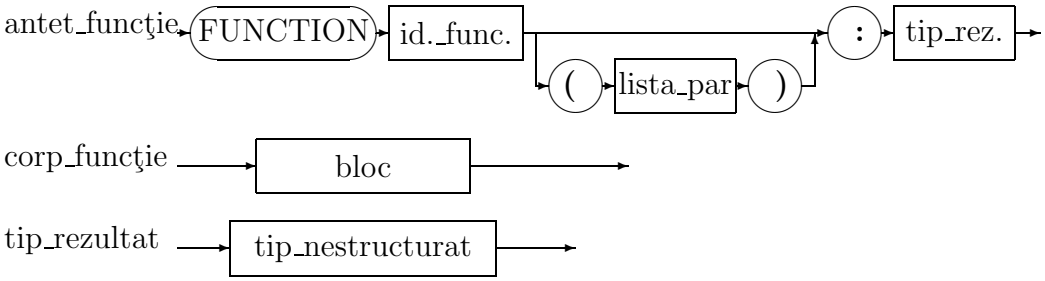
```
label 1, 2, 3;
const n=100;
type sir=array[1..n] of char;
var nr:real; ba:byte;  ta:char;
procedure intgr(nr:real; ba:integer);
 var de, re, im, i, m:integer;  x:sir; a:char;
 begin
 i := 1; m := 0; de := trunc(nr);
 repeat
   re := de mod ba;  im := de div ba;
   if re >= 10 then x[i] := chr(ord('A')+(re mod 10))
   else x[i] := chr(re+48);
   i := i+1; m :=m+1; de := im;
 until de=0;
 for i :=m downto 1 do write(x[i]); write('.');
 end;  {end procedure intgr}
 procedure frac(nr:real; ba:integer);
   var de, m, re, i:integer; x:sir; a:char; ra, im:real;
   begin
   i := 1; m := 0; de :=trunc(nr); ra := nr-de;
   repeat
     im :=ra*ba;  re := trunc(im);
     if re >= 10 then x[i] :=chr(ord('A')+(re mod 10))
     else x[i] := chr(re+48);
     ra := im-re; i := i+1; m := m+1;
   until m=10;
   for i:=1 to m do write(x[i]);
   end; { end procedure frac}
 BEGIN { programul principal}
 1: writeln('introduceti numarul in baza 10 '); read(nr);
    writeln('introduceti baza in care doriti sa treceti numarul');
    read(ba);
    writeln('Numarul scris in baza ', ba, 'este:');
    intgr(nr, ba); frac(nr, ba); writeln;
 2: writeln('doriti reluarea programului?(d/n)');
    readln(ta); if upcase(ta)='N' then goto 3;
                if upcase(ta)='D' then goto 1 else goto 2;
 3: readln;
```

```
END.
```

6.2 Funcţii

În afara funcţiilor standard programatorul îşi poate defini funcţii proprii. Diagrama de sintaxă pentru declararea unei funcţii este următoarea:

în care:

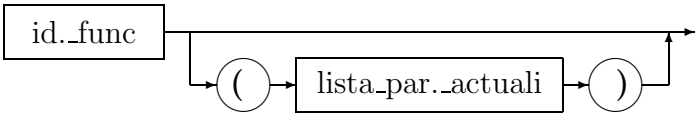


Notă:

- lista parametrilor formali conţine doar parametri valoare (ei având rolul datelor de intrare din declaraţia de procedură) şi parametri variabilă fără tip.
- tip_rezultat apare în plus faţă de declaraţia de procedură şi specifică tipul rezultatului transmis prin numele funcţiei.

Corpul funcţiei fiind un bloc poate include declaraţii de constante, tipuri, variabile, proceduri şi funcţii locale. În partea executabilă a funcţiei trebuie să apară cel puţin o atribuire prin care se transferă identificatorului funcţiei valoarea rezultatului calculat.

Apelarea (activarea) funcţiei se face conform următoarei diagrame de sintaxă:



Exemple: 1. Un program ce găseşte pentru ecuaţia:

$$x^4 - 9x^3 - 2x^2 + 120x - 130. = 0.$$

o rădăcină în intervalul [ai,bi] dat (programul rezec4 din EX3.PAS).

Acum să scriem programul ce găseşte toate cele patru rădăcini reale ale ecuaţiei fiind date intervalele în care ele se află.

EX37.PAS

```
PROGRAM rezec4;
 LABEL zero;
 VAR a, b : real; i:byte;
   ai:array[1..4] of real; {limita inferioara a intervalului}
   bi:array[1..4] of real; { limita superioara a intervalului}
 FUNCTION f(x:real):real;
   begin
   f:=sqr(x)*sqr(x) - 9*sqr(x)*x - 2*sqr(x) + 120.*x - 130.
   end;
 PROCEDURE solutie(a,b:real; i:byte);
  label unu, doi;
  CONST eps=1.E-05;
  var c:real;
 begin
  unu: c:=(a+b)/2;
   if f(c)=0 then
    begin
    writeln(' x(',i:2,')=',c);     goto doi;
    end;
   if f(a)*f(c) < 0 then b:=c else a:=c;
   if abs(b-a)<eps then
     begin
     writeln(' x(',i:2,')=',(a+b)/2); goto doi;
     end
     else goto unu;
doi: end;
BEGIN {main program}
 writeln(' introdu limitele a, b pentru cele 4 intervale ');
 writeln('in care se afla o radacina:');
```

```
 writeln(' (-4., -3.), (1., 2.), (4., 5.), (7., 8.) ');
 for i:= 1 to 4 do
   begin
   zero: readln(a, b);
    { verifica corectitudinea intervalelor}
    if f(a)*f(b) > 0 then
      begin
    writeln(' interval dat gresit, reintrodu a si b: ');
      goto zero;
      end;
   ai[i]:=a; bi[i]:=b;
   end;
 for i:=1 to 4 do solutie(ai[i], bi[i],i);  readln;
END.
```

2. Un algoritm mai general de găsire a rădăcinilor unui polinom este metoda Birge-Vieta (vezi /4/, pag.46-49). Programul EX38.PAS foloseşte 2 funcţii, b0 şi c1 pentru a calcula valoarea polinomului si a derivatei sale pentru o valoare x0.

```
 EX38.PAS
 program Birge_Vieta;
 label 10, 20, 30;
 const eps=1.E-05; itermax=30;
 var x0, x:real;  {variabile globale}
     a,b:array[0..20] of real;
     i, iter, m, ic:byte;
function b0(x:real):real;
   var j:byte;  {variabila locala}
 begin
  b[m]:=a[m];
 for j:=m-1 downto 0 do b[j]:=b[j+1]*x+a[j];
  b0:=b[0];
  end;
function c1(x:real):real;
  var k:byte;   {variabile locale}
      c:array[1..20] of real;
begin
 c[m]:=b[m];
```

```
 for k:=m-1 downto 1 do c[k]:=c[k+1]*x+b[k];
 c1:=c[1]
end;
BEGIN
write(' introdu gradul polinomului,m=');readln(m);
writeln(' introdu coef. polinom am*x^m+...+a1*x+a0');
for i:=m downto 0 do
begin
 write(' introdu a(',i:2,')='); readln(a[i]);
end;
writeln('polinomul dat este:');
writeln(a[m],'*x^',m,'+',a[m-1],'*x^',m-1,'+...',a[0]);
 30:  write(' introdu x0 ='); readln(x0);
   for iter:=1 to itermax do
      begin
 20:  x:=x0-b0(x0)/c1(x0);
    if abs(x-x0)<eps then goto 10 else
      begin
      x0:=x; goto 20;
      end;
      end;
 10: writeln( ' la iteratia ',iter:2,' radacina este ',x:10:5);
write(' introdu 1 pentru alta radacina'); readln(ic);
if ic=1 then goto 30;
readln;
end.
```

3. Un program ce calculează

$$C_m^k = \frac{m!}{(m-k)!k!}$$

folosind funcţia fact(n).

EX39.PAS

```
PROGRAM comb_mk;
 {Uses CRT;}
 VAR m, k:integer;  c:real;
```

```
FUNCTION fact(n:integer):longint;
 VAR i:integer; f:longint;
 begin   {begin fact}
 f:=1;   for i:=1 to n do f:=f*i;   fact:=f;
 end;   {end fact}
BEGIN {main program}
  {clrscr;}  {sterge ecranul}
  write(' introdu m si k, numere intregi, pozitive; m>k);
  readln(m,k);    c:=fact(m)/fact(k)/fact(m-k);
  writeln(' comb. de ',m:3,' luate cate ',k:3,' = ', c);
  readln;
END.
```

6.3 Parametri funcţii şi parametri proceduri

O procedură sau funcţie poate apela o altă procedură sau funcţie dacă aceasta din urmă a fost deja declarată.

Sunt însă situaţii în care numele şi efectul procedurii sau funcţiei apelate nu sunt cunoscute la scrierea procedurii sau funcţiei apelante ci doar în momentul execuţiei programului. În aceste cazuri numele procedurii/funcţiei apelate va fi transmis ca parametru.

Astfel funcţia ce calculează prin metoda Simpson integrala

$$\int_a^b f(x)dx$$

va avea antetul:
FUNCTION Simpson(a, b:real; n:integer; f:fct):real;

Tipul fct este definit în programul apelant astfel:
TYPE fct=function(x:real):real;

Parametrii formali funcţii şi procedură pot fi numai parametri de tip valoare. Funcţiile şi procedurile standard nu pot fi transmise ca parametri actuali.

La apelarea unei funcţii/proceduri care are printre parametri formali o funcţie sau procedură, parametrul actual corespunzător trebuie să fie un identificator de funcţie sau procedură.

Observaţie: Folosirea parametrilor de tip procedură sau funcţie se face

sub controlul directivei de compilare {$F+} aşezată fie înainte de antetul programului fie înaintea declarării funcţiei actuale şi, în acest ultim caz, se foloseşte la sfârşitul declarării funcţiei directiva de compilare {$F-}.

Exemple

1. Să se calculeze, folosind metode Simpson, integrala

$$K(\theta) = \int_0^{\pi/2} \frac{d\phi}{\sqrt{1 - sin^2(\theta)sin^2(\phi)}}$$

pentru $\theta = 30$ şi 60 grade.

EX40.PAS

```
Program calcul_int_simpson;
uses crt;
const pi=3.14159;
type fct=function(x:real):real;
var a, b, t1, t2, t, s1, s2:real;
{$F+}
function k(x:real):real;
 begin
 k:=1/sqrt(1-sin(t)*sin(t)*sin(x)*sin(x))
 end;
{$F-}
function simpson(a, b:real; n:integer; f:fct):real;
 var s, h:real; i:integer;
 begin
 h:=(b-a)/n/2;  s:=f(a)+f(b);  i:=1;
 while i<2*n do
   begin
   if i mod 2 = 0 then s:=s + 2*f(a+i*h)
       else s:=s + 4*f(a+i*h);
   i:=i+1;
   end;
 simpson:=s*h/3;
 end; {end simpson}
BEGIN { incepe programul principal}
clrscr;
a:=0; b:=pi/2;
```

```
t1:=30; t:=t1*pi/180; s1:=simpson(a, b, 10, k);
t2:=60; t:=t2*pi/180; s2:=simpson(a, b, 10, k);
writeln(' K(',t1:4:1,')=',s1);
writeln(' K(',t2:4:1,')=',s2);
readln;
END.
```

Observaţie Salvaţi funcţia simpson într-un fişier cu numele simpson.inc. De câte ori aveţi nevoie de ea folosiţi directiva de copmpilare {$i simpson.inc} înainte de BEGIN (programul principal); în acest fel nu mai este nevoie să o tastaţi în program.
2. Pentru a calcula integrala

$$I = \int_{-1}^{+1} \frac{x^7\sqrt{1-x^2}dx}{(2-x)^{13/2}}$$

se vor folosi funcţiile xp(x:real; p:integer):real pentru ridicarea lui x la o putere întreagă, fs pentru funcţia de sub semnul integralei şi simpson pentru calculul integralei. Cu observaţia de mai sus programul se scrie:

```
EX41.PAS
Program calcul_integrala;
uses crt;
type fct=function(x:real):real;
var a, b, s1:real;
function xp(x:real; p:integer):real;
 var i:integer; px:real;
 begin
 px:=1; for i:=1 to p do px:=px*x;  xp:=px;
 end;
{$F+}
function fs(x:real):real;
 begin
 fs:=xp(x,7)*sqrt(1-sqr(x))/sqrt(xp(2-x,13));
 end;
{$F-}
{$i simpson.inc}
BEGIN
```

```
clrscr;
s1:=simpson(-1.,1.,10,fs);  writeln(' integrala este: ',s1);
readln;
END.
```

Următorul program calculează produsul a 2 matrici cu ajutorul procedurii **promat(a:matrice;la,ca:byte; b:matrice;lb,cb:byte; Var c:matrice; Var lc,cc:byte);** salvata in fisierele promat.pas si promat.inc
Datele sunt citite din fisierul promat.in (pe prima linie la si ca, apoi elementele matricii a pe linii (ca linii), apoi lb si cb si cb linii cu elementele matricii b). Matricile a si b si matricea produs sunt scrise in fisierul promat.out

```
procedure promat(a:matrice;la,ca:byte;b:matrice;lb,cb:byte;
 Var c:matrice;Var lc,cc:byte);
 var i, j, k:byte;
     s:real;
 begin
  if ca<>lb then
        begin
  write('matricile nu se pot inmulti, ca<>lb');
    halt;
        end
  else
        begin
   for i:=1 to la do
   for j:=1 to cb do
         begin
      s:=0;
    for k:=1 to ca do
    s:=s+a[i,k]*b[k,j];
    c[i,j]:=s;
         end;
         end;
end; {end procedura}
```

EX42.PAS

```
program matrici;  {calculeaza produsul matricilor ma si mb, rezultat in mc}
type matrice=array[1..10,1..10] of real;
var ma,mb,mab:matrice;
    la,ca,lb,cb,lab,cab:byte;
    i,j:byte;
    f,g:text;
{$i promat.inc}
begin
assign(f,'promat.in'); reset(f); readln(f,la,ca);
for i:=1 to la do
 begin
for j:=1 to ca do read(f,ma[i,j]);readln(f);
 end;
 readln(f,lb,cb);
for i:=1 to lb do
 begin
for j:=1 to cb do read(f,mb[i,j]);
 readln(f);
 end;close(f);
assign(g,'matr.out');rewrite(g);
writeln(g,' matricea a este:');
  for i:=1 to la do
   begin
   for j:=1 to ca do write(g,ma[i,j]:6:1); writeln(g);
   end; writeln(g);
write(g, 'matrice b este:');
   for i:=1 to lb do
    begin
   for j:=1 to cb do write(g,mb[i,j]:6:1); writeln(g);
    end; writeln(g);

 promat(ma,la,ca,mb,lb,cb,mab,lab,cab);

writeln(g,' matricea produs este:');
  for i:=1 to lab do
   begin
  for j:=1 to cab do write(g,mab[i,j]:6:1); writeln(g);
   end; close(g);
```

```
end.
```

Fişierul promat.in poate arăta aşa dacă vrem să verificăm rezultatul înmulţirii matricii a cu matricea unitate:

```
3 3
1 2 3
3 4 5
1 2 6
3 3
1 0 0
0 1 0
0 0 1
```

6.4 Definiţii recursive.

Un obiect este recursiv dacă este definit în funcţie de el însuşi. Recursivitatea este un instrument puternic îndeosebi în definiţiile matematice.
Exemple:

- numerele naturale, factorialul (n!=n*(n-1)! cu 0!=1 şi $n > 0$)
- polinoamele Legendre

$$P_m(u) = \frac{1}{m}[u(2m-1)P_{m-1}(u) - (m-1)P_{m-2}(u)]$$

cu $P_0(u) = 1$ şi $P_1(u) = u$

Apel recursiv înseamnă folosirea numelui procedurii (funcţiei) în cadrul textului procedurii (funcţiei); apelul recursiv este permis în Turbo Pascal.

Puterea recursivă stă în posibilitatea de a defini o mulţime infinită de obiecte printr-o declaraţie finită; un număr infinit de calcule poate fi descris printr-un program recursiv finit chiar dacă programul nu conţine repetiţii explicite. Pentru terminarea programului apelarea recursivă a unei proceduri trebuie condiţionată fie printr-o condiţie ce la un moment dat devine falsă fie asociind procedurii un parametru n şi apelând-o recursiv cu parametrul n-1.

Iată două exemple pentru aceste două posibilităţi.
1. Fiind dat un numă întreg şi pozitiv să se scrie numărul obţinut prin citirea cifrelor numărului dat de la dreapta la stânga (numărul răsturnat).

EX43.PAS

```
program nr_rasturnat;
 uses crt;
 var m:longint;
 procedure invers(n:longint);
   begin
   write(n mod 10);
   if n div 10 <> 0 then invers( n div 10);
   end;
 begin
 clrscr;
 write('introdu numarul intreg: ');readln(m);
 writeln('numarul dat este ',m);
 wrute('numarul rasturnat este: '); invers(m);
 readln;
 end.
```

2. Introduceţi în EX40.PAS forma recursivă a funcţiei fact(n).

```
FUNCTION fact(n:integer):longint;
 begin
 if n=0 then fact:=1 else fact:=n*fact(n-1);
 end;
```

Recursivitatea poate fi întotdeauna transformată în iteraţie. În majoritatea cazurilor forma nerecursivă a unui program este mai eficientă decât forma recursivă în ceea ce priveşte timpul de execuţie şi memoria ocupată. Varianta recursivă este preferată acolo unde înlocuirea ei cu iteraţia ar cere un efort deosebit sau tehnici speciale de programare, algoritmul pierzându-şi claritatea exprimării.

Un astfel de exemplu în care folosirea funcţiei recursive e justificată este problema partiţiilor unui număr natural N, prin care se înţelege totalitatea posibilităţilor de exprimare a numărului N ca o sumă de alte numere naturale, fiecare din ele nedepăşind o valoare M.

Algoritmul este următorul/1/: se ataşează o funcţie P(n, m) numărului de partiţii, valoarea ei fiind dată de următoarele reguli:

P(n, m) = 1	dacă m=1 sau n=1
P(n, m) = 1 + P(n, n-1)	dacă $n <= m$
P(n, m) = P(n, m-1) + P(n-m, m)	dacă $n > m$

Varianta iterativă ar fi greoaie în timp ce varianta recursivă este foarte simplă.

EX44.PAS

```
program partitii;
 uses crt;
 var l, k:integer;
 function p(n, m:integer):longint;
  begin
  if (n=1) or (m=1) then p:=1
  else
    if n<=m then p:=1 + p(n, n-1)
    else p:= p(n, m-1) + p(n-m, m);
  end;  {end function p}
begin  {program principal}
 clrscr;
 write('introdu n si m, intregi si pozitive, n>m ');
 readln(L, K);
 writeln(' numar de partitii p(', L:3,',', K:3,')=', p(L,K));
 readln;
 end.
```

Exemple cunoscute de apelare recursivă justificată sunt programele ce rezolvă probleme ca turnurile din Hanoi şi asezarea pe tabla de sah a opt regine, probleme descrise în /1/.

Capitolul 7

UNIT-uri Turbo Pascal, unit-ul GRAPH

Primele versiuni ale compilatorului Turbo Pascal nu permiteau scrierea programelor mai mari de 64 kbytes deoarece procesorul 8086 limita dimensiunea unui segment la această valoare. Începând cu versiunea a 4-a s-a introdus noţiunea de unit. Prin **unit** se înţelege o colecţie de constante, declaraţii de tip şi variabile, proceduri şi funcţii, care poate fi compilată separat şi care poate fi utilizată de un program principal sau de un alt unit prin specificarea numelui unit-ului într-o clausă **uses**. Lungimea unui unit rămâne limitată la 64Kb dar un program poate folosi un număr arbitrar de unit-uri funcţie doar de memoria disponibilă a calculatorului folosit. Prin utilizarea unit-urilor creşte viteza de compilare.

Exisă opt unit-uri standard în Turbo Pascal, fiecare cu un profil şi o structură bine delimitate: **SYSTEM, DOS, OVERLAY, CRT, GRAPH, PRINTER, GRAPH3, TURBO3.**

- Unit-ul System conţine toate procedurile şi funcţiile standard din Turbo Pascal. El se încorporează automat în toate programele fără a fi necesară o clauză Uses pentru el;
- Unit-ul Dos conţine proceduri şi funcţii echivalente cu apelurile DOS cele mai obişnuite: citire de date, sistemul de întreruperi, etc. Folosirea lui se face cu clauza **Uses Dos;**
- Unit-ul Overlay permite folosirea tehnicii de scriere a programelor mari din bucăţi. Se foloseşte cu clauza **Uses Overlay;**.

Aceste trei unit-uri se găsesc în fişierul **Turbo.tpl**

- Unit-ul Crt (character) permite utilizarea funcţiilor şi procedurilor pentru comanda funcţionării ecranului alfanumeric: ştergerea ecranului cu procedura **clrscr** (clear screen), funcţiile **KeyPressed:boolean;** şi **ReadKey:char;** Se foloseşte cu clauza **Uses crt;** după antetul programului.
- Unit-ul Graph conţine procedurile şi funcţiile grafice. Acest unit se află în fişierul **Graph.tpu**. Subprogramele din acest unit au nevoie de informaţiile din fişierele *.BGI şi *.CHR Unit-ul Graph se foloseşte cu clauza **Uses Graph;**
- Unit-ul Printer permite redirectarea scrierilor în fişierul text cu numele lst direct la imprimantă.
- Unit-urile Graph3 şi Turbo3 permit folosirea subprogramelor grafice din versiunea a 3-a a compilatorului Turbo Pascal.

Exemplu: Uses Dos, Crt, Graph;

Programele în Turbo Pascal permit atât folosirea unit-urilor standard enumerate mai sus cât şi scrierea şi folosirea unor unit-uri scrise de programator.

Structura unui unit este asemănătoare cu a unui program dar există câteva diferenţe semnificative.

unit identificator_unit;
interface
uses listă de unit-uri; { opţional }
{ declaratii publice }
implementation
uses lista de unit-uri { opţional }
{ declaratii locale }
{ implementarea procedurilor si functiilor }
begin
{ codul de calcul }
end.

Antetul unit-ului începe cu cuvântul rezervat **unit** urmat de numele unitului (un identificator) ca şi un program în Turbo Pascal. Imediat după

antet urmează cuvântul cheie **interface** care semnalează începutul secţiunii interface a unit-ului adică a părţii vizibile pentru orice program ce foloseşte acest unit.

Un unit poate folosi alte unit-uri specificându-le într-o clauză **uses**.

- Clauza uses poate apare imediat după cuvântul cheie interface; în acest caz orice constantă sau tip de date declarate în secţiunile interface ale acelor unit-uri poate fi folosită în secţiunea interface a acestui unit.

- Clauza uses poate apare imediat după cuvântul cheie implementation; în acest caz declaraţiile din acele unit-uri pot fi folosite doar în secţiunea implementation.

Partea 'publică' a unui unit începe cu cuvâtul rezervat **interface** şi se termină la cuvântul rezervat **implementation**. Această parte este 'vizibilă' oricărui program ce foloseşte acest unit. In această secţiune declaraţiile de constante, tipuri de date, variabile, proceduri, funcţii pot fi făcute în orice ordine ca şi într-un program. Procedurile şi funcţiile sunt declarate aici dar corpul lor se găseşte în secţiunea implementation fără a fi necesară declaraţia **forward**. Aşa dar în interface apare doar antetul procedurilor şi funcţiilor.

Secţiune **implementation** (partea 'privată') a unui unit începe cu cuvântul rezervat **implementation**. Tot ce s-a declarat în interface este vizibil în implementation. Mai mult aici pot fi făcute şi declaraţii suplimentare proprii, invizibile pentru programul ce foloseşte unit-ul dar folosite la scrierea procedurilor şi funcţiilor declarate în interface.

Exemplu:

```
unit sir;

interface

type vector=array[1..100] of real;
procedure ordonare(x:vector; n:byte; VAR xo:vector);
function xmin(x:vector; n:byte):real;
function xmax(x:vector; n:byte):real;

implementation
```

```
procedure ordonare;
var i, j:byte;  aux:real;
begin
for j:=n-1 downto 1 do
  for i:=1 to j do
    if x[i] > x[i+1] then
      begin
      aux:=x[i];  x[i]:=x[i+1];  x[i+1]:=aux;
      end;
for i:=1 to n do xo[i]:=x[i];
end;

function xmin;
var i:byte;  xm:real;
begin
xm:=x[1];
for i:=2 to n do if xm > x[i] then xm:=x[i];
xmin:=xm;
end;

function xmax;
var i:byte;  xm:real;
begin
xm:=x[1];
for i:=2 to n do if xm < x[i] then xm:=x[i];
xmax:=xm;
end;

end.
```

Salvăm fisierul sub numele sir.pas apoi trecem în fereastra COMPILE şi schimbăm Destination Memory în Destination Disk plasându-ne pe Destination Memory şi tastând ENTER. Compilarea programului din fişierul sir.pas va conduce la crearea fişierului sir.tpu
Următorul program foloseşte unit-ul sir pentru a ordona vectorul **a** şi a găsi valorile minimă şi maximă din vector. Numărul de elemente ale vectorului **a** şi valorile elementelor sale sunt citite dintr-un fişier creat de utilizator astfel: pe prima linie este scrisă valoarea lui n iar pe liniile următoare valoarea câte

unui element al vectorului **a**. În total n+1 linii.

EX45.PAS

```
program test_unit_sir;
uses sir;
var a, aord:vector;  {tipul vector e definit in unit sir}
    i, j, n:byte;   f:text;   nume:string;
    amin, amax:real;
begin
write(' introdu nume fisier input: '); readln(nume);
assign(f, nume); reset(f);
readln(f, n); for i:=1 to n do readln(f, a[i]); close(f);
writeln(' sirul ordonat este: ');  ordonare(a, n, aord);
for i:=1 to n div 8 do
  begin
  for j:=8*(i-1)+1 to 8*(i-1)+8 do
     write(aord[j]:9:2,' ');
  writeln;
  end;
for i:=n+1 - n mod 8 to n do
  write(aord[i]:9:3,' ');
writeln;
{ pentru a vedea ca vectorul a este parametru valoare
si deci nu e mdificat in procedura ordonare, sa-l scriem}
writeln(' sirul initial este: ');
for i:=1 to n div 8 do
  begin
  for j:=8*(i-1)+1 to 8*(i-1)+8 do
     write(a[j]:9:2,' ');
  writeln;
  end;
for i:=n+1 - n mod 8 to n do write(a[i]:9:3,' ');
amin:=xmin(a,n); amax:=xmax(a,n);
writeln; writeln(' amin=',amin:9:3);
         writeln(' amax=', amax:9:3);
readln;
end.
```

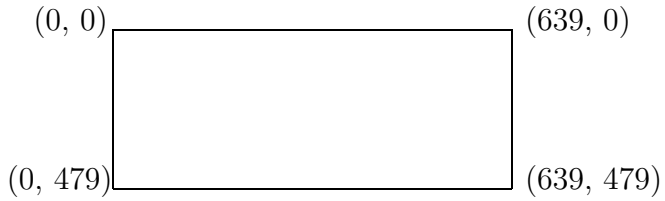
7.1 Unit-ul GRAPH

Programele care utilizează subprogramele din acest unit trebuie să aibe directiva **Uses Graph;** după antetul programului.

Subprogramele din unit-ul Graph realizează:

1. iniţializarea modului grafic
2. tratarea erorilor grafice
3. definirea de ferestre şi pagini
4. reprezentarea punctelor
5. reprezentarea liniilor
6. reprezentarea de cercuri şi alte curbe
7. reprezentarea de poligoane şi haşurări
8. scrierea grafică

Ecranul grafic pentru placă VGA (640x480 pixeli) este:



7.1.1 Iniţializare mod grafic

Pentru iniţializarea modului grafic se poate folosi:

```
procedure grstart;
   begin
   gd:=detect; InitGraph(gd, gm, '\TP\BGI');
   If graphresult <>grok then halt(1);
   end;
```

cu antetul procedurii InitGraph:
procedure InitGraph(var gd:integer; var gm:integer; cd:string);
în care **gd** şi **gm** reprezintă codul corespunzător driverului şi modului grafic iar **cd** reprezintă calea spre fişierele de grafică. Variabila gd a fost iniţializată cu constanta predefinită **const detect=0** pentru a se omite apelarea procedurii DetectGraph înaintea apelării procedurii InitGraph. Procedura DetectGraph cu antetul:
procedure DetectGraph(var gd, gm:integer);
determină valorile pentru variabilele gd şi gm.

Funcţiile **GetMaxX** şi **GetMaxY** dau valorile maxime pentru X şi, respectiv, Y în modul grafic actual. Pentru placa VGA cu 640x480 pixeli aceste valori sunt respectiv 639 şi 479.

Funcţia **GetDriverName**, apelată după activarea procedurii InitGraph, dă numele driverului grafic actual. Antetul ei este:
function GetDriverName:string;

Funcţia **GetGraphMode** dă codul modului grafic actual. Antetul ei este:
function GetGraphMode:integer;
Valoarea ei variază între 0 şi 5 în funcţie de driverul grafic actual.

Procedura **GetAspectRatio** dă rezoluţia ecranului grafic din care se poate calcula raportul Xasp/Yasp ce dă alungirea formelor circulare; raportul Xasp/Yasp e necesar la unele 'rotunjiri' în trasarea cercurilor, sectoarelor de cerc, arcelor. Procedurile folosite pentru aceasta sunt:
GetAspectRatio(var Xasp, Yasp:word);
SetAspectRatio(Xasp, Yasp:word);

Revenirea la ecranul alfanumeric se face cu procedura **CloseGraph**

EX46.PAS

```
Program verifgraph;
 Uses dos, graph;
 Var ggm, gd, gm:integer;
     Xaspect, Yaspect:word;
     aspratio:real;
```

```
    gdn:string;
Procedure grstart;
  begin
  gd:=detect; InitGraph(gd, gm, '\tp\bgi');
  if graphResult <>grok then halt;
  end;
Begin
grstart;
ggm:=GetGraphMode; gdn:=GetDriverName;
GetAspectRatio(Xaspect, Yaspect);
aspratio:=Xaspect/Yaspect;
CloseGraph;
writeln('graphdriver=', gdn);
writeln('graphmode=', ggm);
writeln('aspectratio: ', aspratio:2:2);
readln;
End.
```

7.1.2 Erori grafice

Erorile în modul grafic sunt returnate de funcţia **GraphResult** sub forma unui cod de eroare. Dacă codul returnat este zero înseamnă că operaţia grafică a reuşit; dacă codul este mai mic decât zero înseamnă că operaţia grafică a eşuat. Antetul funcţiei este:
function GraphResult:integer;
Câteva valori predefinite ale funcţiei sunt:

```
Const
      grOK=0;           {operaţie grafică reuşită, nici o eroare}
 grNoInitGraph=-1;    { grafica BGI neinstalată cu InitGraph}
 grNotDetected=-2;        {grafica hardware nedetectată}
grFileNotFound=-3;     {fişierul driver *.BGI nu a fost găsit}
```

De notat că GraphResult e pus la zero după ce este apelat aşa că rezultatul trebuie stocat într-o variabilă temporară pentru a fi testat.

7.1.3 Definire ferestre

Pentru a defini şi lucra cu ferestre, prin fereastră înţelegându-se o zonă dreptunghiulară a ecranului (în particular tot ecranul), se folosesc procedurile:

procedure SetViewPort(x1,y1,x2,y2:integer; clip:boolean);
unde:

x1, y1	sunt coordonatele absolute stânga-sus ale ferestrei active
x2, y2	sunt coordonatele absolute dreapta-jos ale ferestrei active
clip	determină dacă liniile din afara ferestrei sunt sau nu vizibile.

Constantele predefinite pentru stabilirea valorii variabilei **clip** sunt:
const
clipon=true; {tăierea este activă; exterior invizibil}
clipoff=false; {tăierea este pasivă; exterior vizibil}
Procedura defineşte fereastra grafică.
procedure GetViewSettings(var infofer:ViewPortType);
permite obţinerea de informaţii referitoare la fereastra actuală şi felul tăierii. Variabila **infofer** este de tipul predefinit ViewPortType şi va conţine informaţii referitoare la fereastră.

```
Type ViewPortType = RECORD
                    x1, y1, x2, y2 : integer;
                    clip : boolean;
                    END;
```

procedure ClearDevice;
şterge ecranul grafic actual şi poziţionează pointerul în poziţia (0, 0).
procedure ClearViewPort
şterge fereastra grafică actuală. Culoarea ferestrei va fi culoarea de fond; este apelată procedura **rectangle** pentru trasarea unui dreptunghi corespunzător ferestrei şi pointerul este mutat în colţul stânga-sus al ferestrei, punctul de coordonate relative (0, 0).

EX47.pAS

```
 program ex_viewport;
uses graph;
 var gd,gm,centrux,centruy,i:integer;
    inf:viewporttype;
    cx1,cy1,cx2,cy2:string[5];
 begin
 gd:=detect; initgraph(gd,gm,'\tp\bgi');
 centrux:=getmaxx div 2; centruy:=getmaxy div 2;
rectangle(centrux-179, centruy-169, centrux+179, centruy-1);
```

```
setviewport(centrux-180,centruy-170, centrux+180,centruy, clipon);
{incercati si clipoff} randomize; readln; for i:=1 to 10 do
lineto(random(getmaxx)-180,random(getmaxy)-180);
getviewsettings(inf); readln;
str(inf.x1,cx1); str(inf.y1,cy1);
str(inf.x2,cx2); str(inf.y2,cy2);
 outtextxy(10,30,cx1); outtextxy(10,50,cy1);
 outtextxy(10,100,cx2); outtextxy(10,120,cy2);readln;
setviewport(0,0,getmaxx,getmaxy,clipon);
 rectangle(centrux-179,centruy+10,centrux,centruy+170);
 {a doua fereastra}
setviewport(centrux-180,centruy+10,centrux,centruy+170,clipon);
randomize; readln; for i:=1 to 10 do
lineto(random(getmaxx)-180,random(getmaxy)-180);
 readln;
setviewport(0,0,getmaxx, getmaxy,clipon);
rectangle(centrux+10,centruy+10, centrux+181,centruy+171);readln;
setviewport(centrux+10,centruy+10,centrux+181,centruy+171,clipon);
 readln;
for i:=1 to 10 do lineto(random(centrux),random(centruy));
 readln;
clearviewport; readln; cleardevice; readln;
closegraph;
end.
```

7.1.4 Reprezentare puncte

Pentru a desena puncte se foloseşte procedura:
procedure PutPixel(x, y:integer; cul:word);
care pune un punct, de culoarea dată de variabila **cul** în punctul de coordonate x, y. Variabila cul ia valori între zero şi valoarea dată de funcţia **GetMaxColor**. (max=15 pentru placa VGA). Antet funcţie:
function GetMaxColor:word;

EX48.PAS

```
program test_pixel;
 uses graph;
```

```
var gd, gm:integer; xMax, yMax, MaxColor:integer;
         i:word; x, y, culoare :word;
procedure grstart;
  begin
  gd:=detect; InitGraph(gd, gn,'\tp\bgi');
  if graphresult <> grOK then halt(1);
  end;
function ranCul:word;
  begin
   randCul:=random(MaxColor)+1;
  end;
BEGIN
  grstart;
  xMax:=getMaxX; yMax:=getMaxY; MaxColor:=getMaxColor;
  randomize;
  i:=0; while i < 100 do
    begin
    i:=i+1; putpixel(random(xMax)+1, random(yMax)+1, randCul);
    end;
readln; closegraph; writeln('program executat');
END.
```

7.1.5 Reprezentare linii, culori, stiluri şi grosimi, deplasarea în fereastra grafică

Pentru a desena linii se folosesc procedurile:
Line(x1,y1,x2,y2:integer):
desenează o linie între punctele de coordonate (x1, y1) şi (x2, y2). **Atenţie:** pointerul nu rămâne în punctul de coordonate (x2, y2) ci revine la poziţia (0, 0). Culoarea în care este desenată linia poate fi stabilită cu procedura:
SetColor(color:word):
Culorile de desen merg de la 0 la 15 şi sunt funcţie de driverul grafic curent şi de modul grafic curent. Ele sunt definite de constantele:
CONST

black=0;	blue=1;	green=2;	cyan=3;
red=4;	magenta=5;	brown=6;	lightgray=7;
darkgray=8;	lightblue=9;	lightgreen=10;	lightcyan=11;
lightred=12;	lightmagenta=13;	yellow=14;	white=15;
blink=128;			

iar stilul şi grosimea se stabilesc cu procedura:
SetLineStyle(linestyle:word; pattern:word; thickness:word);
Constantele pentru LineStyle sunt:
CONST

solidln=0;	dottedln=1;	centerln=2;
dshedln=3;	userbitln=4;	{user-define line style}

iar pentru grosime:
CONST
normwidth=1; thickwidth=3;

Variabila **pattern** este ignorată cât timp linestyle este diferit de 4. Când linestyle = 4, linia este trasată folosind un model 16-bit definit de parametrul Pattern. De exemplu dacă **pattern=$AAAA** atunci modelul 16-bit arată astfel:

```
1010101010101010 {pentru normwidth}

1010101010101010
1010101010101010
1010101010101010 {pentru thickwidth}
```

EX49.PAS

```
program ex_line;
 uses graph;
 var gd,gm,i:integer;
     x1,y1,x2,y2:integer;
 BEGIN
 gd:=detect; InitGraph(gd,gm,'\tp\bgi');
 line(0,0,100,0); outtextxy(150,0,'standard');
 setlinestyle(dottedln,0,normwidth);
line(0,50,100,50); outtextxy(150,50,'dottedline, normwidth');
 setlinestyle(dottedln,0,thickwidth);
```

```
line(0,100,100,100); outtextxy(150,100,'dottedline, thickwidth');
 setlinestyle(userbitln,$c3,thickwidth);
line(0,150,100,150); outtextxy(150,150,' userbitln, thickwidth');
 for i:=0 to 3 do begin setcolor(i+2);
setlinestyle(i,0,normwidth);line(10,20*i+250,210,20*i+250);
setlinestyle(i,0,thickwidth);line(310,20*i+250,510,20*i+250); end;
readln; closegraph;
end.
```

LineTo(x, y:integer);
desenează o linie din poziţia curentă a pointerului până la un punct de coordonate (x, y). Aceleaşi observaţii pentru culoare şi stil ca la procedura **Line**.

LineRel(Dx, Dy:integer);
desenează o linie din poziţia curentă a pointerului până la un punct definit de distanţele Dx, Dy. Aceleaşi observaţii pentru culoare şi stil ca la procedura **Line**.

Deplasarea pointerului în punctul de coordonate (x, y) se face cu:
MoveTo(x, y:integer):
Dacă este definită o fereastră, valorile (x, y) sunt relative la fereastră.
MoveRel(Dx, Dy:integer);
mută pointerul din poziţia curentă la un punct definit de distanţele Dx, Dy.

7.1.6 Reprezentare cerc, arc de cerc, elipsă, sector de cerc, model de umplere a suprafeţelor închise

Pentru a desena un cerc, un arc de cerc, un arc de elipsă sau un sector de cerc se folosesc următoarele proceduri.
procedure Circle(x, y:integer; raza:word);
desenează un cerc cu centru în punctul de coordonate (x, y) şi cu raza dată.
procedure Arc(x, y:integer; ustart, ufinal,r:word);
desenează un arc de cerc cu centrul în punctul de coordonate (x, y) şi raza dată începând de la unghiul ustart şi sfârşind la unghiul ufinal, ambele unghiuri date în grade şi mergând în sens trigonometric.
procedure Ellipse(x,y:integer;ustart,ufinal:word;xraza,yraza:word);
desenează un arc de elipsă, (x, y) sunt coordonatele centrului, ustart, ufinal

sunt unghiurile de început şi sfârşit luate în sens trigonometric, xraza, yraza sunt axele orizontală şi, respectiv, verticală ale elipsei.
procedure PieSlice(x, y:integer; ustart, ufinal,r:word);
desenează şi haşurează un sector de cerc. Modelul şi culoarea de haşurat sunt definite cu:
procedure SetFillStyle(pattern:word; color:word);
Modelul implicit este compact (solid) şi culoarea implicită este culoarea cu cod maxim din paletă.
Constantele de haşurare sunt:
CONST

```
 emptyfill=0;          {umple in culoarea de fond}
 solidfill=1;          {umple cu culoare\}
 linefill=2;           {umple aria cu modelul - - -}
 ltslashfill=3;        {umple aria cu modelul /// }
 slashfill=4;          {umple aria cu modelul /// mai groase }
 bkslashfill=5;        {umple aria cu modelul \\\ }
 ltbkslashfill=6;  {umple aria cu modelul \\\ dar mai subtiri}
 hatchfill=7;          {umple aria cu modelul ||| }
 xhatchfill=8;         {umple aria cu modelul xxx }
 ...                    ...
 userfil=12;           {modelul e dat de programator }
```

procedure SetFillPattern(pattern:FillPatternType; color:word);
umple figura grafică cu un model definit de programator în culoarea definită de parametrul **color**.
Type FillPatternType=array[1..8] of byte;
Motivul de umplere se bazează pe valoarea celor 8 bytes (octeţi) conţinuţi în vectorul **pattern**; fiecare octet corespunde la 8 pixeli de motiv. Dacă un bit are valoarea 1 pixelul are culoarea cernelii, dacă el are valoarea 0 pixelul are culoarea fondului.

EX50.PAS

```
program ex_setmodel;
 uses graph;
 const
dungi:fillpatterntype=($aa,$55,$aa,$55,$aa,$55,$aa,$55);
 var gd,gm,i:integer;
```

```
 begin
 gd:=detect; InitGraph(gd,gm,'c:\tp\bgi');
 setcolor(4); setfillpattern(dungi,3);
pieslice(100,100,0,270,100); for i:=0 to 11 do begin
setfillstyle(i,i);pieslice(300,200,0,270,100);readln; end; readln;
closegraph;
end.
```

procedure FillEllipse(x, y:integer; xraza, yraza:word);
desenează şi haşurează o elipsă în culoarea curentă şi modelul de haşurare definit de procedurile **SetFillStyle** sau **SetFillPattern**.

7.1.7 Reprezentare poligoane

Pentru a desena poligoane se folosesc următoarele proceduri.
rectangle(x1, y1, x2, y2:integer);
desenează un dreptunghi cu
(x1, y1) coordonatele colţului stânga-sus
(x2, y2) coordonatele colţului dreapta-jos
Bar(x1, y1, x2, y2:integer);
desenează un dreptunghi şi îl haşurează în culoarea şi modelul stabilite cu procedurile SetFillStyle sau SetFillPattern. (x1, y1) şi (x2, y2) au aceeaşi semnificaţie ca în procedura rectangle.
Bar3d(x1, y1, x2, y2:integer; ad:word; top:boolean);
desenează un paralelipiped dreptunghi şi îl haşurează în culoarea şi modelul curente. (x1, y1) şi (x2, y2) au aceeaşi semnificaţie ca în procedurile rectangle şi bar
ad este adâncimea corpului
top = TRUE înseamnă că deasupra paralelipipedului se poate aşeza un alt paralelipiped
top = FALSE înseamnă că deasupra paralelipipedului nu se poate aşeza alt paralelipiped. Există predefinite constantele:
CONST
topon = TRUE;
topoff=FALSE;
DrawPoly(nrpct:word; var puncte); desenează o linie poligonală în culoarea şi stilul curente.

nrpct reprezintă numărul vârfurilor liniei poligonale
puncte este un parametru variabilă fără tip care conţine coordonatele fiecărui vârf în linia poligonală. De notat că pentru a desena o figură închisă cu n vârfuri trebuie să se transmită n+1 coordonate şi **puncte(n+1)=puncte(1)**

EX51.PAS

```
program ex_poligon;
 uses graph;
 const triunghi:array[1..4] of pointtype=
((x:50;y:100), (x:200; y:100), (x:200; y:250), (x:50; y:100));
 var gd, gm:integer;
 BEGIN
 gd:=detect; InitGraph(gd, gm, '\tp\bgi');
 if graphresult <> grOK then halt(1);
 drawpoly(sizeof(triunghi) div sizeof(poittype), triunghi);
 readln; closegraph;
 END.
```

Notă: funcţia Sizeof(x) dă numărul de octeţi ocupat de argument; x poate fi o referire de variabilă sau un identificator de tip.

procedure FillPoly(nrpct:word; var puncte);
desenează şi haşurează un polinom în culoarea şi modelul curente.

EX52.PAS

```
program ex_poligon2;
uses graph;
 const triunghi:array[1..4] of pointtype=((x:50;y:100),(x:200;y:100),
                                          (x:200;y:250),(x:50;y:100));
      triunghi1:array[1..4] of pointtype=((x:250;y:100),(x:400;y:100),
                                          (x:400;y:250),(x:250;y:100));
var gd,gm,i:integer;
 BEGIN gd:=detect; InitGraph(gd,gm,'\tp\bgi');
drawpoly(sizeof(triunghi) div sizeof(pointtype), triunghi);
 circle(350,100,100); readln;
 for i:=1 to 11 do
  begin
```

```
 setfillstyle(i,i);
 fillpoly(sizeof(triunghi) div sizeof(pointtype),triunghi1);
 fillellipse(300,300,50,50);
readln;
  end;
readln;closegraph;
end.
```

7.1.8 Scrierea grafică

Pentru scrierea textelor în modul grafic sunt incluse fonturi (tipuri de caractere) 8*8 bit-mapped şi fonturi vectoriale.
Un caracter 8*8 bit-mapped este definit de o matrice 8*8 pixeli.
Un font 'vectorial' este definit de o serie de vectori care spun sistemului grafic cum să deseneze caracterul.
Scrierea şi selecţia caracterelor se face cu următoarele proceduri:
SetTextStyle(font:word; direction:word; charsize:word);
constantele pentru caracter, direcţie şi dimensiune sunt:

CONST

```
 DefaultFont=0;          {8*8 bit-mapped font}
 TriplexFont=1;          {font vectorial}
 SmallFont=2;            {font vectorial}
 SansSerifFont=3;        {font vectorial}
 GothicFont=4;           {font vectorial}

 HorizDir=0;
 VertDir=1;
```

userCharSize=0; permite programatorului să varieze lărgimea şi înălţimea caracterului în cazul folosirii fonturilor vectoriale. Aceasta se face cu:
SetCharSize(multX, divX, multY, divY:word);
în care:

multX : divX	este raportul cu care trebuie înmulţită lăţimea normală pentru fontul activ
multY : divY	este raportul cu care trebuie înmulţită înălţimea normală pentru fontul activ

De exemplu, pentru a face fontul de 2 ori mai lat se foloseşte 2 pentru multX

şi 1 pentru divX.
Apelul acestei proceduri pune mărimea caracterului curent la valoarea dată.
Textul grafic se realizează apelând una din procedurile următoare:
OutText(text:string);
unde text este textul trimis la ieşire la poziţia curentă a pointerului folosind fontul curent pus cu SetTextStyle.
OutTextXY(x, y:integer; text:string);
textul este scris începând din punctul de coordonate (în pixeli) (x, y).

EX53.PAS

```
program ex_font;
 uses graph;
 var gd, gm:integer;
 BEGIN
  gd:=detect; InitGraph(gd, gm, '\tp\bgi');
  if graphresult <> grOK then halt(1);
  setviewport(0, 0, 600, 400, clipon);
  settextstyle(9, 0, 3);
  outtextxy(10, 100,'8 * 8 bit-mapped font');
  readln;
  for i := 1 to 4 do
    begin
    clearviewport; setviewport(0, 0, 600, 400)
    settextstyle(i, 0, 4); outtextxy(10, 20, ' normal');
    setusercharsize(1, 3, 1, 1); outtextxy(10, 50,'i n g u s t');
    setusercharsize(5, 1, 5, 1); outtextxy(10, 100, 'marit ');
    end;
  readln; closegraph;
 END.
```

Alte programe cu diferite marimi de fonturi. Funcţiile
TextHeight(TextString:string):word
şi
TextWidth(TextString:string):word
dau înălţimea şi, respectiv, lăţimea string-ului TextString în pixeli prin înmulţirea numărului de pixeli pentru fontul curent (pus cu SetTextStyle) cu numărul de caractere din string. Sunt utile la aşezarea textului pe linii în modul grafic.

EX54.PAS

```
program fonturi;
  uses graph;
  var gd, gm, y, size : integer;
BEGIN
gd := detect; InitGraph(gd, gm, '\tp\bgi');
ifGraphResult <> grok then halt(1);
y :=20; for size := 1 to 6 do
          begin
          SetTextStyle(DefaulFont, HorizDir, Size);
          OutTextXY(0, y, 'Size = '+ chr(size + 48));
                              {chr(48)='0' in ASCII}
          Inc(y, TextHeight('H') + 50);
          {creste y cu inaltimea unui caracter +50 pixeli}
          end;
 readln; closegraph;
 END.
```

EX55.PAS

```
 program dim_font;
 uses graph;
 var gd, gm : integer; nh, nw : word;
 BEGIN
 gd := detect; InitGraph(gd, gm, '\tp\bgi');
 settextstyle(0, 0, 1); nh := textheight('font 0, size 1');
                        nw := textwidth( 'font 0, size 1');
 writeln('nh=', nh, '   nw=', nw);
 readln; closegraph;
 END;
```

EX56.PAS

```
PROGRAM ex_write_graph;
uses graph;
 var gd,gm:integer;
 BEGIN
gd:=detect; initgraph(gd,gm,'c:\tp\bgi');
 if graphresult<>grok then halt(1);
```

```
 settextstyle(triplexfont,horizdir,4);
{settextstyle(smallfont,horizdir,4);}
{settextstyle(sansseriffont,horizdir,4);}
{settextstyle(gothicfont,horizdir,4);}
  outtextxy(10,20,' Colocvviu la Limbaje de Programare');
  setusercharsize(1,3,1,1);
outtextxy(10,60,' in saptamana 19-22 mai 2004');
setusercharsize(5,1,5,1); outtextxy(10,70,'Noroc');
setfillstyle(1,4); pieslice(530,200,0,30,100);
pieslice(530,200,60,90,100); pieslice(530,200,120,150,100);
pieslice(530,200,180,210,100); pieslice(530,200,240,270,100);
line(530,200,580,300); line(580,300,630,400); setfillstyle(1,2);
{line(550,280,580,290); line(580,290,590,280);
line(590,280,550,280);} pieslice(580,300,50,80,70);
pieslice(600,340,150,180,70);
readln; closegraph;
END.
```

EX57.PAS

```
program ferestre;
 uses graph;
 const
fer1:viewporttype=(x1:10;y1:80;x2:300;y2:100;clip:clipon);
fer2:viewporttype=(x1:310;y1:110;x2:600;y2:300;clip:clipon);
 var gd,gm:integer;
 begin
 gd:=detect; InitGraph(gd,gm,'c:\tp\bgi');setcolor(11);
 with fer1 do {constanta fer1}
 begin
 setfillstyle(2,3); setlinestyle(3,0,1);
  bar(succ(x1),succ(y1),pred(x2),pred(y2)+100);
 rectangle(succ(x1),succ(y1),pred(x2),pred(y2)+100);
 setviewport(x1,y1,x2,y2+100,clipon);setlinestyle(0,0,1);
 rectangle(0,0,290,120);  outtextxy(10,10,'prima fereastra');
 end;
 setcolor(14);
 setviewport(0,0,getmaxx,getmaxy,clipon);
 {revenirea la ecran pentru a crea un nou viewport}
```

```
 with fer2 do  {constanta fer2}
 begin
 setfillstyle(8,2); setlinestyle(4,$c3,1);
  bar(succ(x1),succ(y1),pred(x2),pred(y2));
  rectangle(succ(x1),succ(y1),pred(x2),pred(y2));
 setviewport(x1,y1,x2,y2,clipon);
 outtextxy(10,50,'a doua fereastra si in afara cadrului ei?');
 setfillstyle(4,0);bar(50,100,200,150);setviewport(50,100,200,150,clipoff);
 setcolor(7);
 outtextxy(320,120,'a treia fereastra si in afara ei');
 end; readln; closegraph;
 end.
```

Următorul program reprezintă grafic funcţia

$$f(x) = x^4 - 9x^3 - 2x^2 + 120x - 130$$

şi axele de coordonate.

```
EX58.PAS
program repr_fct;
uses graph;
const n=49; dx=0.25;
type vectorw=array[1..n] of word;
     vectorr=array[1..n] of real;
var i:byte; xp,yp:vectorw; xarg,yf:vectorr;
    xpmax,xpmin,ypmax,ypmin,xscal,yscal,f1:real;
    xreper:array[1..3] of integer;
function xlap(p:byte; x:real):real;
  var i:byte; xp:real;
   begin
    xp:=1.; for i:=1 to p do xp:=xp*x;
    xlap:=xp;
   end;
function f(x:real):real;
    begin
  f:=xlap(4,x)-9*xlap(3,x)-2*x*x+120*x-130;
    end;
procedure gr_fct;
```

```
 var i:byte; gd,gm:integer;
  begin
  gd:=detect; InitGraph(gd,gm,'c:\tp\bgi');
  setviewport(0,0,639,479,clipon);
  setlinestyle(1,1,1);
  line(0,225,600,225); line(300,0,300,400);
  line(xreper[1],220,xreper[1],230); outtextxy(xreper[1]-10,240,'-4.');
  line(xreper[2],220,xreper[2],230); outtextxy(xreper[2]-10,240,'4.');
  line(xreper[3],220,xreper[3],230); outtextxy(xreper[3]-10,240,'8.');
  setcolor(2);
  for i:=1 to n do circle(xp[i],yp[i],2);
  outtextxy(330,50,'f(x)=x^4-9x^3-2x^2+120x-130');
  readln;
  closegraph;
  end;
BEGIN
xpmax:=8.; xpmin:=-4.;
ypmax:=-1E30; ypmin:=1E30;
for i:=1 to n do
  begin
 xarg[i]:=xpmin+(i-1)*dx;
 f1:=f(xarg[i]);yf[i]:=f1;
    if ypmax<f1 then ypmax:=f1;
    if ypmin>f1 then ypmin:=f1;
  end;
xscal:=400/12.; yscal:=300/(ypmax-ypmin);
xreper[1]:=300-trunc(4*xscal);
xreper[2]:=300+trunc(4*xscal);
xreper[3]:=300+trunc(8*xscal);
for i:=1 to n do   begin
xp[i]:=300+trunc(xarg[i]*xscal);
yp[i]:=225-trunc(yf[i]*yscal);
end;
 gr_fct;
end.
```

Capitolul 8

Elemente de prelucrare statistică a datelor experimentale cu programe în Turbo Pascal

8.1 Valoare medie, varianţă, abatere standard, fluctuaţie

Scopul acestei lecţii nu este de a trata riguros subiectul ci de a arăta cum să folosim prelucrarea statistică pentru a obţine din datele experimentale cele mai bune rezultate şi a cunoaşte limitările rezultatelor obţinute. De asemeni cum să folosim ceea ce am învăţat despre limbajul Turbo Pascal pentru a scrie programe utile în munca de laborator.

Adesea în procesul de măsurare a unei mărimi fizice obţinem valori diferite la repetarea măsurătorii în aceleaşi condiţii fizice. Aceasta fie pentrucă mărimea fizică respectivă are un caracter statistic (de exemplu numărul de dezintegrări într-o secundă ale unui element radioactiv, lungimea drumului între două ciocniri ale unui neutron în combustibilul nuclear sau în mediul moderator într-un reactor nuclear, etc.) fie din cauza preciziei limitate a aparaturii de măsură (de exemplu erorile de calibrare a ceasului, riglei, balanţei folosite la măsurarea intervalelor de timp, a lungimilor sau a maselor).

În ambele cazuri repetarea măsurătorilor în aceleaşi condiţii va conduce

la rezultate diferite. Dacă aceste măsurători se fac de un număr foarte mare de ori se constată că valorile pentru mărimea fizică respectivă fluctuează în jurul unei valori medii. Fie N numărul de repetări ale măsurătorii mărimii fizice în exact aceleaşi condiţii şi x_i valoarea obţinută într-o măsurătoare. Valoarea medie $\bar{x}$ a şirului finit de N măsurători va fi media aritmetică a valorilor x_i :

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{8.1}$$

iar abaterea valorii x_i de la $\bar{x}$ va fi Δ_i dată de:

$$\Delta_i = x_i - \bar{x} \Rightarrow x_i = \bar{x} + \Delta_i \tag{8.2}$$

Aceste abateri vor fi mai mari sau mai mici. O măsură a împrăştierii valorilor x_i faţă de valoarea medie $\bar{x}$ poate fi dată de ansamblul abaterilor sau de o mărime care să descrie acest ansamblu. Această mărime nu poate fi abaterea medie $\bar{\Delta}$ deoarece ea este zero indiferent care sunt valorile Δ_i.

$$\bar{\Delta} = \frac{1}{N}\sum_{i=1}^{N} \Delta_i = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x}) = \frac{1}{N}\sum_{i=1}^{N} x_i - \bar{x} = \bar{x} - \bar{x} = 0 \tag{8.3}$$

Abaterea pătratică medie, varianţa, este însă diferită de zero:

$$\sigma_x^2 = \overline{\Delta^2} = \frac{1}{N}\sum_{i=1}^{N} \Delta_i^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i^2 - 2x_i\bar{x} + \bar{x}^2) = \overline{x^2} - \bar{x}^2 > 0 \tag{8.4}$$

de unde şi $\overline{x^2} > \bar{x}^2$ adică media pătratelor este mai mare decât pătratul mediei unui şir de valori.
Pentru caracterizarea împrăştierii se mai foloseşte mărimea:

$$\sigma_x = \sqrt{\overline{\Delta^2}} \tag{8.5}$$

numită abatere standard şi abaterea standard relativă definită de:

$$\epsilon_x = \frac{\sigma_x}{\bar{x}} \tag{8.6}$$

numită şi fluctuaţie.

8.2 Propagarea erorilor

Presupunem o funcţie f(x, y) de două variabile independente. Fie $\bar{f}$, $\bar{x}$ şi $\bar{y}$ valorile medii ale funcţiei şi variabilelor. Prin repetarea măsurătorilor se obţin abaterile Δ_i şi Δ_j ale variabilelor x şi y (i şi j iau toate valorile întregi între 1 şi N, N fiind numărul de repetări ale măsurătorii).

Pe baza ipotezei că abaterile sunt mici faţă de mărimile respective, funcţia f(x, y) se poate dezvolta î serie Taylor. Obţinem:

$$f(x,y) = \bar{f} + \Delta_{ij}f = f(\bar{x}+\Delta_i, \bar{y}+\Delta_j) = f(\bar{x},\bar{y}) + \frac{\partial f}{\partial x}|_{\bar{x},\bar{y}}\Delta_i + \frac{\partial f}{\partial y}|_{\bar{x},\bar{y}}\Delta_j \quad (8.7)$$

neglijând termenii de ordin mai înalt.
Identificăm:

$$\bar{f} = f(\bar{x}+\bar{y}) \quad (8.8)$$

$$\Delta_{ij}f = \frac{\partial f}{\partial x}|_{\bar{x},\bar{y}}\Delta_i + \frac{\partial f}{\partial y}|_{\bar{x},\bar{y}}\Delta_j$$

Abaterea medie pătratică a funcţiei f este, prin definiţie:

$$\sigma_f^2 = \overline{(\Delta_{ij}f)^2} = \frac{1}{N}\sum_i\sum_j\left(\frac{\partial f}{\partial x}\Delta_i + \frac{\partial f}{\partial y}\Delta_j\right)^2$$

$$= \frac{1}{N}\left[\left(\frac{\partial f}{\partial x}\right)^2\sum_i(\Delta_i)^2 + \left(\frac{\partial f}{\partial y}\right)^2\sum_j(\Delta_j)^2\right]$$

$$\sigma_f^2 = \left(\frac{\partial f}{\partial x}\right)^2\sigma_x^2 + \left(\frac{\partial f}{\partial y}\right)^2\sigma_y^2 \quad (8.9)$$

conform definiţiei (8.4) şi a faptului că suma produselor $\Delta_i\Delta_j$ conţine termeni ce se vor anula.

Generalizând, pentru o funcţie $f(x_1, x_2, ..., x_m)$ de m variabile x_k (k=1,m):

$$\sigma_f^2 = \sum_{k=1}^{m}\left(\frac{\partial f}{\partial x_k}\right)^2\sigma_k^2 \quad (8.10)$$

Relaţia (8.10) reprezintă **teorema de propagare a abaterilor standard** sau, mai obişnuit, **teorema de propagare a erorilor**.

Să aplicăm această teoremă la o lucrare din laboratorul de mecanică. Se fac N măsurători ale înălţimii **h** de la care cade liber un corp şi ale timpului de cădere **t** . Vrem să determinăm astfel acceleraţia gravitaţională:

$$g = \frac{2h}{t^2} = f(h,t) \tag{8.11}$$

şi abaterea standard a valorii obţinute.

```
EX59.PAS
Program prop_er;
 {programul calculeaza valoarea medie si abaterea standard
 pentru f(x1, x2,...,xm) cu m <= 20 folosind n, n <= 100,
 masuratori pentru setul de variabile x1, x2, ... , xm}
 {Aici programul este folosit pentru a calcula g(h, t); pentru
 alta  functie se rescrie partea de declarare a functiei si
 derivatelor sale}

 Type vec100=array[1..100] of real;
      vec20=array[1..20] of real;
 Var xmediu, sigx2:vec20;
     valx:vec100;
     i, j, m, n:integer;
     fmediu, sigfm2, sig1fm2, sig2fm2:real;
 Procedure med_abstd( x:vec100; n:integer; Var xm, sigx2:real );
   Var sx:real;  i:integer;
   begin
   sx:=0; for i:=1 to n do sx:=sx + x[i]; xm:=sx/n;
   sx:=0; for i:=1 to n do sx:=sx + (x[i] - xm)*(x[i]-xm);
   sigx2:=sx/n;  {pentru n mare}
   end;
 Function f( h, t:real ) : real;
   begin
   f := 2 * h / ( t * t );
   end;
 Function dfh ( h, t : real ) : real;
   begin
   dfh := f( h, t) / h;
```

```
   end;
 Function dft ( h, t : real ) : real;
   begin
   dft := -2 * f( h, t) / t;
   end;
 BEGIN
  write('introdu numarul de variabile independente, m=');
  readln(m);
  write('introdu numarul de seturi de masuratori, n='); readln(n);
  for i:= 1 to m do
    begin
    write('introdu valorile x[',i:2,']');
    for j:=1 to n do read(valx[j]);
    med_abstd(valx,n,xmediu[i], sigx2[i]);
    end;
  fmediu := f(xmediu[1], xmediu[2]);
  sig1fm2 := SQR( dfh(xmediu[1], xmediu[2]) );
  sig2fm2 := SQR( dft(xmediu[1], xmediu[2]) );
  sigfm2 := sig1fm2 * sigx2[1] + sig2fm2 * sigx[2];
  writeln('h=', xmediu[1]:10:2, '+/-', SQRT(sigx2[1]):10:2);
  writeln('t=', xmediu[2]:10:2, '+/-', SQRT(sigx2[2]):10:2);
  writeln('g=',fmediu:10:2, '+/-', SQRT(sigfm2):10:2);
  END.
```

8.3 Distribuţii

În cadrul unei măsurători se încearcă obţinerea valorii 'adevărate' a mărimii fizice. Ceea ce obţinem de obicei este valoarea medie care este o aproximaţie a valorii adevărate cu atât mai bună cu cât numărul de măsurători este mai mare. Dar valoarea pentru mărimea fizică măsurată deşi nu este unică nu este totuşi liberă a fi oricare. Ea are o distribuţie care de obicei poate fi aproximată cu o funcţie matematică simplă numită **funcţie densitate de probabilitate, p(x)**, sau **funcţie de distribuţie**. Aceasta înseamnă că probabilitatea de a obţine o valoare x în intervalul (x, x+dx) este dată de p(x)dx dacă variabila x variază continuu şi de $p(x_i)$ pentru x_i dacă x are valori discrete.

Funcţia de distribuţie satisface condiţia de normare la unitate:

$$\int_{-\infty}^{\infty} p(x)dx = 1 \tag{8.12}$$

pentru x continuu şi

$$\sum_{i=1}^{n} p(x_i) = 1$$

pentru x discret, n fiind numărul de valori posibile pentru x_i

Parametrii importanţi ce caracterizează o distribuţie sunt:
valoarea medie definită de

$$\mu = \int_{-\infty}^{\infty} xp(x)dx \tag{8.13}$$

sau

$$\mu = \sum_{i=1}^{n} x_i p(x_i)$$

şi abaterea pătratică medie (varianţa) definită de:

$$\sigma^2 = \int_{-\infty}^{\infty} (x-\mu)^2 p(x)dx =< x^2 > -\mu^2 \tag{8.14}$$

cu

$$< x^2 >= \int_{-\infty}^{\infty} x^2 p(x)dx$$

sau

$$\sigma^2 = \sum_{i=1}^{\infty} (x_i - \mu)^2 p(x_i)$$

8.3.1 Distribuţia normală

Distribuţia normală sau **distribuţia Gauss** este distribuţia cea mai des folosită în prelucrarea datelor experimentale. Forma sa este:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{8.15}$$

şi conţine 2 parametri independenţi, μ şi σ, (Fig.1).

Distribuţia Gauss are următoarele proprietăţi:

1. este normată la unitate

$$\int_{-\infty}^{\infty} p(x)dx = 1 \tag{8.16}$$

2. este simetrică faţă de dreapta $x = \mu$ în sensul că $p(x_1) = p(x_2)$ dacă $|\mu - x_1| = |x_2 - \mu|$

3. admite un singur maxim, la $x = \mu$, $p(\mu) = \frac{1}{\sigma\sqrt{2\pi}}$

4. prin integrare numerică se poate verifica faptul că probabilitatea ca o valoare măsurată x_i să aparţină intervalului $(\mu - \sigma, \mu + \sigma)$ este:

$$r(\mu - \sigma \leq x \leq \mu + \sigma) = \int_{\mu-\sigma}^{\mu+\sigma} p(x)dx = 0.693 \tag{8.17}$$

analog pentru intervalele de 2σ şi 3σ,

$$r(\mu - 2\sigma \leq x \leq \mu + 2\sigma) = 0,954$$

$$r(\mu - 3\sigma \leq x \leq \mu + \sigma) = 0.9973$$

De unde se vede că pentru o mărime, ale cărei valori sunt distribuite Gauss, valoarea ei apare cu o probabilitate de:

- 69,3 % în intervalul $(\mu - \sigma, \mu + \sigma)$
- 95,4 % în intervalul $(\mu - 2\sigma, \mu + 2\sigma)$
- 99,7 % în intervalul $(\mu - 3\sigma, \mu + 3\sigma)$

Concludem astfel că greşim puţin (cu mai puţin de 0,3%) dacă aruncă valorile ce nu aparţin intervalului $(\mu - 3\sigma, \mu + 3\sigma)$.

Iată un program care dă valorile numerice de mai sus pentru integralele pe intervalele considerate folosind un set de valori pentru care se calculează μ şi σ.

```
EX60.PAS
program dist_Gauss;
 uses crt;
 type fct = function(x : real) : real;
 const pi = 3.14159;
 var x : array[1..100] of real;
     s, xmediu, sigx, xmin, xmax : real;
     r : array[1..3] of real;
     i, n : integer;
 function simpson(a, b : real; n : integer; f : fct) : real;
  var s, h : real;  i : integer;
  begin
  h := (b - a)/2/n;  s := f(a) + f(b);
  for i := 1 to 2 * n do
      if i mod 2 = 0 then s := s + 2 * f(a+i*h)
        else s := s + 4 * f(a+i*h) ;
  simpson := s * h / 3;
  end; {end Simpson}
  {$F+}
  function Gauss ( x : real) : real;
    begin
    gauss := exp(-(x-xmediu)/sigx*(x-xmediu)/2/sigx);
    end; {end Gauss}
  {$F-}
  BEGIN  {programul principal}
  clrscr;
  write('introdu numarul de valori, n='); readln(n);
  writeln('introdu valorile x(i):');
  for i := 1 to n do read(x[i]);
  s := 0; for i := 1 to n do s := s + x[i]; xmediu := s/n;
  s := 0; for i := 1 to n do s := s + sqr(x[i]-xmediu);
  sigx := sqrt(s/n);
  writeln('xmediu=', xmediu, ' sigx=', sigx);
  for i := 1 to 3 do
    begin
    xmin := xmediu - i*sigx;
    xmax := xmediu + i*sigx;
    r[i] := simpson(xmin, xmax, 20, gauss)/sigx/sqrt(2*pi);
```

```
  end;
for i:= 1 to 3 do
begin
writeln('r(xmediu-',i:1,'*sigx<=x<=xmediu+',i:1,'*sigx)=',r[i]);
writeln('r este probabilitatea ca o valoare x sa cada in');
writeln('intervalul de 1, 2 sau 3 abateri standard in jurul');
writeln(' valorii medii');
end;
repeat until keypressed;
END.
```

8.3.2 Distribuţia Poisson

Distribuţia Poisson descrie apariţia evenimentelor rare, adică a evenimentelor a căror probabilitate de apariţie este foarte mică şi constantă. Această distribuţie este foarte importantă pentru măsurătorile de radiaţii unde se înregistrează dezintegrări puţine faţă de numărul foarte mare al nucleelor prezente.

Forma ei este:

$$p(k) = \frac{m^k}{k!} e^{-m} \tag{8.18}$$

unde k este valoarea obţinută pentru numărul de dezintegrări înregistrate în intervale egale de timp iar m este valoarea medie dată de:

$$m = \frac{\sum_{i=0}^{k_{max}} n_i k_i}{\sum_{i=0}^{k_{max}} n_i} \tag{8.19}$$

în care n_i este frecvenţa cu care s-a obţinut valoarea k_i.

$$N = \sum_{i=0}^{k_{max}} n_i \tag{8.20}$$

este numărul total de măsurători.

Proprietăţi ale distribuţiei Poisson:

1. este normată la unitate; deoarece $\sum_{k=0}^{\infty} \frac{m^k}{k!} = e^m$ rezultă:

$$\sum_{k=0}^{\infty} p(k) = 1 \tag{8.21}$$

2. valoarea medie este m.

$$\bar{k} = \sum_{k=0}^{\infty} kp(k) = \sum_{k=0}^{\infty} k\frac{m^k}{k!}e^{-m} =$$

$$= m\sum_{k=0}^{\infty} \frac{m^{k-1}}{(k-1)!}e^{-m} = m \tag{8.22}$$

3. abaterea standard este $\sqrt{m}$;

$$\overline{k^2} = \sum_{k=0}^{\infty} k^2 p(k) = \sum_{k=0}^{\infty} [k(k-1)+k]p(k) =$$

$$= m^2 \sum_{k=0}^{\infty} \frac{m^{k-2}}{(k-2)!}e^{-m} + m\sum_{k=0}^{\infty} \frac{m^{k-1}}{(k-1)!}e^{-m} = m^2 + m$$

rezultă că(vezi şi ecuaţia 8.4)

$$\sigma_k = \sqrt{\overline{k^2} - \bar{k}^2} = \sqrt{m} \tag{8.23}$$

4. probabilitatea de a obţine zero evenimente (număr de dezintegrări zero) este diferită de zero şi cu atât mai mare cu cât valoarea medie, m, a variabilei k este mai mică.

5. deoarece variabila k ia doar valori întregi, reprezentarea grafică a funcţiei p(k) este o **histogramă**. Figura 2 prezintă două histograme pentru diferite valori medii, m, întregi. Se vede că distribuţia este mai asimetrică pentru m mai mic şi că p(m-1) = p(m) dacă m este întreg:

$$p(m-1) = \frac{m^{m-1}}{(m-1)!}e^{-m} = \frac{m}{m}\frac{m^{m-1}}{(m-1)!}e^{-m} = \frac{m^m}{m!}e^{-m} = p(m) \tag{8.24}$$

6. distribuţia Poisson are un singur parametru independent. Acesta este valoarea medie m.

```
EX64.PAS
program distr_Poisson;
 uses crt, graph;
 {verifica distributia Poisson si reprezinta histograma;
 valori k intre 0 si 10, valori nl intre 0 si 100}
```

```
type vector=array[0..10] of integer;
const k:vector=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
     nk:vector=(35, 70, 98, 100, 70, 40, 30, 20, 10, 5, 2);
label unu, doi, trei;
var nkmax, kmax, snk, sknk:integer;
    cx:array[0..10] of string[2];
    kmediu, sigk, scy, sknk1:real;
    niuk:vector;
    i, scx, gd, gm, ix, iy1, iy2:integer;
    g:char;
BEGIN
sknk := 0; snk := 0; kmax := 10;
for i := 0 to kmax do
  begin
  sknk := sknk+nk[i]*k[i];
  snk := snk+nk[i];
  end;
kmediu := 1. * sknk/snk;  {1. pentru a transforma in real rezultatul}
sknk1 := 0; for i := 0 to kmax do
  sknk1 := sknk1+sqr(k[i]-kmediu)*nk[i];
sigk := sqrt(1.*sknk1/snk);
niuk[0] := ROUND(snk*exp(-kmediu));
for i := 1 to kmax do niuk[i] := ROUND(niuk[i-1]*kmediu/i);
writeln('   k    n(k)   niu(k)   '); writeln;
for i :=0 to kmax do writeln(k[i]:7, nk[i]:7, niuk[i]:7);
trei: write('doriti histograma?(d/n)');
readln(g); if (g='n') or (g='N') then goto unu;
           if (g='d') or (g='D') then goto doi;
           goto trei;
doi: {reprezintare histograma}
nkmax := nk[0]; if nkmax < niuk[0] then nkmax := niuk[0];
for i :=1 to kmax do
  begin
  if nkmax < nk[i] then nkmax := nk[i];
  if nkmax < niuk[i] then nkmax := niuk[i]
  end;
scx := 50; scy := 300./nkmax;
gd := detect; InitGraph(gd, gm, '\tp\bgi');
```

```
setlinestyle(0, 0, 1);
{ cu linie continua nk, cu linie intrerupta niuk}
line(10, 330, 10+scx*(kmax+1), 330); {linia de jos a histogramei}
line(10, 330, 10, 330-round(scy*nk[0]));
moveto(10, 330-round(scy*nk[0])); linerel(scx, 0);
for i := 1 to kmax do {histograma in nk}
  begin
  ix := 10_i*scx; iy1 := 330-round(scy*nk[i-1]));
  iy2 := 330-round(scy*nk[i]));
  line(ix, iy1, ix, iy2); moveto(ix, iy2); linerel(scx, 0);
  end;
moveto((kmax+1)*scx+10, iy2); lineto((kmax+1)*scx+10, 330);
setlinestyle(1,0,1); line (10,330,10,330-round(scy*niuk[0]));
moveto(10, 330-round(scy*niuk[0])); linerel(scx, 0);
for i := 1 to kmax do  {histograma in niuk}
  begin
  ix := 10+i*scx;
  iy1 := 330-round(scy*niuk[i-1]); iy2 := 330-round(scy*niuk[i]);
  line(ix, iy1, ix, iy2); moveto(ix, iy2); linerel(scx, 0);
  end;
moveto((kmax+1)*scx+10, iy2); lineto((kmax+1)*scx+10, 330);
for i := 0 to kmax do
  begin
  str(i,cx[i]); outtextxy(35+i*scx, 335, cx[i]);
  end;
repeat until keypressed;
closegraph;
unu: END.
```

8.4 Metoda celor mai mici pătrate pentru o dreaptă

Presupunem că măsurăm în punctele x[1]..x[n] valorile unei mărimi fizice y ce ştim că depinde liniar de x (y = a*x + b) şi obţinem valorile y[1]..y[n]. Din cauza erorilor (σ_i) în măsurarea lui y punctele nu se aşează perfect pe o dreaptă. Cum trecem totuşi dreapta printre punctele experimentale şi cum

obţinem parametrii ei?

Din considerente statistice se spune că dreapta care descrie cel mai bine punctele experimentale este cea care satisface condiţia:

$$S = \sum_{i=1}^{n} \frac{(y_i - a * x_i - b)^2}{\sigma_i^2} = min \tag{8.25}$$

Din $\frac{\partial S}{\partial a} = 0$ rezultă:

$$a \sum_{i=1}^{n} \frac{x_i^2}{\sigma_i^2} + b \sum_{i=1}^{n} \frac{x_i}{\sigma_i^2} = \sum_{i=1}^{n} \frac{x_i y_i}{\sigma_i^2} \tag{8.26}$$

şi din $\frac{\partial S}{\partial b} = 0$ rezultă:

$$a \sum_{i=1}^{n} \frac{x_i}{\sigma_i^2} + b \sum_{i=1}^{n} \frac{1}{\sigma_i^2} = \sum_{i=1}^{n} \frac{y_i}{\sigma_i^2}$$

un sistem de 2 ecuaţii cu 2 necunoscute, a şi b. Notând $\frac{1}{\sigma_i^2} = p_i$ = ponderea unei măsurători, sistemul se scrie:

$$a \sum_{i=1}^{n} p_i x_i^2 + b \sum_{i=1}^{n} p_i x_i = \sum_{i=1}^{n} p_i x_i y_i \tag{8.27}$$

$$a \sum_{i=1}^{n} p_i x_i + b \sum_{i=1}^{n} p_i = \sum_{i=1}^{n} p_i y_i$$

Soluţiile sunt:

$$a = \frac{\sum p_i \sum p_i x_i y_i - \sum p_i x_i \sum p_i y_i}{\sum p_i \sum p_i x_i^2 - (\sum p_i x_i)^2} \tag{8.28}$$

$$b = \frac{\sum p_i x_i^2 \sum p_i y_i - \sum p_i x_i \sum p_i x_i y_i}{\sum p_i \sum p_i x_i^2 - (\sum p_i x_i)^2}$$

Dacă $\sigma_1 = \sigma_2 = ... = \sigma_n$ atunci $p_1 = p_2 = ... = p_n$ şi expresiile pentru a şi b devin cele întâlnite adesea în fitul cu o dreaptă prin metoda celor mai mici pătrate:

$$a = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{n \sum x_i^2 - (\sum x_i)^2} \tag{8.29}$$

$$b = \frac{\sum x_i^2 \sum y_i - \sum x_i \sum x_i y_i}{n \sum x_i^2 - (\sum x_i)^2}$$

Pentru a găsi erorile cu care sunt determinaţi parametrii a şi b ai dreptei aplicăm teorema de propagare a erorilor (ec. 8.10):

$$a = a(x,y) \Rightarrow \sigma_a^2 = \left(\frac{\partial a}{\partial x}\right)^2 \sigma_x^2 + \left(\frac{\partial a}{\partial y}\right)^2 \sigma_y^2 \tag{8.30}$$

$$b = b(x,y) \Rightarrow \sigma_b^2 = \left(\frac{\partial b}{\partial x}\right)^2 \sigma_x^2 + \left(\frac{\partial b}{\partial y}\right)^2 \sigma_y^2$$

Cum am presupus că erori se fac doar în y rezultă că $\sigma_x = 0$ şi:

$$\sigma_a^2 = \sum_{i=1}^{n} \left(\frac{\partial a}{\partial y_i}\right)^2 \sigma_y^2 \tag{8.31}$$

$$\sigma_b^2 = \sum_{i=1}^{n} \left(\frac{\partial b}{\partial y_i}\right)^2 \sigma_y^2$$

cu $\sigma_{y_1} = \sigma_{y_2} = ... - \sigma_{y_n}$Cum:

$$\left(\frac{\partial a}{\partial y_i}\right)^2 = \left[\frac{nx_i - \sum x_i}{n\sum x_i^2 - (\sum x_i)^2}\right]^2 \tag{8.32}$$

rezultă:

$$\sigma_a = \sqrt{\frac{n}{n\sum x_i^2 - (\sum x_i)^2}}\,\sigma_y \tag{8.33}$$

La fel din:

$$\left(\frac{\partial b}{\partial y_i}\right)^2 = \left[\frac{\sum x_i^2 - x_i \sum x_i}{n\sum x_i^2 - (\sum x_i)^2}\right]^2 \tag{8.34}$$

rezultă

$$\sigma_b = \sqrt{\frac{\sum x_i^2}{n\sum x_i^2 - (\sum x_i)^2}}\,\sigma_y \tag{8.35}$$

EX61.PAS

```
program fit_dreapta;
 {metoda celor mai mici patrate}
 uses crt, graph;
 label unu, doi, trei, patru;
 var x, y, z : array[1..100] of real;
     sx, sy, sxy, sx2, sigy, num, a, b:real;
```

```
     siga, sigb, ymax, scx, scy:real;
     i, n, gd, gm, px, py.sigpy:integer;
     g:char;
BEGIN
write('introdu numar de puncte(x,y) masurate, n='); readln(n);
writeln('introdu valorile x[i], y[i], i=1,n, in');
writeln(' ordinea:x[1]<x[2]<...<x[n]');
writeln('cate o pereche x[i], y[i] pe linie');
for i := 1 to n do readln(x[i], y[i]);
write('introdu eroarea in y, sigy='); readln(sigy);
sx := 0; sy := 0; sxy := 0; sx2 := 0; {initializare pentru sume}
for i :=1 to n do
  begin
  sx := sx + x[i]; sy := sy + y[i]; sxy := sxy + x[i]*y[i];
  sx2 := sx2 + x[i}*x[i];
  end;
num := n*sx2 -sx*sx; {calcul numitor si verificare cu zero}
if num = 0 then goto unu;
{calcul parametrii dreptei si erorile lor}
a := (n*sxy - sx*sy)/num; b := (sx2*sy - sx*sxy)/num;
siga := SQRT(n/num)*sigy; sigb := SQRT(sx2/num)*sigy;
for i := 1 to n do z[i] := a*x[i} + b; {punctele de pe dreapta}
Writeln(' i   x        y        y=ax+b    '); {scrie rezultate}
for i := 1 to n do writeln(i:3, x[i]:8:2, y[i]:8:2, z[i]:8:2);
writeln('parametrii dreptei y=ax+b sunt:');
writeln('a= ',a:8:2, '+/-',siga:8:2);
writeln('b= ',b:8:2, '+/-',sigb:8:2);
trei: write('doriti reprezentare grafica?(d/n)'); readln(g);
if (g='n') or (g='N') then goto doi;
if (g='d') or (g='D') then goto patru else goto trei;
{reprezentare grafica a dreptei si punctelor experimentale}
patru: ymax := y[1]; if ymax < z[1] then ymax := z[1];
       for i :=2 to n do
          begin
          if ymax < y[i] then ymax := y[i];
          if ymax < z[i] then ymax := z[i];
          end;
       scx := 600. /x[n]; scy := 300./ymax; {factori de scalare}
```

```
       gd:=detect; InitGraph(gd,gm,'\tp\bgi');
       line(5,5,5,300); line(5, 300, 600, 300);
        {axele de coordonate, Oy si Ox}
for i := 1 to n do
  begin
  px := round(scx*x[i]); py := 300 - round(scy*y[i]);
  sigpy := round(scy*sigy);
  circle(px, py, sigpy div 3); putpixel(px, py, 15);
  end;
line(5, 300-round(scy*b), round(scx*x[n]), 300-round(scy*z[n]));
repeat until keypressed; closegraph; goto doi;
unu: writeln('calcul imposibil');
doi: END.
```

8.5 Mcmmp pentru o parabolă şi pentru un polinom de grad 3

Dacă punctele experimentale se aşteaptă să descrie o parabolă

$$y = a_0 + a_1 x + a_2 x^2$$

atunci plecând tot de la condiţia:

$$S = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 x_i)^2 = min$$

obţinem următorul sistem de 3 ecuaţii cu 3 necunoscute (a_0, a_1 şi a_2) din anularea derivatelor parţiale $\frac{\partial S}{\partial a_0}$, $\frac{\partial S}{\partial a_1}$ şi $\frac{\partial S}{\partial a_2}$:

$$a_0 n + a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} y_i$$

$$a_0 \sum_{i=1}^{n} x_i + a_1 \sum_{i=1}^{n} x_i^2 + a_2 \sum_{i=1}^{n} x_i^3 = \sum_{i=1}^{n} x_i y_i$$

$$a_0 \sum_{i=1}^{n} x_i^2 + a_1 \sum_{i=1}^{n} x_i^3 + a_2 \sum_{i=1}^{n} x_i^4 = \sum x_i^2 y_i$$

Dacă punctele experimentale se aşteaptă să descrie un polinom de grad 3

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$

atunci condiţia de minim pentru suma pătratelor distanţelor punctelor experimentale faţă de curbă:

$$S = \sum_{i=1}^{n} (y_i - a_0 - a_i x_i - a_2 x_i^2 - a_3 x_i^3)^2 = min$$

conduce prin aceeaşi procedură ca mai sus la un sistem de 4 ecuaţii algebrice liniare în necunoscutele a_0, a_1, a_2 şi a_3.

Sistemul de ecuaţii algebrice lineare se rezolvă prin metoda Gauss-Jordan folosind procedura
gaussj(var a:glnpbynp; n,np:integer; var b:glnpbymp; m,mp:integer)
din /ref.bib. Numerical Recipies/ Procedura cere ca programul principal ce o foloseşte să definească tipurile:

```
type glnpbynp=array[1..np,1..np] of real;
     glnpbymp=array[1..np,1..mp] of real;
     glnp=array[1..np] of integer;
```

a este matricea coeficienţilor şi are dimensiunea np x np
n este numărul de ecuaţii ale sistemului, $n < np$
b este matricea termenilor liberi, de dimensiune np x mp atunci când cu aceeaşi matrice **a** se rezolvă mai multe sisteme de ecuaţii ce diferă prin termenii liberi (în cazul nostru matricea b are o singură coloană).
m este numărul sistemelor de ecuaţii de rezolvat pentru aceeaşi matrice **a**.

Datele experimentale se scriu într-un fişier de tip text. Următoarele 2 programe citesc datele din fişierele respective, realizează fitul cu parabolă şi, respectiv, polinom de grad 3 şi reprezintă grafic curbele şi punctele experimentale.

```
EX62.PAS
program fit_parabola;
{metoda celor mai mici patrate}
uses graph;
const np=10; mp=10; n=3;
type glnpbynp=array[1..np,1..np] of real;
```

```
     glnpbymp=array[1..np,1..mp] of real;
     glnp=array[1..np] of integer;
label doi, trei, patru;
var x, y, z:array[1..100] of real;  f:text;  g:char;
sx, sx2, sx3, sx4, sy, sxy, sx2y, sigy, ap, bp, cp:real;
    a:glnpbynp; b:glnpbymp;
    ymax, ymin, scx, scy:real;
    i, npct, gd, gm, px, py, sigpy:integer;
{$i gaussj.inc}
BEGIN
assign(f,'parabola.dat'); reset(f);
readln(f, npct);
for i:=1 to npct do readln(f, x[i], y[i]);
sx:=0; sx2:=0; sx3:=0; sx4:=0;
sy:=0; sxy:=0; sx2y:=0; {initializare pentru sume}
for i:=1 to npct do
  begin
  sx:=sx+x[i]; sx2:=sx2+x[i]*x[i]; sx3:=sx3+sqr(x[i])*x[i];
  sx4:=sx4+sqr(x[i])*sqr(x[i]); sy:=sy+y[i]; sxy:=sxy+x[i]*y[i];
  sx2y:=sx2y+sqr(x[i])*y[i];
  end;
{formarea matricilor a si b pentru sistemul de ecuatii
algebrice liniare}
a[1,1]:=npct; a[1,2]:=sx; a[1,3]:=sx2; b[1,1]:=sy;
a[2,1]:=sx; a[2,2]:=sx2; a[2,3]:=sx3; b[2,1]:=sxy;
a[3,1]:=sx2; a[3,2]:=sx3; a[3,3]:=sx4; b[3,1]:=sx2y;
gaussj(a, n, np, b, 1, mp); {apelare procedura gaussj}
{calculul punctelor la x[i] de pe parabola cu parametrii obtinuti}
for i:=1 to npct do z[i]:=b[1,1]+b[2,1]*x[i]+b[3,1]*x[i]*x[i]
{scrie rezultatele}
writeln('  i     x      y      y=b[1,1]+b[2,1]*x+b[3,1}*x^2 ');
for i:=1 to npct do writeln(i:3,x[i]:8:2, y[i]:8:2, z[i]:8:2);
writeln('parametrii parabolei y=a+bx+cx^2 sunt:');
writeln('a=',b[1,1]:10:3,' b=',b[2,1]:10:3,' c=',b[3,1]:10:3);
readln;
trei: write('doriti reprezentare grafica?(d/n)'); readln(g);
if upcase(g)='N'then goto doi;
if upcase(g)='D'then goto patru else goto trei;
```

```
{reprezentare grafica a parabolei si punctelor experimentale}
patru: ymax:=y[1]; ymin:=y[1];
if ymax < z[1] then ymax:=z[1];
if ymin > z[1] then ymin:=z[1];
for i:=2 to npct do
  begin
  if ymax < y[i] then ymax:=y[i];
  if ymax < z[i] then ymax:=z[i];
  if ymin > y[i] then ymin:=y[i];
  if ymin > z[i] then ymin:=z[i];
  end;
scx:=300./(x[npct]-x[1]);
{ aici a fost nevoie de ordonarea crescatoare a lui x[i] ceruta in date}
scy:=200./(ymax-ymin); {scx si scy sunt factorii de scalare}
gd:=detect; InitGraph(gd, gm, '\tp\bgi');
line(5, 5, 5, 300); line(5, 300, 600, 300) {axele de coordonate}
outtextxy(620,300,'x'); outtextxy(10, 5, 'y');
sigpy:=round(scy*sigy);
for i:=1 to npct do
  begin
  px:=round(scx*x[i]); py:=300-round(scy*y[i]);
  circle(px, py, sigpy); putpixel(px, py, 15);
  px:=round(scx*(x[i]+0.2));
  py:=300-round(scy*(b[1,1]+b[2,1]*(x[i]+0.2)+b[3,1]*sqr(x[i]+0.2)))
  putpixel(px,py,5);
  end;
  readln;
closegraph;
doi: END.
```

Fişierul 'parabola.dat'arată, de exemplu, astfel:

```
9
0.  1.5
1.  4.5
2.  9.5
3.  13.5
```

```
4.  25.5
5.  36.5
6.  48.5
7.  64.5
8.  80.5
0.5
```

Parametrii parabolei $y = a + bx + cx^2$ ce fitează aceste puncte sunt:
a=-0.496 b=2.862 c=0.907

```
EX63.PAS
program mcmp_polinom;
uses graph;
const np=10; mp=10;
type glnpbynp=array[1..np,1..np] of real;
     glnpbymp=array[1..np,1..mp] of real;
     glnp=array[1..np] of integer;
var x, y, yt:array[1..30] of real;
    a:glnpbynp;  b:glnpbymp;
    i, j, n:byte; gd, gm:integer;
    px, py,pyt:integer;
    scx, scy, ymin, ymax:real;
    sx6, sx5, sx4, sx3, sx2, sx, sx3y, sx2y, sxy, sy:real;
    a0, a1, a2, a3:real f:text; numef:string;
{$i gauss.inc}
BEGIN
write('introdu nume fisier cu date de intrare:'); readln(numef);
assign(f,numef); reset(f); readln(f,n);
for i:=1 to n do readln(f, x[i], y[i]); close(f);
sx6:=0; sx5:=0; sx4:=0; sx3:=0; sx2:=0; sx:=0;
sx3y:=0; sx2y:=0; sxy:=0; sy:=0;
for i:=1 to n do
  begin
  sx6:=sx6+sqr(x[i])*sqr(x[i])*sqr(x[i]);
  sx5:=sx5+ sqr(x[i])*sqr(x[i])*x[i];
  sx4:=sx4+sqr(x[i])*sqr(x[i]);
  sx3:=sx3+sqr(x[i])*x[i]; sx2:=sx2+sqr(x[i]);
```

```
  sx:=sx+x[i]; sx3y:=sx3y+sqr(x[i])*x[i]*y[i];
  sx2y:=sx2y+sqr(x[i])*y[i];
  sxy:=sxy+x[i]*y[i]; sy:=sy+y[i];
  end;
a[1,1]:=n; a[1,2]:=sx; a[1,3]:=sx2; a[1,4]:=sx3;
 b[1,1]:=sy;
a[2,1]:=sx; a[2,2]:=sx2; a[2,3]:=sx3; a[2,4]:=sx4;
 b[2,1]:=sxy;
a[3,1]:=sx2; a[3,2]:=sx3; a[3,3]:=sx4; a[3,4]:=sx5;
 b[3,1]:=sx2y;
a[4,1]:=sx3; a[4,2]:=sx4; a[4,3]:=sx5; a[4,4]:=sx6;
 b[4,1]:=sx3y;
writeln('matricea a este:');
for i:=1 to 4 do
  begin
  for j:=1 to 4 do write(a[i, j]:17:1);
  writeln;
  end; readln;
writeln('matricea b este:');
for i:=1 to 4 do writeln(b[i,1]:15:5);
readln;
gaussj(a,4,np,b,1,mp);
writeln('parametrii poligonului y=a0+a1*x+a2*x^2+a3*x^3 sunt:');
a0:=b[1,1]; a1:=b[2,1]; a2:=b[3,1]; a3:=b[4,1];
writeln('a0=',a0,'   a1=',a1);
writeln('a2=',a2,'   a3=',a3); readln;
writeln(' i       x         y        yt');
for i:=1 to n do
 yt[i]:=a0+a1*x[i]+a2*x[i]*x[i]+a3*sqr(x[i])*x[i];
for i:=1 to n do
 writeln(i:3, x[i]:10:1, y[i]:10:5, yt[i]:10:5);
ymin:=1.E30; ymax:=-1.E30;
for i:=1 to n do
  begin
  if ymin > y[i] then ymin:=y[i];
  if ymin > yt[i} then ymin:=yt[i];
  if ymax < y[i] then ymax:=y[i];
  if ymax < yt[i] then ymax:=yt[i];
```

```
  end;
  writeln('ymin=', ymin, ' ymax=', ymax); READLN;
  scy:=300/(ymax-ymin); scx:=6.
  gd:=detect; InitGraph(gd, gm, '\tp\bgi'); setcolor(3);
  for i:=1 to n do
    begin
    px:=trunc(scx*x[i]+0.5);
    py:=400+trunc(scy*ymin+0.5)-trunc(scy*y[i]+0.5);
    pyt:=400+trunc(scy*ymin+0.5)-trunc(scy*yt[i]+0.5);
    circle(px+5, py, 2); putpixel(px+5, pyt, 2);
    end; readln;
pyt:=400+trunc(scy*ymin+0.5)-trunc(scy+0.5);
line(trunc(scx*x[1])+5, pyt, trunc(scx*x[n])+5, pyt);
line(trunc(scx*x[1])+5, 400+trunc(scy*ymin+0.5),
trunc(scx*x[1])+5, 400-trunc(scy*ymin+0.5)-trunc(scy*ymax+20.5));
readln;     closegraph;
END.
```

Bibliografie

[1] Valeriu Iorga, Ion Fătu, Programare în limbajul Pascal, IPB, Catedra de calculatoare

[2] Turbo Pascal 6.0 - Ghid de utilizare, Cluj Napoca, 1992

[3] Turbo Pascal 6.0 - programe, Cluj Napoca, 1994

[4] W.S.Dorn, D.D.McCracken, Metode numerice cu programe in fortran 4, Editura Tehnică, Bucureşti, 1976

[5] W.H.Press, B.P.Flannery, S.A.Teukolsky, W.T.Vetterling, Numerical Recipes; The Art of Scientific Computing, Cambridge University Press, 1986

[6] A.Constantinescu, A.Dafinei, Informatica pentru prelucrarea datelor de fizică, 2008 (in pagina Web a facultăţii de Fizică)