ANIŞOARA CONSTANTINESCU

NOŢIUNI DE PROGRAMARE ÎN TURBO PASCAL 6.0

EDITURA UNIVERSITĂŢII BUCUREŞTI
-1996-

ANIŞOARA CONSTANTINESCU

NOŢIUNI DE PROGRAMARE ÎN TURBO PASCAL 6.0

(NOTE DE CURS)

EDITURA UNIVERSITĂŢII BUCUREŞTI
– 1 9 9 6 –

Referenţi ştiinţifici: Conf. dr. C. I. NISTOR
Lector MIRCEA ŢUGULEA

Cartea este scrisă pentru studenţii anului I, Facultatea de Fizică, începători în studiul limbajului de programare Turbo Pascal. Atrag atenţia că ea, deşi necesară, nu este suficientă; la curs sunt discutate pe larg programele cu explicarea elementelor de limbaj şi a algoritmilor folosiţi. Programele, selectate din cărţile date în bibliografie sau create de autoare, există în fişierele EXn.PAS şi au fost verificate pe calculatoarele reţelei NOVELL. Pentru discuţiile utile asupra acestor programe şi pentru colaborarea în asigurarea unei bune funcţionări a reţelei NOVELL din catedră îi mulţumesc doamnei asistent universitar Anda Morait.

Conf. dr. A. CONSTANTINESCU
Catedra de Fizică Atomică şi Nucleară

ISBN-973-575-088-X

Cuprins pag.

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. TURBO PASCAL (V. 5.0) REFERENCE GUIDE

5. M.Oncescu: Fluctuaţii statistice în măsurarea radiaţiilor, IFA, 1958

6. LOUIS Lyons: Statistics for nuclear and particle physicists , Oxford, 1986

I. Introducere.

1. Editorul TURBO PASCAL.

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.

Editorul TP se activează tastând: turbo şi ENTER.

Apare pe ecran următoarea figură:

```
-------------------------------------------------
 FILE EDIT SEARCH RUN COMPILE DEBUG OPTIONS ...
-------------------------------------------------




-------------------------------------------------
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 dreapta sau la stânga pe una din componente. Apăsând tasta ENTER se deschide o fereastră cu un submeniu în care ne deplasăm cu tastele săgeţi sus sau 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).

In 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 MENIU activează rândul de sus.

Alte taste de interes sunt:

ALT-F5 determină ieşirea temporară din editor pentru a vedea rezultatele unui calcul; tastând ENTER se revine în editorul TP.

ALT-n determină trecerea în fereastra n (1<=n<=9)

ALT-F3 determină închiderea ferestrei curente.

CTRL-F9 determină lansarea în execuţie a unui program (cel din fereastra curentă).

Intre 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 într-o pagină (fereastră) şi ele sunt numerotate de la 1 la 9.

Incercaţi următoarele programe pentru a vă familiariza cu editorul TP.

EX1.PAS

```
program conversie;
  const  pi=3.14159;
  var  gfr,mfr,radian:real;
          {gfr=grade si fractiuni de grad,...}
       grade,minute,secunde:integer;
          {valori intregi}
  begin
    writeln('introdu unghiul in radiani');
    readln(radian);
    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')
  end.
```

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);
    writeln(' x=',xm,' +/- ',sigx)
  end.
```

EX3.PAS

```
program rezec4;
  label 10,20,30;
{de incercat si cu label unu,doi,trei; cu}
{inlocuirea corespunzatoare in textul programului}
  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+120*x-130
    end;
  begin
  20: writeln('introdu limitele interv. a,b ');
      read(a,b);  if a=b then goto 30;
 {asigura iesirea din executie}
      if f(a)*f(b)>0 then
        begin
    writeln('interv. nu contine nici-o rad.');
        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
  30:   end.
```

EX4.PAS

```
program fractie;
{simplificare fractie a/b prin impartire cu}
{c.m.m.d.c.}
  var a,b,x,y,z:integer;
  procedure cmmdc;
{calcul cmmdc cu algoritmul lui Euclid}
    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;   {cmmdc}
   begin   {programul principal}
writeln('introdu a si b ca nr. intregi,pozitive');
     read(a,b);
     writeln('fractie 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('fractie simplificata: ',a,'/',b)
       end
     else
      writeln(' nu se poate simplifica')
   end.
```

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: <= >= <> := .. (* *) (. .)

In TP, în scrierea programului se pot folosi litere mici sau mari.

Unităţile lexicale ale unui program în TP sunt:

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

i) identificatori şi cuvinte rezervate.

Identificatorii notează etichete, constante, tipuri, variabile, funcţii, proceduri, unituri, 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_** (subliniere).

Anumiţi identificatori, cunoscuţi ca **identificatori standard** sunt predeclaraţi. 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. Utilizatorul poate redefini funcţia lor.

Limbajul TP interzice 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..2147483647 (adică -2^{31} .. 2^{31} -1). Constantele zecimale pot fi scrise cu exponent; ele trebuie să fie în intervalul 2.9 E-38..1.7 E38. O constantă întreagă hexazecimală foloseşte semnul $ ca prefix. Ea trebuie să fie în domeniul $00000000..$FFFFFFFF (4 bytes / element).

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.

Intre 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*)

ATENTIE 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.

3. Structura programelor în TP

Diagrame de sintaxă.

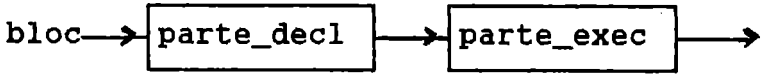
Reprezentarea sintaxei prin diagrame de sintaxă utilizează ca grafică:

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

Ca exemplu, structura unui program în TP se poate reprezenta astfel:

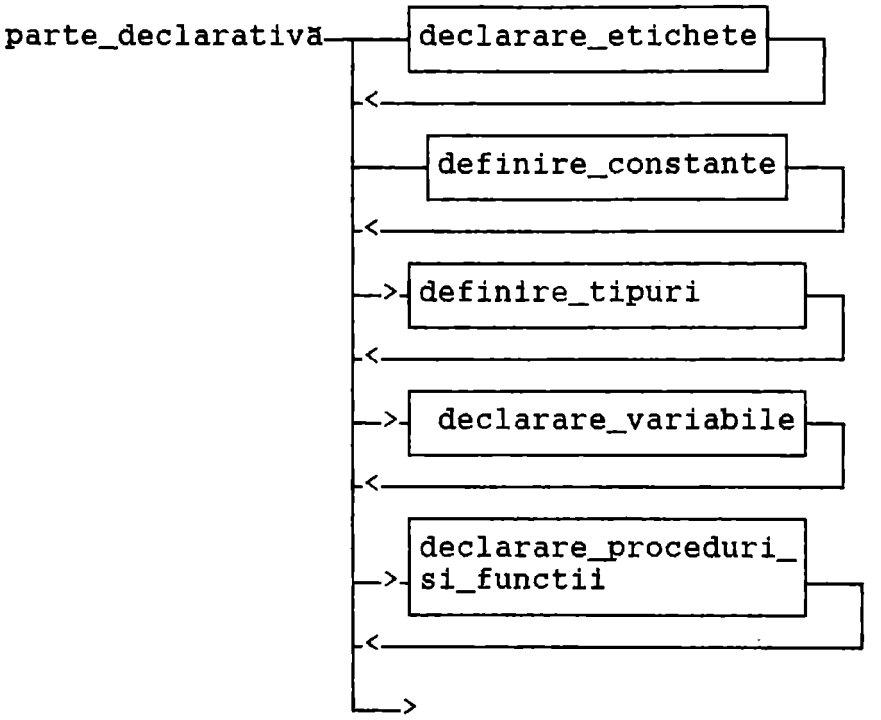
Un program este compus dintr-un bloc precedat de un antet.

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.



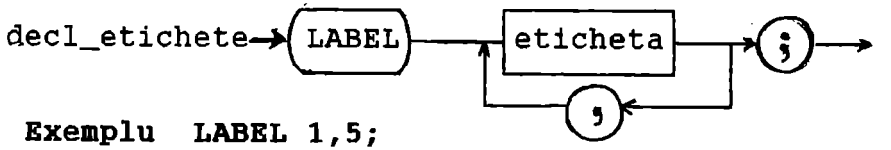
Obiectele (etichete, constante, tipuri, variabile, proceduri, funcţii) declarate în partea declarativă a blocului sunt locale acestui bloc.

Dau mai jos diagrama de sintaxă a părţii declarative a unui program în PASCAL.



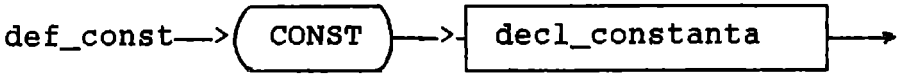
In TP declaraţiile sunt scrise în orice ordine (spre deosebire de PASCAL standard unde ordinea declaraţiilor este strictă şi precizată de diagrama de sintaxă de mai sus).

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

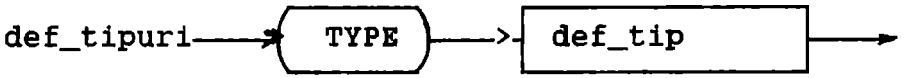


Exemplu **LABEL 1,5;**
LABEL unu,cinci; {singurele etichete permise în program sunt 1 , 5 , unu , cinci}

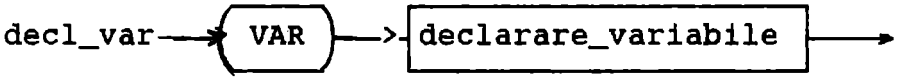
Fiecare etichetă trebuie să marcheze doar o singură instrucţiune.



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

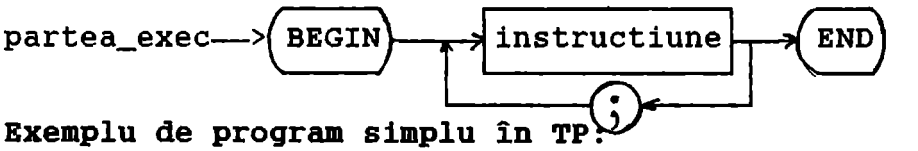



```
Exemplu TYPE vara=(iunie,iulie,august);
            litera='A'..'Z';
            matricep=ARRAY[1..n,1..n] OF REAL;
```




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

Diagrama de sintaxă a părţii executabile este:



Exemplu de program simplu în TP:

EX5.PAS

```
PROGRAM cerc;   {antetul programului}
{calculeaza aria cercului de diametru dat}
{partea declarativa}
CONST pi=3.14159;  {definirea constantei pi}
VAR diam,arie:REAL; {declararea variabilelor
diam si arie de tip real}
{partea executabila}
BEGIN
writeln('introdu  diametrul');  read(diam);
{citire date}
arie:=pi*SQR(diam)/4.0  {calcul arie}
write('arie=',arie);  {afisare rezultat}
END.
```

II. Tipuri de date (1)

1. Tipuri de date nestructurate predefinite: INTEGER, REAL, CHAR, BOOLEAN.

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

Un tip de date definește o mulțime finită de valori și o mulțime finită de operații asociate.

Fiecărei date i se asociază un tip unic.

In limbajul PASCAL se pot defini tipuri de date structurate pe baza unor tipuri nestructurate și a unor tipuri definite de programator.

Există 4 tipuri de date nestructurate reprezentate prin identificatorii de tip predefiniți: INTEGER, REAL, CHAR, BOOLEAN.

i) tipul **INTEGER** reprezintă o submulțime a mulțimii numerelor întregi dependentă de implementare. In TP există 5 tipuri întregi predefinite și 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

Operațiile 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 convertiți la tipul lor comun înainte de operație. 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.

Exemplu

= de operatori binari:
INTEGER (*) INTEGER -->INTEGER
cu (*) = {+,-,*,DIV,MOD}
= de funcţii:
ABS, SQR
= de operatori relaţionali
= <> >= <= > <

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

ii) **tipul REAL** reprezintă o submulţime finită a numerelor reale. In TP 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 bytes/variabilă, au domeniul de valori 2.9 E-38..1.7 E38 şi 11-12 cifre semnificative.

Operatori binari: REAL (*) REAL --> REAL
cu (*) = {+,-,*,/}
Funcţii standard: ABS, SQR, LN, EXP, SQRT, SIN, COS, ARCTAN
Funcţii de transfer: TRUNC --> pentru conversia în întreg cu trunchierea părţii fracţionare a argumentului.
ROUND --> pentru conversia în întreg cu rotunjirea părţii fracţionare a argumentului.

ROUND(x)= TRUNC(x+0.5) pentru x>=0
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.

iii) tipul **CHAR** reprezintă o mulţime finită şi ordonată de caractere din setul ASCII (American Standard Code for Information Interchange) extins. Incercaţi programul:

```
program caractere_ascii;
  var i:integer;
  begin
    for i:=1 to 6 do write(i:2,':',chr(i):2,';');
    for i:=14 to 255 do
        write(i:3,':',chr(i):2,';')
  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.

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.

iv) 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. 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.

i) 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,negru);
```

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).

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

ORD(alb)=0 ORD(rosu)=1..

PRED(rosu)=alb.. SUCC(alb)=rosu..

ii) 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    -"-       -"-        -"-    shortint
 'C'..'R'    -"-       -"-        -"-     char
rosu .. albastru    -"-        -"-      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 din permisiunea pentru capetele intervalului a expresiilor constante ca în următorul exemplu:

```
CONST x=50; y=10;
TYPE culoare=(rosu,verde,albastru);
     scala=(x-y)*2..(x+y)*2;
```

Sintaxa TP spune că dacă o definiţie de tip începe cu o paranteză,tipul este enumerare.

Ambiguitatea sintactică de mai sus se poate evita fie prin:

```
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 culoare=(rosu,verde,albastru);
     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.

```
Alte exemple:
TYPE vara=(iunie, iulie, august);
an=(ian, feb, mar, apr, mai, iun, iul, aug,
    sept, oct,nov, dec);
sem1=ian..iun;  {interval al tipului
                 enumerare an}
sem2=iul..dec;          -"-
secolXX=1900..1999  {interval al numerelor
                     intregi}
VAR  sezon:vara;
     annastere:secolXX;
```

sau se poate şi:

```
VAR  sezon:(iunie,iulie,august);
     {tip anonim, fara nume declarat in TYPE}
     annastere:1900..1999
```

III. Expresii.

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

Evaluarea se face de la stânga spre dreapta respectând nişte reguli de prioritate ale operatorilor. In TP există 4 nivele de prioritate şi anume:

Operatori	Prioritate
@ , not	prima (cea mai înalta)
* , / , 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:

a) un operand aflat între 2 operatori de priorităţi diferite este legat de operatorul de prioritate mai înaltă.

b) un operand aflat între operatori de priorităţi egale este legat de operatorul din stânga sa.

c) expresiile din paranteze se evaluează prioritar fiind tratate ca un singur operand.

La scrierea expresiilor se iau următoarele precauţii:

- să nu se omită operatorul * între 2 operanzi
- să nu apară 2 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	operatie	tip operanzi	tip rezultat
+	adunare	integer,real	integer,real
-	scadere	-"-	-"-
*	inmultire	-"-	-"-
/	impartire	-"-	real
div	impartire intreaga	integer	integer
mod	restul impartirii	integer	integer

OPERATORI LOGICI

Operator	operatie	tip operanzi	tip rezultat
not	negatie bit cu bit	integer	integer
and	si logic bit cu bit	-"-	-"-
or	sau logic -"-	-"-	-"-
xor	sau exclusiv -"-	-"-	-"-
shl	deplasare stanga	-"-	-"-
shr	deplasare dreapta	-"-	-"-

= Dacă operandul operatorului NOT este de un tip întreg, rezultatul este de același 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.

= Operațiile i shl j și i shr j deplasează valoarea lui i la stânga și,respectiv, la dreapta cu j biți. Tipul rezultatului este același cu tipul lui i.

OPERATORI BOOLEENI.

Operator	operație	tip operanzi	tip rezultat
not	negație	boolean	boolean
and	și 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	-"-	-"-
<	mai mic	tipuri simple compatibile, string	-"-
>	mai mare	-"-	-"-
<=	mai mic sau egal	-"-	-"-
>=	mai mare sau egal	-"-	-"-
<=	subset al	tipuri set compatibile	-"-
>=	superset al	-"-	-"-
in	membru al	operand stânga: orice tip ordinal,t operand dreapta: setul a cărui baza este compatibila cu t	-"-

= Compararea tipurilor simple: ele trebuie să fie compatibile; totuşi dacă un operand este de tip real, celălat poate fi de un tip întreg.

= Compararea şirurilor: se face conform cu ordonarea setului de caractere ASCII; oricare 2 valori string pot fi comparate deoarece toate valorile string sunt compatibile. O valoare tip CHAR este compatibilă cu o valoare STRING; când ele sunt comparate valoarea tip CHAR este tratată ca o valoare tip STRING de lungime 1.

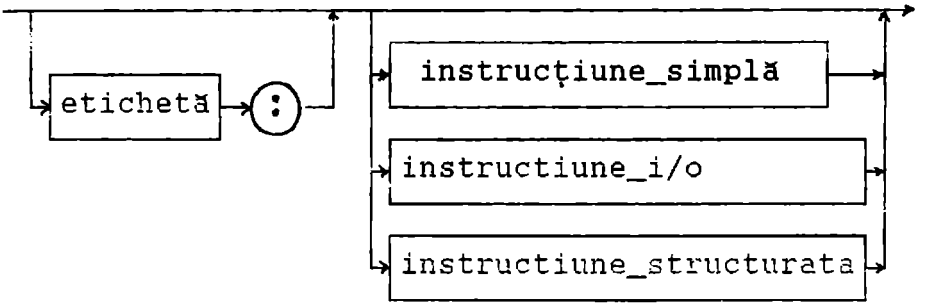
```
     FUNCŢII STANDARD:       f:A-->B
= funcţii matematice: sin cos arctan exp ln sqrt
                      A={real,integer}, B={real}
                       abs sqr
                      A={real,integer}, B=A
= funcţii de transfer: trunc round
                      A={real}, B={integer}
= funcţii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

EX6.PAS

```
program operatii;
  var i,j,k,l,m,n,o,sl,sr:integer;
  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.
```

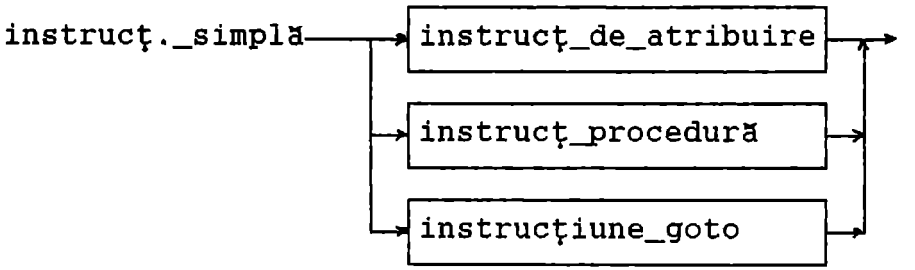
IV. Instrucţiuni.

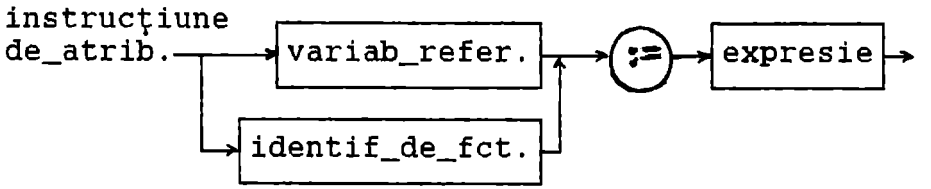
Diagrama de sintaxă pentru o instrucţiune este următoarea:



1.Instrucţinui simple.

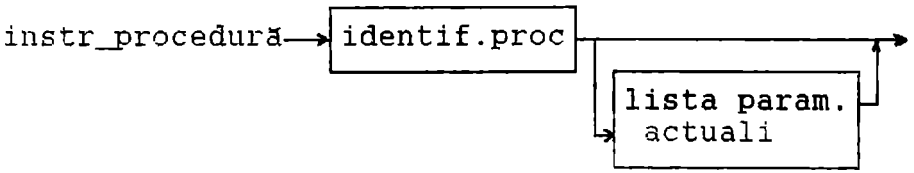
Diagramele de sintaxă pentru instrucţiunile simple sunt următoarele:





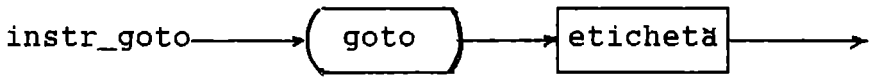
Exemple: `radian:=1.5; gfr:=radian*180.0/pi; grade:=trunc(gfr);` din EX1.PAS, sau `f:=x*x*x*x-9*x*x*x+120.*x-130.` din EX3.PAS

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.



Exemplu: cmmdc din EX4.PAS.

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



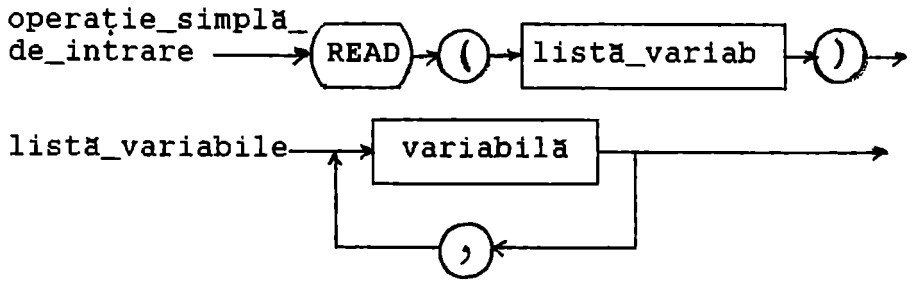
Exemple: **goto 10; goto unu;**

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.

2. Instrucţiunile de transfer de date: READ, READLN, WRITE, WRITELN.

Intr-un program în PASCAL se consideră predeclarate fişierul standard de intrare (INPUT) şi fişierul standard de ieşire (OUTPUT).

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

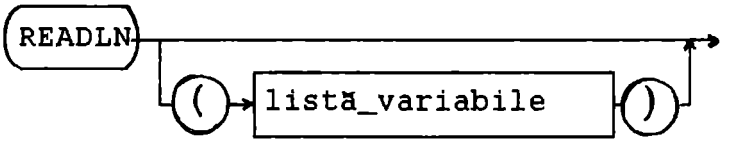


Variabilele din listă pot aparţine numai tipurilor simple INTEGER, REAL, CHAR.

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 către 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.

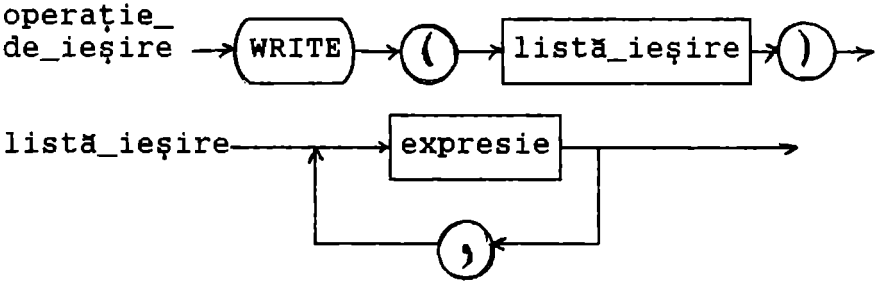
O formă deosebită a operaţiei de citire are sintaxa:



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.

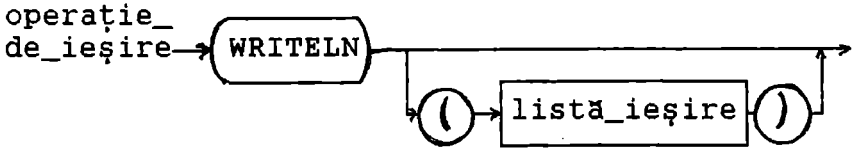
Procedura READLN fără listă_de_variabile nu realizează un transfer de informaţie ci doar o poziţionare la începutul liniei următoare.

ii) Operaţia de ieşire se realizează prin apelul procedurii WRITE conform diagramei de sintaxă:



Spre deosebire de lista de intrare (listă de variabile) în lista de ieşire pot apare expresii.

Ca şi la intrare datele de ieşire sunt transmise în flux continuu. Ele pot fi structurate pe linii folosind procedura WRITELN:



Forma WRITELN realizează doar trecerea la linia următoare în timp ce WRITELN(listă_ieșire) scrie în linia curentă valorile expresiilor din listă_ieșire după care face trecerea la linia următoare.

= o valoare booleană apare la ieșire sub forma șirului de caractere TRUE sau FALSE.

= o valoare întreagă apare ca un șir de cifre precedat, eventual, de semnul - .

= o valoare reală apare la ieșire în virgulă mobilă normalizată conținând o mantisă și un exponent.

Operația de ieșire permite specificarea numărului de poziții în care se transferă valoarea de ieșire.

= în cazul în care valoarea de ieșire este de tip INTEGER, CHAR sau BOOLEAN se poate specifica **un singur parametru** și anume lungimea totală a zonei ca o expresie întreagă. În acest caz valoarea respectivă va fi plasată în linia de ieșire aliniată la dreapta în zona de lungime specificată. Dacă lungimea zonei < lungimea valorii de tipărit, zona va fi extinsă pentru a afișa întreaga valoare.

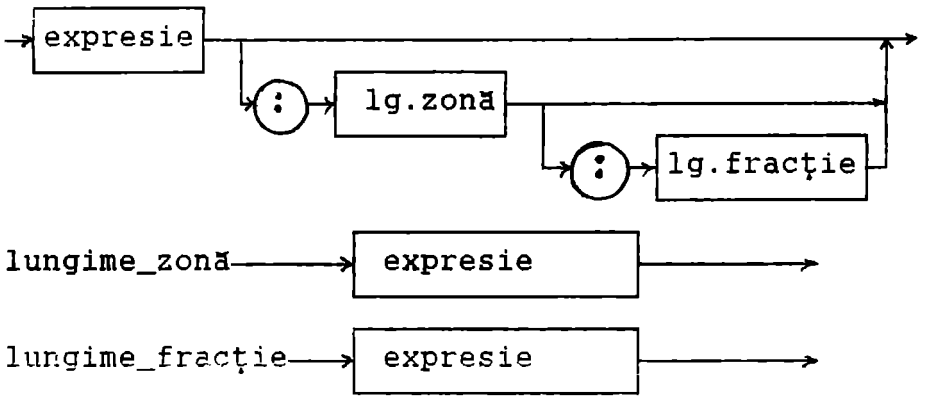
= când valoarea de ieșire este de tip REAL sunt necesari **doi parametri**: lungimea totală a zonei și lungimea fracției. Dacă sunt specificați ambii parametri, valoarea reală e scrisă fără exponent, cu punct zecimal și semn în zona rezervată, aliniată la dreapta, având partea fracționară rotunjită la numărul de cifre precizat ca lungime a fracției.

Exemplu: valoarea din memorie x=-15.864 va apare cu WRITE(x:6:1) ca -15.9

Dacă se specifică doar lungimea zonei, valoarea reală va fi afișată normalizat având mantisa rotunjită să încapă în lungimea totală a zonei minus 6.

Exemplu: WRITE(x:8) afișează, pentru aceeași valoare x de mai sus, -1.6 E+01

Diagrama de sintaxă:



EX7.PAS

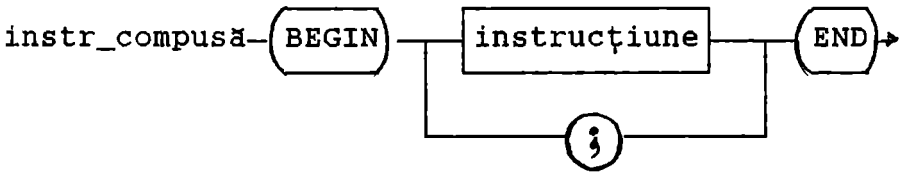
```
Program racheta;
  var ore,min,sec,tzbor:0..maxint;
  begin
  writeln('dati ora lansarii rachetei in
ore/minute/secunde');
  readln(ore,min,sec);
  writeln('lansare ',ore:2,'/',min:2,'/',sec:2);
  writeln('dati in secunde durata de zbor a
rachetei');
  readln(tzbor);
  writeln('durata de zbor: ',tzbor:6,' sec');
  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, se introduc fie ore ENTER minute ENTER secunde ENTER fie ore minute secunde (valorile separate prin blanc).

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).

i. **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:



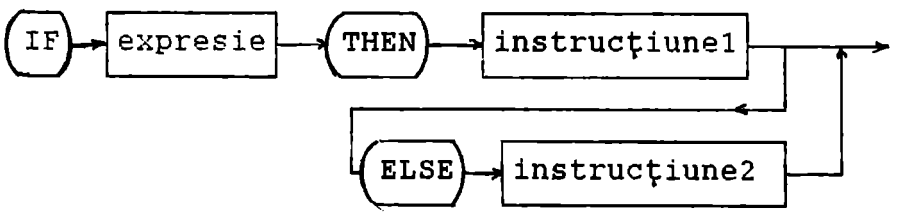
Instrucţiunea compusă este tratată sintactic ca o singură instrucţiune. Partea executabilă a unui program poate fi astfel considerată ca o singură instrucţiune compusă.

Exemple:
partea executabilă din EX1.PAS, EX2.PAS...

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

iia) 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:
TRUE - se execută instrucţiune1
FALSE şi este prezent ELSE - se execută instrucţiune2
FALSE şi nu este prezent ELSE - 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.

NOTA: 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
       instrucţiune1
    else
       instrucţiune2
   end
```

sau, în cazul următor necesitând pentru interpretare corectă paranteza begin..end,:

```
if expresie1 then
   begin
    if expresie2 then instrucţiune1
   end
else
  instrucţiune2
```

In general un ELSE este asociat cu cel mai apropiat IF neasociat deja cu un alt ELSE.

Dacă instrucţiune1 sau/şi instrucţiune2 sunt formate din mai multe instrucţiuni, ele se reprezintă ca instrucţiuni compuse.

Exemple:

1. Se consideră trei valori reale 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 'numerele formează un triunghi' se exprimă prin valoarea expresiei booleene:

(c < a + b) AND (c > ABS(a - b))

expresie numită inegalitatea triunghiurilor.

EX8.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    {citeste date a<=b<=c}
    writeln(' introdu 3 valori reale, a<=b<=c');
    readln(a,b,c);
    writeln(a:6:2,' ',b:6:2,' ',c:6:2);
    if ( c< a+b ) AND  (c > ABS(a-b) ) then
      begin
     write(' numerele date formeaza un triunghi');
         echl:= (a=b) AND (a=c);
         isos:= (a=b) OR (c=b) OR (a=c);
         drept:= c*c=a*a+b*b;
         if echl then writeln(' echilateral')
         else
           begin
            if isos then writeln(' isoscel');
            if drept then writeln(' dreptunghi')
       else if not isos then writeln(' oarecare')
           end
      end
else
writeln('numerele date nu formeaza un triunghi');
    readln
  END.
```

2. Fiind citit un caracter să se scrie dacă este vocală sau consoană.

EX9.PAS

```
program vocale_consoane;
  LABEL 1;
  VAR c:CHAR;
  BEGIN
  1:  writeln(' 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 consoana');
    GOTO 1
  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 numărul introdus este 5413, numărul cerut este 3145.

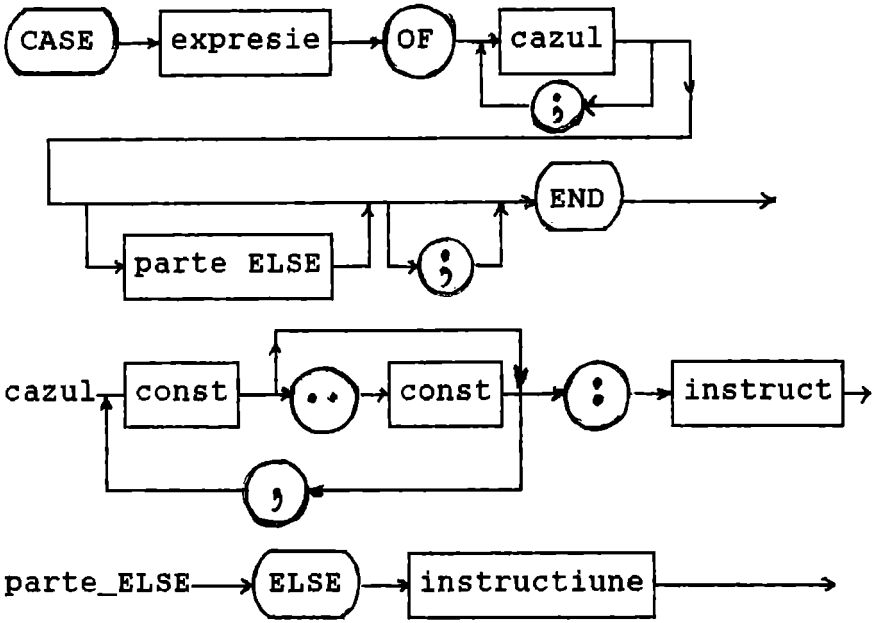
EX10.PAS

```
PROGRAM calcul_inversare_cifre;
  LABEL  1;
  VAR   m:integer;
  BEGIN
  writeln(' 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.
```

Pentru seminar: să se scrie programul ce rezolvă o ecuaţie de gradul 2 cu coeficienţi reali.

iib) instrucţiunea CASE

Diagrama de sintaxă a instrucţiunii CASE este următoarea:



Instrucţiunea 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) numite **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.

Selectorul, de tip ordinal, are 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 selectorului sau de un domeniu CASE ce conţine valoarea selectorului. Dacă nu există o astfel de constantă CASE a domeniului CASE şi este prezentă partea_ELSE se execută instrucţiunea ce urmează pe ELSE. Dacă nu există partea_ELSE nu se execută 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 caracter special.

EX11.PAS

```
PROGRAM testcase1;
  VAR c:CHAR;
  BEGIN
 writeln(' 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
   instrucţiunea IF cu ELSE}
else writeln (' semn special');  end;  {end case}
     readln;
   END.
```

2. Fiind citit un număr să se scrie:
   dacă e format dintr-o singură cifră, dacă e par sau impar
   dacă este între 10 și 20 sau între 30 și 40
   dacă este între 21 și 29
   dacă este >40 sau negativ

EX12.PAS

```
program testcase2;
  var   i:integer;
  begin
    writeln(' introdu un numar intreg');
    readln(i);
    case i of
    0,2,4,6,8 :  writeln(' cifra para');
    1,3,5,7,9 :  writeln(' cifra impara');
10..20,30..40 :
       writeln('nr. intre 10..20 sau 30..40');
    21..29    : writeln(' nr. intre 21..29');
    else writeln(' nr.> 40 sau negativ');
    end
  end.
```

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

EX13.PAS

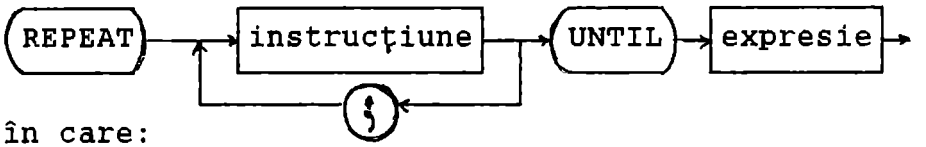
```
program maine;
{calculeaza data zilei de maine fiind data data
 zilei de azi}
  var an:1900..2000;
      luna:1..12;
      zi:1..31;
      ultima_zi:28..31;
      bisect:boolean;
  beginwriteln(' introdu in cifre zi,luna,an,data
                  zilei de azi');
    readln(zi,luna,an);
    writeln(' azi:',zi,'/',luna,'/',an);
    {determinarea ultimei zile a lunii}
    bisect:=(an mod 4 =0) and (an mod 100 <>0) or
            (an mod 400 =0);
    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}
  {actualizarea datei}
  if zi< ultima_zi then zi:=zi+1
  else if zi> ultima_zi
         then writeln(' data incorecta')
       else
         begin  {luna urmatoare}
          zi:=1;
          if luna=12 then
            begin
              luna:=1;
              an:=an+1  {anul urmator}
            end
          else luna:=luna+1;
        end;  {luna urmatoare}
   writeln(' maine:',zi,'/',luna,'/',an)
 END.
```

Pentru seminar: să se scrie EX9.PAS cu instrucţiune CASE.

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

iiia) instrucţiunea REPEAT

Diagrama de sintaxă a instrucţiunii REPEAT este următoarea:



în care:

expresie - controlează repetarea execuţiei succesiunii de instrucţiuni din cadrul instrucţiunii REPEAT.
- trebuie să producă 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.

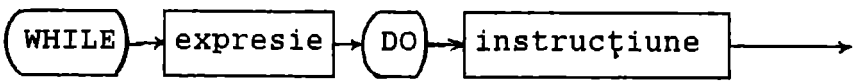
Exemple:

```
1) REPEAT  K:=I MOD J; I:=J; J:=K;
   UNTIL J=0;

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

iiib) instrucţiunea WHILE

Diagrama de sintaxă a acestei instrucţiuni 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 **instrucţiunea** conţinută să fie executată; **instrucţiune** este executată repetat cât timp **expresie** dă valoarea TRUE; dacă **expresie** dă FALSE la început, **instrucţiune** nu se execută nici o dată.

Exemple:

```
     1) nr:=0; READ(c);
        WHILE c=' ' DO
{aceasta secventa de program contorizeaza}
{in variabila nr numarul de blancuri}
{alaturate dintr-un sir de caractere}
        begin
           nr:=nr+1;
           READ(c);
         END;

     2) x:=0.5*a;
{aceasta secventa de program calculeaza}
{cu precizia relativa 1.e-07}
        WHILE ABS(x-a/x) > 1 e-07 DO
        x:=0.5*(x+a/x);
{sqrt(a) prin aplicarea formulei lui}
{Heron}
```

Exemple de programe cu 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. Exemplu: n=121, 323, 3113,..

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

EX14.PAS

```
program palindrom;
```

```
{stabileste daca un numar este 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=nr.cifre din 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 cea mai semnificativa cifra}
        q:=n mod 10;
 { separa cea mai putin semnificativa cifra}
        n:=n mod mare;
 {suprima cea mai semnificativa cifra}
        n:=n div 10;
 {suprima cea mai putin semnificativa cifra}
        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

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

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

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

pornind cu

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

Calculul se opreşte când

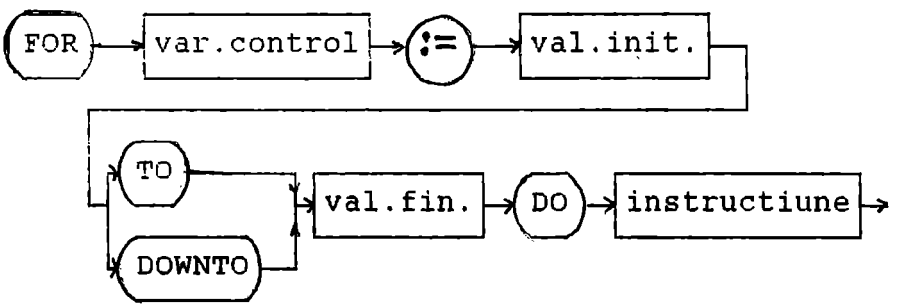
$$|u_n - v_n|<\epsilon \quad cu\ \epsilon=1.0e-05$$

EX15.PAS

```
program val_integrala_sir;
  var   a,b,u,u1,v: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,' iter');
  end.
```

iiic) instrucţiunea FOR.

Diagrama de sintaxă a instrucţiunii FOR este următoarea:



în care
var.control este un identificator de variabilă iar **val.init** (valoarea iniţială) şi **val.fin** (valoarea finală) sunt câte o expresie.

Instrucţiunea FOR face ca **instructiune** (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 val.init. şi val.fin. trebuie să fie de un tip compatibil din punct de vedere atributiv[1)] 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.

Instrucţiune 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 > valoarea finală, **instructiune** nu se execută. Când instrucţiunea FOR foloseşte DOWNTO, valoarea variabilei de control **scade cu unu** la fiecare repetare. Dacă valoarea iniţială < valoarea finală, **instrucţiune nu se execută.**

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

După execuţia instrucţiunii FOR valoarea variabilei de control este nedefinită cu excepţia cazului când execuţia instrucţiunii FOR a fost întreruptă printr-un goto din instrucţiunea FOR.

Exemple de programe cu instrucţiunea FOR:

1. programul val_medie din EX2.PAS

2. Fiind citit un număr întreg n să se calculeze suma

$$s = 1 + 1/2 + 1/3 + \ldots + 1/n$$

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.
```

3. Program ce scrie codul și caracterele ASCII

EX17.PAS

```
program caractere_ascii;
  var i:integer;
  begin
    for i:=1 to 6 do write(i:2,':',chr(i):2,';');
    for i:=14 to 255 do
 write(i:3,':',chr(i):2,';')
  end.
```

****

1) tipuri compatibile din punct de vedere atributiv: T1:= T2 este permis dacă:

- T1 și T2 sunt tipuri identice
- T1 și T2 sunt tipuri ordinale[2)] compatibile și valorile tipului T2 se află în domeniul valorilor posibile ale lui T1
- T1 și T2 sunt reale, valorile tipului T2 se află în domeniul valorilor posibile ale lui T1
- T1 este real, T2 este întreg
- T1 și T2 de tip string
- ...

2) tipuri compatibile:

Compatibilitatea a două tipuri se impune în cazul expresiilor și operațiilor de relație. Ea este premiza compatibilității din punct de vedere atributiv.

Două tipuri sunt compatibile atunci când e adevărată cel puțin una din următoarele afirmații:

- cele 2 tipuri sunt identice
- cele 2 tipuri sunt reale
- cele 2 tipuri sunt întregi
- un tip este un interval al celuilalt tip
- ambele tipuri sunt intervale ale aceluiași tip de bază
- ...

Alte exemple:

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

EX18.PAS

```
program conv_hex_zec;
  {calculeaza valoarea zecimala a unui sir de
   caractere hexazecimale, nr. intreg}
  var  c:char; d:integer; val:0..15;
begin
    writeln(' introdu nr. in forma hexa, 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 instr.compusa din while}
    writeln;  {trece la linia urmatoare}
    writeln(d)
        {scrie nr. in reprezentarea zecimala}
  end.
```

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

EX19.PAS

```
PROGRAM nr_PITAGO;
Uses crt;
var i,j,n:byte; v,z:real;
 begin
 clrscr;
 i:=1;  j:=1; write('introdu n<40,=');readln(n);
 while i<=n do
   begin
   j:=i;
{pentru a elimina nr. duble, de ex. 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.

EX20.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 nr 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 afișează numerele prime cuprinse între A și B precum și numărul lor

EX21.PAS

```
program numere_prime;
Uses crt;
 label salt;
 const nr:word=0;
 var a,b,i,j:word;
 begin
 clrscr;
 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,' num. prime intre ',a,' si '
        ,b);
    end.
```

5. Un program care calculează mediile aritmetică și geometrică pentru un șir de N numere citite de la tastatură.

EX22.PAS

```
program medii;
Uses crt;
 var  nr,suma,x:integer;
     contor,produs:word;
 begin
   clrscr; contor:=1;
  write(' cate numere avem?'); readln(nr);
    suma:=0; produs:=1;
    while contor <=nr do
      begin
       write('x[',contor,']= '); readln(x);
       inc(suma,x);produs:=produs*x;inc(contor);
      end;
```

```
    writeln;
writeln(' media aritmatica=',suma/nr:5:2);
writeln(' media geometrica=',
          exp(ln(produs)/nr):5:2)
 end.
```

6. Un program care calculează valorile expresiei:

$$E=(2*a+3*b+c)/(d+1)$$

pentru valori a,b,c,d 1..2

EX23.PAS

```
program expresie;
 Uses crt;
 var a,b,c,d:byte;
 begin
  clrscr;
 writeln('val.expresiei pt. a,b,c,d in int.1..2');
 writeln;
  a:=0;
  repeat
    inc(a);b:=0;
      repeat
        inc(b); c:=0;
          repeat
            inc(c);d:=0;
            repeat
              inc(d);
write('E(',a,',',b,',',c,',',d,')= ');
writeln((2*a+3*b+c)/(d+1):3:2);
            until d=2
          until c=2
      until b=2
  until a=2;
  writeln;  readln;
end.
```

V. Tipuri de date (2).

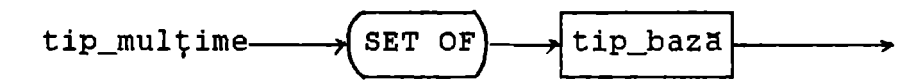
Tipuri structurate de date: set, array, string,record.

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

1. Tipul set (mulţime)

Tipul set se defineşte în raport cu un **tip de bază**, care 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.

Diagrama de sintaxă a definirii acestui tip este:



Exemple:

```
1)  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]

NOTĂ: = dacă tipul de bază are n valori, tipul mulţime (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 pentru tipul întreg doar BYTE şi nu şi celelalte.

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

Valorile variabilelor stud1,stud2 pot fi modificate prin atribuiri de forma:

```
stud1:=[analiza,fizica];
stud2:=[ ];
```

NOTĂ: 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 elementul cât şi limitele de interval pot fi expresii. Dacă sup < inf nu se generează nici-un element.

Operaţiile cu valori tip mulţime (set) sunt:

+ **reuniune** o valoare de tip ordinal **c** este în **a + b** dacă **c** este în **a** sau în **b**.
- **diferenţă** o valoare de tip ordinal **c** este în **a - b** dacă ea este în **a** şi nu este în **b**.
* **intersecţie** o valoare de tip ordinal **c** este în **a * b** dacă ea este în **a** şi în **b**.

Relaţiile referitoare la mulţimi:

dacă **a** şi **b** sunt operanzi de tip set,

a = b este adevărată numai dacă **a** şi **b** conţin exact aceleaşi elemente; altfel **a <> b**.

a <= b este adevărată dacă fiecare element al lui **a** este de asemeni un element al lui **b**.

a >= b este adevărată dacă fiecare element al lui **b** este de asemeni un element al lui **a**.

x in a este adevărată dacă **x** aparţine lui **a**. Dacă **x** este de tipul ordinal **t**, **a** este de tip mulţime cu tipul de bază compatibil cu **t**.

Exemplu.

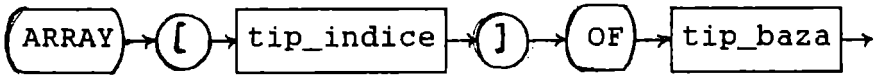
Fiind citite elementele a două mulţimi m1 şi m2 (ce au tipul de bază BYTE) să se scrie elementele mulţimilor m1 + m2, m1 * m2, m1 - m2.

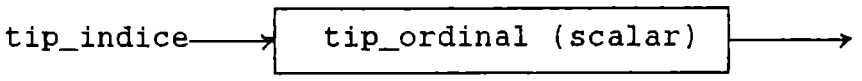
EX24.PAS

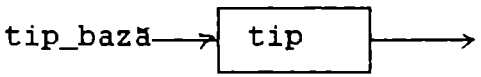
```
program setop;
Uses crt;
  type mult=set of byte;
            {tipul de baza este byte}
  var  m1,m2,int,un,dif:mult;
  nrm1,nrm2,x,i:byte;
  BEGIN
  clrscr;
  write('introdu nr.de elem.din mult.m1,nrm1= ');
  readln(nrm1); m1:=[];
  for i:=1 to nrm1 do
   begin
    write('elementul tip BYTE ',i,' este:');
    readln(x); m1:=m1+[x];
   end;
  write('introdu nr.de elem.din mult.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 elem. multimilor int., reun., dif.}
   int:=m1*m2; un:=m1+m2; dif:=m1-m2;
   {scrierea rezultatelor}
   write(' Operandul stang este: ');write('[');
    for i:=0 to 255 do
      if i in m1 then write(i,',');writeln(']');
   write(' Operandul din dreapta este: ');
   write('[');
    for i:=0 to 255 do
      if i in m2 then write(i,',');writeln(']');
   write(' M1 ',#239,' M2 = ');write('[');
   for i:=0 to 255 do
     if i in int then write(i,','); writeln(']');
   write(' M1 U M2 = ');write('[');
    for i:=0 to 255 do
      if i in un then write(i,','); writeln(']');
   write(' M1 - M2 = '); write('[');
    for i:=0 to 255 do
      if i in dif then write(i,','); writeln(']');
 END.
```

2. Tipul array (tablou).

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:







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 exceptând longint şi subdomenii 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 (eventual printr-o expresie al cărei rezultat trebuie să se afle în domeniul definit de tip_indice).

Exemplu: variabila x din EX2.PAS.

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 multidimensional. 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 suplimentând la identificatorul tabloului valorile indicilor în paranteze drepte. De exemplu **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]**, deoarece a[I] reprezintă un element de tip ARRAY[1..20] OF REAL, adică linia I a matricii. Linia L2 a matricii poate fi copiată în linia L1 simplu prin a[L1]:=a[L2]. Acest exemplu poate fi dat şi printr-un tablou cu 2 dimensiuni:
VAR a:ARRAY[1..10,1..20] OF REAL; în care caz o componentă este accesată cu a[I,J].

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[1,0,0]=4
tabel[0,0,1]=1   tabel[1,0,1]=5
tabel[0,1,0]=2   tabel[1,1,0]=6
tabel[0,1,1]=3   tabel[1,1,1]=7
```

sau

```
const cifra:array[0..9] of
            char=('0', '1', '2', '3', '4', '5',
                  '6', '7', '8', '9');
```

sau, mai convenabil exprimat,

```
    const cifra:array[0..9] of char='0123456789';
```

dă pentru **cifra** inițializarea la:

```
cifra[0]='0'   cifra[1]='1'   cifra[2]='2'
cifra[3]='3'   cifra[4]='4'   cifra[5]='5'
cifra[6]='6'   cifra[7]='7'   cifra[8]='8'
cifra[9]='9'
```

Exemple

1. Fiind dată o matrice să se scrie programul ce calculează puterea n (n este un număr natural) a matricii.

EX25.PAS

```
program matrice;
 {ridicarea unei matrice la o putere naturala}
  const n=3;   {dimensiune fixata pentru matrice}
  type matricep=array[1..n,1..n] of real;
  var a,b,c:matricep;
          s:real;
      i,j,k:1..n;
      l,p:byte;
  BEGIN
  writeln(' introdu puterea ');
  readln(p);
  {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;
   writeln;
  {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:=0.0;
     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;  {trecere la randul urmator}
   end
END.
```

2. Program ce elimină linia L şi coloana C dintr-o matrice M*N.

EX26.PAS

```
program eliminare;
uses crt;
 { elimina linia l si coloana c din matricea m*n}
 {program interesant pt. declararea dim. matricii}
 const min=0; max=99;
 type interval=min..max;   {tip de baza BYTE}
 var  m,n,c,l,i,j:byte;
      a:array[interval,interval] of interval;
 begin
 clrscr;
 write(' introdu m,n:');readln(m,n);
 writeln;
 for i:=1 to m do
  for j:=1 to n do
    begin
    write('a[',i,',',j,']=');readln(a[i,j]);
```

```
    end;
    writeln;
    for i:=1 to m do {ecou date}
     begin
     for j:=1 to n do write(a[i,j]:3);
     writeln;
     end;
    write(' introdu l si c: ');readln(l,c);
     for i:=L to m-1 do {elimina linia L}
       for j:=1 to n do a[i,j]:=a[i+1,j];
     for j:=C to n-1 do    {elimina coloana C}
           for i:=1 to m do a[i,j]:=a[i,j+1];
     dec(n); dec(m); writeln;
     {scrie noua matrice}
     for i:=1 to m do
      begin
       for j:=1 to n do write(a[i,j]:3);
       writeln;
      end;
     write(#7);readln;
  end.
```

3. Fiind dată o sumă în lei să se afişeze numărul minim de bancnote şi monezi pentru achitarea ei.
bancnote: 10 000 5 000 1 000 500 200
monezi: 100 50

EX27.PAS

```
program retributie;
uses crt;
 { exprima o suma cu nr. minim de bancnote si }
 { monezi }
type bancmon=(zecemii,cincimii,omie,cincisute,
              douasute,osuta,cincizeci);
     bani=array[bancmon] of longint;
const
valori:bani=(10000,5000,1000,500,200,100,50);
var  retrib,totretrib:bani;
                    v:bancmon;
                    c:char; {nume salariat}
                    k:1..20
           suma,total:longint;
                 nsal:1..1000 ;
                           {nr. salariati}
                    i:1..1000;
```

```
BEGIN
  clrscr;
  for v:=zecemii to cincizeci do  {initializare}
    totretrib[v]:=0;
    total:=0;
    writeln;
  write(' introdu nr. salariati,nsal=');
  readln(nsal);
  for i:=1 to nsal do
   begin
   writeln;
 writeln('introdu nume salariat,<20 caractere:');
    for k:=1 to 20 do  read(c);
    writeln;
 write(' introdu suma de plata pt. salariat:');
 readln(suma);
    total:=total+suma;
    for v:=zecemii to cincizeci do
     begin
      retrib[v]:=suma div valori[v];
      suma:=suma mod valori[v];
      write(retrib[v]:5);
      totretrib[v]:=totretrib[v]+retrib[v];
      end;
    end;
    writeln;
    writeln('totaluri');
    write(total:6);
    for v:=zecemii to cincizeci do
                  write(totretrib[v]:5);
    readln;
END.
```

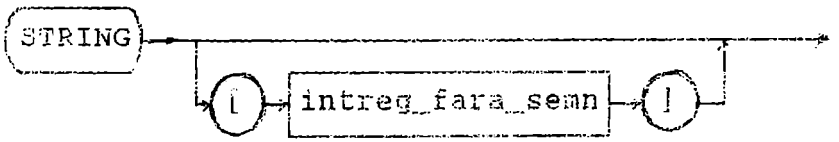
Probleme propuse pentru seminar:

1. Un program care caută un element într-un vector ordonat crescător prin metoda înjumătăţirii intervalului.
2. Fiind dat un vector neordonat să se găsească elementul maxim în valoare absolută şi elementul maxim.
3. Să se verifice dacă o valoare dată c se află printre cele N componente ale unui vector dat şi în caz afirmativ să se afişeze poziţia primei componente din vector egală cu valoarea.

3. Tipul string (şir de caractere)

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

Tipurile string sunt tipuri structurate într-un mod asemănător cu tipurile 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_fara_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 bytes lungimea maximă plus un byte (primul) ce conţine lungimea curentă a variabilei. Caracterele individuale într-un şir sunt indexate de la 1 la lungimea şirului.

Expresii 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: de exemplu
age:='optsprezece';

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 subşirul conţinând Num caractere din şirul St începând cu poziţia Pos.
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: dacă St='abcdefg' atunci DELETE(St,2,4) dă pentru St valoarea 'afg'

INSERT sintaxa: **insert(Obj,Target,Pos)**
scop: înserează 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 truncheate
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 - tip întreg sau real
st - tip string

VAL sintaxă: **Val(St,Var,Code)**
scop: convertește expresia de tip string St într-o valoare întreagă/reală (în funcție de tipul lui var) și 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 poziționat pe primul caracter ce produce eroare și valoarea lui Var este nedefinită.

Funcții pentru șiruri:

copy sintaxa: **copy(St,Pos,Num)**
scop: dă un subșir conținând Num caractere din șirul St începând cu poziția Pos.
St - tip STRING
Pos,Num - tip întreg
. dacă Pos > Length(St) funcția dă un șir vid
. dacă Pos + Num > Length(St) funcția dă doar caracterele din șir
. dacă Pos este în afara domeniului 1..255 funcția dă o eroare în timpul execuției
exemplu: dacă St='abcdefg' atunci
copy(St,4,2) dă 'de'
copy(St,4,10) dă 'defg'

length sintaxa: **length(St)**
scop: dă lungimea reală a lui St
St - tip STRING
rezultat - tip întreg

pos sintaxă: **Pos(obj,target)**
scop: scanează șirul target pentru a găsi șirul 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.

Exemple:

EX28.PAS

```
program sir;
 var s1:string[10];  s2:string[20];  l:integer;
 begin
   s1:='turbo';
   s2:=s1+' pascal';
   l:=length(s2);
   writeln(s2); writeln(' lungime reala=',l)
 end.
```

EX29.PAS

```
program reversie_sir;
Uses crt;
  var str:string;
        i:byte;
  begin
  clrscr;
writeln('introdu un sir de caract.',#17#196#217);
  readln(str);
  writeln(' sirul inversat este: ');
   for i:=length(str) downto 1 do  write(str[i]);
   readln;
  end.
```

Alte probleme:

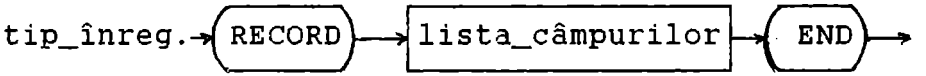
1. Să se scrie câţi studenţi au promovat examenul de analiză, câţi au promovat examenul de mecanică, câţi au promovat examenul de fizica moleculei, câţi au promovat examenul de programare şi pentru fiecare student câte examene a promovat.

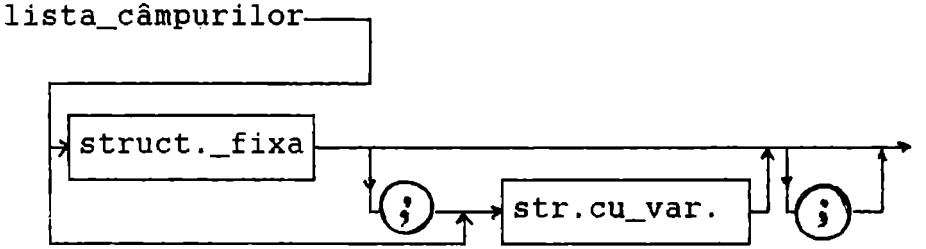
EX30.PAS

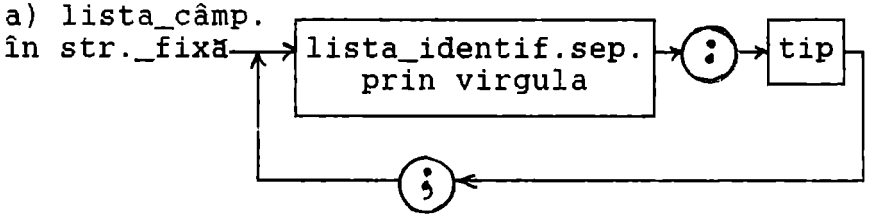
```
program student_examen_promovat;
uses crt;
 const n=5;
 type
examen=(analiza,mecanica,molecula,programare);
exprom= set of examen;
  var stud:array[1..n] of exprom;
      nexprom:array[1..n] of integer;
      i,anal,mec,mol,prog:integer;
      j:examen;
  BEGIN
  clrscr;
  stud[1]:=[analiza,programare];
  stud[2]:=[mecanica,molecula,programare];
  stud[3]:=[];
  stud[4]:=[analiza,mecanica,molecula,programare];
  stud[5]:=[molecula,programare];
  anal:=0;mec:=0; mol:=0; prog:=0;
  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:11);
 writeln(' - " -      fizica moleculei: ',mol:2);
 writeln(' - " -       mecanica: ',mec:10);
 writeln(' - " -       program.: ',prog:7');
  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;
     write(' studentul ',i);
     write( ' a promovat ',nexprom[i]);
     writeln(' examene ');
   end;
 readln;
 END.
```

4. Tipul record (înregistrare).

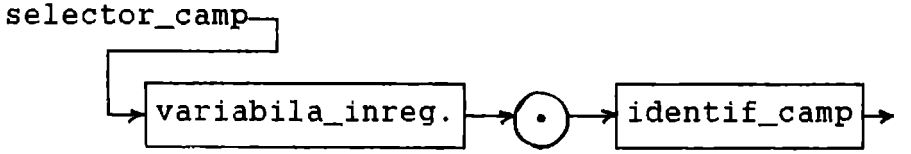
Tipul înregistrare este un tip compus format dintr-un număr de componente, numite **câmpuri**. Spre deosebire de variabilele de tip tablou, câmpurile, elemente ale variabilelor de tip înregistrare, pot fi de tipuri diferite. Fiecare câmp are un nume - **identificatorul de câmp** -. Numărul componentelor poate fi fix sau variabil.
In 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 tipului înregistrare este următoarea:







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



Exemple: TYPE data=RECORD
an:1900..2000;

```
          luna:(ian,feb,mar,apr,mai,iun,iul,
                aug,sep,oct,nov,dec);
            zi:1..31;
                  END;
        VAR astazi:data;
```

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

EX67.PAS

```
PROGRAM exrecord;
Type data=RECORD
          an:1900..2000;
        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:=1996;
astazi.luna:=ian;
astazi.zi:=10;
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 operaţie de atribuire; de exemplu:
data1:=data2;

Alte exemple, folosite în grafică, sunt:

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

la EX59.PAS

```
TYPE pointtype=RECORD
               x,y:integer;
               END;
```
la EX62.PAS

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

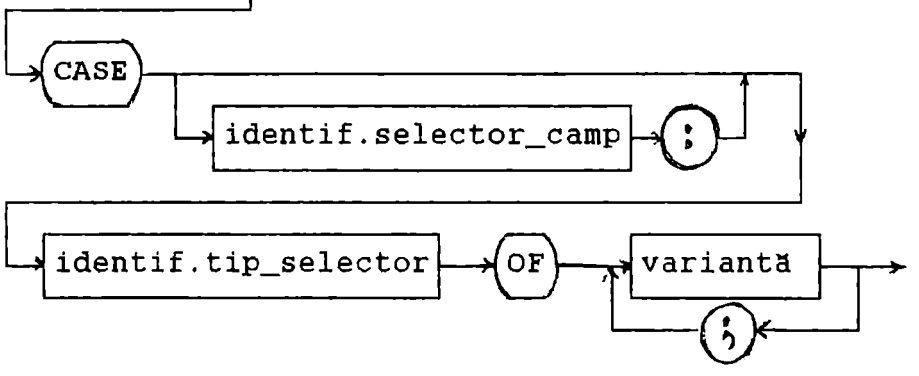
```
TYPE data=RECORD
          an:1900..2000;
        luna:(ian,feb,mar,apr,mai,iun,iul,aug,sep,
              oct,nov,dec);
          zi:1..31;
          END;
     carte=RECORD
           titlu:ARRAY[1..20] of char;
           autor:ARRAY[1..15] of char;
       dateautor:data;
           END;
```

Declarația 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 cărții c

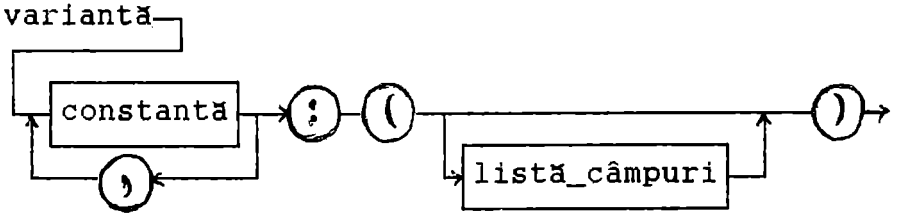
c.dateautor.zi reprezintă ziua nașterii autorului cărții c

b)lista_câmpurilor_ în_structura_ cu_variante



Structura cu variante este folosită când se dorește includerea în structura unei înregistrări a unor informații ce depind de o altă informație

deja prezentă în înregistrare. Diferitele variante sunt selectate de valorile posibile ale identificatorului tip_selector din câmpul identificator selector_câmp. Fiecare variantă este 'etichetată' cu una sau mai multe valori ale tipului selector (separate prin virgulă sau date ca domeniu de valori:vi..vf).



Dacă o variantă este vidă, adică nu are niciun câmp, forma ei este constanta:(). O listă de câmpuri poate să conţină numai o singură clauză CASE, plasată după câmpurile fixe. Tipul selector trebuie să fie ordinal.

Exemplu

```
TYPE studii=(elem,medii,sup);
     informatii=RECORD
                  nume:string[20];
                 salar:50000..300000;
                 CASE preg:studii OF
                      elem:();
                     medii:(bac:real);
              sup:(anabs:1950..2000;stat:boolean);
                  END;
VAR  x,y,z:informatii;
     Variabilele  x,y,z pot primi valori astfel:
x.nume:='John'; x.salar:=100000; x.preg:=elem;
y.nume:='Mary'; y.salar:=200000; y.preg:=medii;
 y.bac:=8.50;
z.nume:='Tom';  z.salar:=300000; z.preg:=sup;
 z.anabs:=1990; z.stat:=true;
```

Numai o singură variantă poate fi activă la un moment dat. Astfel dacă este activă varianta medii pentru y.preg, nu are sens o atribuire de forma y.anabs:=1980. Atenţie, compilatorul nu poate să semnaleze astfel de erori logice!

O categorie specială de înregistrări cu va-

riante apare când nu e specificat numele câmpului selector ci numai tipul selector ca în exemplul:

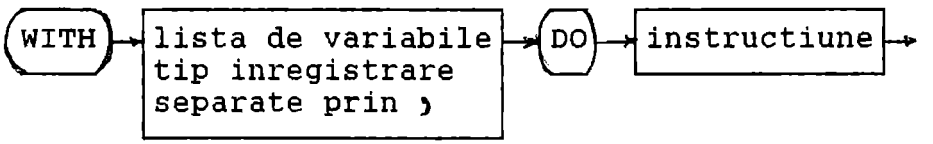
```
TYPE   c=(unu,doi,trei);
     art=RECORD
         CASE c of   {fara camp selector}
          unu:(u:ARRAY[1..4] of char);
          doi:(d:ARRAY[1..2] of integer);
         trei:(t:ARRAY[1..4] of byte);
         END;
VAR   s:art;
```

Atunci printr-o atribuire
s.d:='abcd';
valoarea depusă este interpretată în moduri diferite (citind-o, de exemplu):
s.u are valoarea ('a','b','c','d')
s.d are valoarea ($6162,$6364) adică (25185,25699)
s.t are valoarea ($61,$62,$63,$64)

Această facilitate permite efectuarea implicită a diferitelor conversii.

Lungimea unei variabile de tip înregistrare este egală cu lungimea părţii fixe propriu zise plus lungimea câmpului selector (dacă există) plus lungimea celei mai mari părţi variabile.

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



In 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 indexării 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 echivalenta cu t[5].x:=1; t[5].y:=2;
```

Constante de tip înregistrare

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;
     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:15;luna:09;an:1995);
```

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

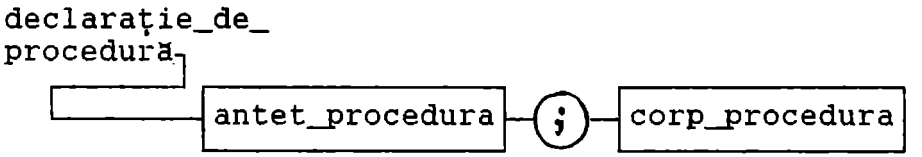
VI. Proceduri şi funcţii.

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**).

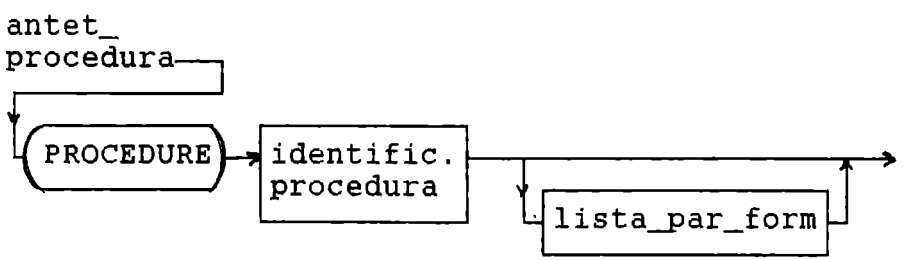
1. Proceduri.

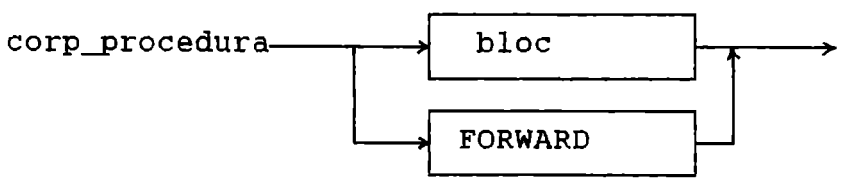
O declaraţie de procedură asociază un **identificator** cu un bloc de procedură.

i) **Diagrama de sintaxă** pentru cazurile simple, pe care le vom studia, este următoarea:



în care:





= In antetul procedurii se dă **identificatorul pentru procedură** şi, dacă există, **parametrii formali**. Procedura este apelată în cadrul programului prin numele ei urmat, dacă există, de parametrii actuali.

= In 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 activează pe Q iar Q activează pe P. Este clar că în acest caz oricare ar fi forma programului utilizarea numelui uneia din proceduri precede locul ei de definiţie. Pentru rezolvarea acestei probleme 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 - ca în exemplul următor:

```
PROCEDURE Q(x,y,z:integer);    FORWARD  ;
PROCEDURE P(u,v,w:integer);
  begin
  ...
  Q(1,2,3)
  ...
  end;
PROCEDURE Q;  {nu se repeta declararea
               parametrilor lui Q}
  begin
  ...
   P(5,6,7);
  ...
  end;
...
```

ii) Exemple de proceduri simple; 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 cmmdc, modifică numerele a și b, ele vor fi copiate în x și y. Etapele programului vor fi deci:

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

EX4P.PAS

```
program fractie;
{simplificare fractie a/b prin impartire cu}
{c.m.m.d.c.}
  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;   {cmmdc}
   begin   {programul principal}
     write(' introdu numaratorul');
writeln('si numitorul ca nr. intregi, pozitive');
     read(a,b);
     writeln('fractie 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('fractie simplificata: ',a,'/',b)
       end
     else
      writeln(' nu se poate simplifica')
   end.
```

In exemplul dat mai sus declarația de procedură nu conține partea de declarație de aceea toate variabilele, declarate în programul principal, sunt **variabile globale.**

Există posibilitatea de a scrie o parte de declarație în interiorul procedurii la fel ca în orice program în TP. Identificatorii introduși în partea de declarație a procedurii sunt **locali.** ei pot fi referiți și cunoscuți numai în blocul în care au fost declarați, acesta reprezentând domeniul acestor identificatori.

Iată cum arată exemplul de mai sus cu variabilele x,y,z locale.

EX4S.PAS

```
program fractie2;
uses crt;
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
clrscr;
write('introdu a si b, nr. intregi,pozitive: ');
readln(a,b);
writeln(' fractie nesimplificata: ',a,'/',b);
cmmdc;
if c>1 then
 begin
 a:=a div c;  b:=b div c;
 writeln(' fractie simplificata: ',a,'/',b);
 end
else writeln('fractia nu se poate simplifica');
readln;  end.
```

iii) Domeniul de valabilitate al obiectelor.

Fiecare corp de procedură poate conține în partea sa de declarație o declarație de procedură sau funcție (procedură sau funcție inclusă sau locală). Prin **obiecte** înțelegem constante, tipuri, variabile, proceduri, funcții, identificate prin identificatorul asociat.

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

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

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

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

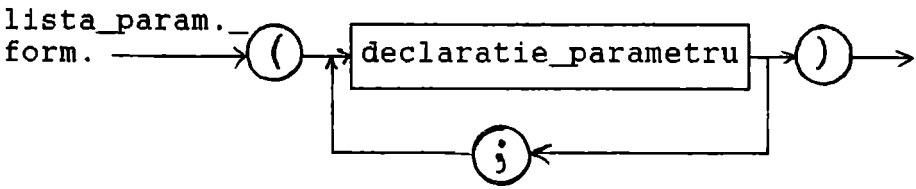
d) o procedură se poate referi la ea însăși (**apelare recursivă**).

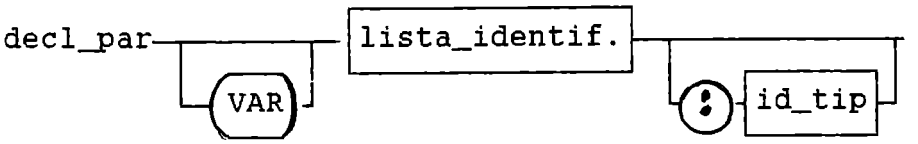
Aceste reguli determină și durata de viață a identificatorilor; o variabilă declarată ca locală într-o procedură există numai în timpul execuției procedurii fiind creată la activarea procedurii prin alocarea de memorie și distrusă la ieșirea din procedură prin eliberarea memoriei ocupate.

iv) 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. Când există, lista parametrilor formali are diagrama de sintaxă următoare;





Există 3 feluri de parametri: **valoare, variabile, variabile fără tip**, caracterizaţi astfel:

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

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

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

Acţiunea lor este următoarea:

a) 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.

b) 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 (se poate evita această restricţie prin folosirea unor parametri formali variabilă fără tip).

c) 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 fractie2 cu procedura cmmdc cu parametri valoare și apoi cu parametri valoare și variabile.

EX32.PAS

```
program fractie3;
uses crt;
 var a,b,c:integer;
 procedure cmmdc(x,y:integer);
  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}
  end;   {end cmmdc}
 begin   {program principal}
 clrscr;
 write(' introdu a si b,nr. intregi, a>b,: ');
 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(' fractie simplificata: ',a,'/',b);
     end
 else writeln(' cmmdc=',c);
 readln;
 end.
```

EX33.PAS

```
program fractie4;
```

```
uses crt;
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  {prog. princ.}
 clrscr;
 write(' introdu a,b,nr. intregi:');readln(a,b);
 writeln(' fractie 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 în următorul exemplu : un program ce converteşte o sumă (între 0 şi 999) din cifre în cuvinte.

EX34.PAS

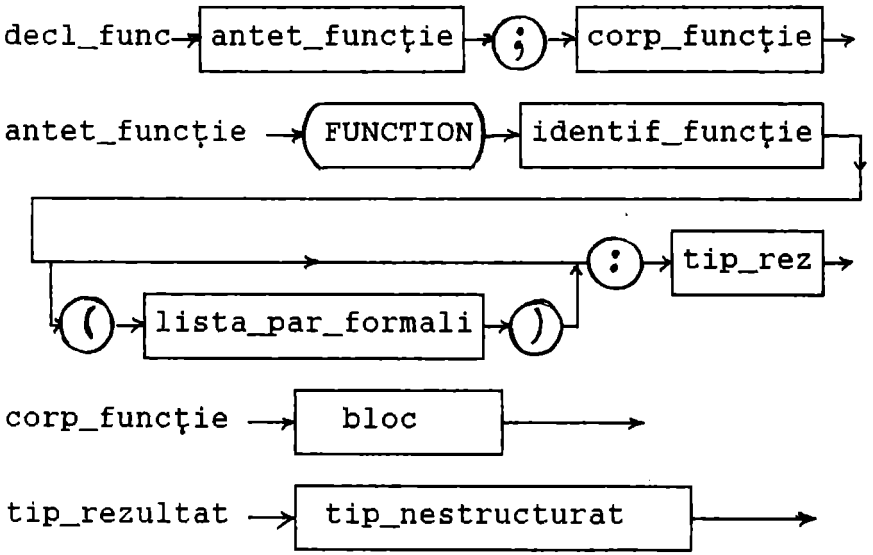
```
program lei_cuvinte;
uses crt;
 type suma=0..999;
 var lei:suma;
 procedure convincuv(x:suma);
    {converteste un numar >0 si <999}
    { in cuvinte}
```

```
type cifra=0..9;
var sute,zeci,unita: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}
{separarea cifrelor numarului}
sute:= x div 100; zeci:= x mod 100 div 10;
unita:=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 unita>0 then
```

```
    begin
     rang:=unu;
     if zeci >1 then write(' si ');
     unitati(unita);
     if zeci=1 then write('sprezece ');
    end
  else
   if zeci=1 then write(' zece ');
 end;   {end convincuv}
 begin   {prog. princ.}
 clrscr;
  write(' introdu suma in cifre <999 :');
  readln(lei);
  convincuv(lei);
  writeln(' lei');readln;
end.
```

2. Funcţii.

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

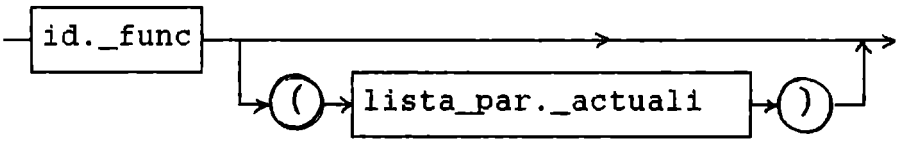


NOTĂ : a) lista parametrilor formali conţine doar parametri valoare (ei având rolul datelor de intrare din declaraţia de procedură) şi parametri variabilă fără tip.

b) 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. In partea executabilă a funcţiei trebuie să apară cel puţin o atribuire prin care se transferă identificatorului funcţiei valoarea rezultatului calculat.

Apelarea sau activarea funcţiei se face conform următoarei diagrame de sintaxă:



Exemple:

1. Un program ce găseşte o rădăcină a ecuaţiei

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

fiind dat intervalul în care ea se află (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.

EX35.PAS

```
program rezec4;
uses crt;
label zero,unu,doi;
const eps=1.e-05;
var a,b,c:real; i:integer;
   ai:array[1..4] of real;    {limita inf. a int.}
   bi:array[1..4] of real;    {limita sup. a int.}
function f(x:real):real;
  begin
   f:=sqr(x)*sqr(x)-9*sqr(x)*x+120*x-130.
  end;
begin
```

```
clrscr;
write('introdu limitele a,b');
write(' pentru cele 4 intervale in care se afla');
writeln(' o radacina:(-4,-3),(1,2),(4,5),(6,7) ');
for i:=1 to 4 do
begin
zero:  readln(a,b);
   {verifica corectitudinea intervalelor}
  if f(a)*f(b)>0 then
  begin
writeln(' interval prost dat,reintrodu a,b: ');
goto zero;
  end;
  ai[i]:=a; bi[i]:=b;
end;
for i:=1 to 4 do
{incepe cautarea radacinilor in intervalele date}
  begin
   a:=ai[i]; b:=bi[i];
   unu: c:=(a+b)/2;
   if f(c)=0 then
     begin
      writeln('x(',i,')=',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,')=',(a+b)/2); goto doi;
     end
     else goto unu;
   doi: end;
   readln;
end.
```

2. Un program ce calculează

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

folosind funcţia fact(n).

EX36.PAS

```
program comb_mk;
uses crt;
```

```
var m,k:integer;
      c:real;
function fact(n:integer):longint;
 var i:integer; f:longint;
 begin
  f:=1;
  for i:=1 to n do f:=f*i;
  fact:=f;
 end;   {end fct fact}
begin
clrscr;
write('introdu m, k,intregi,pozitive,m>k,m<12 ');
readln(m,k);
 c:=fact(m)/fact(k)/fact(m-k);
 write(' comb. de ',m,' luate cate ',k,);
 writeln(' = ',c);readln;
end.
```

3. Folosind funcţia fact(n) scrieţi programul ce afişează toate numerele N, N=1,2,... care satisfac condiţia

$$S_n=N!+(N+1)!$$

pătrat perfect.

NOTĂ: condiţia pătrat perfect înseamnă că
SQR(TRUNC(SQRT(S_n))=S_n

EX31.PAS

```
program sn;
uses crt;
var  n:integer;
     s,s12:real;
function fact(m:integer):longint;
 var i:integer; sf:longint;
 begin
 sf:=1; for i:=2 to m do sf:=sf*i; fact:=sf;
 end;  {end function}
begin                  {main program}
  clrscr;
  for n:=1 to 15 do
    begin
    s:=1.*fact(n)+1.*fact(n+1);  s12:=sqrt(s);
```

```
    if sqr(trunc(s12))=s then
  writeln(n,' da sn= ',s,' = ',s12,' la patrat');
    end;readln;
end.
```

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

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

va avea antetul

```
FUNCTION
integS(A,B:REAL;N:INTEGER;F:FCT):REAL;
```

Tipul FCT este definit in programul apelant astfel:

```
TYPE fct=function(x:real):real;
```

Parametri 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-}.

Exemplu

1. Să se calculeze, folosind metoda Simpson, integrala

$$K(\theta)=\int_0^{\pi/2}\frac{d\varphi}{\sqrt{1-\sin^2(\theta)\sin^2(\varphi)}} \quad pentru\ \theta=30^\circ\ si\ 60^\circ$$

```
EX37.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 {prog.princ.}
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,')=',s1);
```

```
writeln('K(',t2,')=',s2);
readln;
end.
```

2. Pentru a calcula integrala

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

se vor folosi funcţiile

```
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-}
```

şi se va scrie programul principal, folosind funcţia Simpson din exemplul anterior.

EX37P.PAS

```
program calcul_int_simpson;
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-}
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 {prog.princ.}
clrscr;
s1:=simpson(-1.,1.,10,fs);
writeln('integrala este: ',s1);
readln;
end.
```

Observaţie:{sugestie dată de studentul Lişcă Gabriel, gr.105, anul univ.1995-1996).

Salvaţi funcţia simpson într-un fişier cu numele simpson.inc

De câte ori aveţi nevoie de ea folosiţi directiva de compilare {$i simpson.inc} înainte de BEGIN {program principal}; în acest fel nu mai este nevoie să o tastaţi în programul d-voastră.

Astfel programul de mai sus devine:

EX37T.PAS

```
program calcul_int_simpson;
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 {prog.princ.}
clrscr;
s1:=simpson(-1.,1.,10,fs);
writeln('integrala este: ',s1);
readln;
end.
```

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:

a) numerele naturale: 1 fiind un număr natural, succesorul lui este tot un număr natural. Factorialul n!=n*(n-1)! cu 0!=1 şi n>0
b) polinoamele Legendre

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

$$cu\ P_0(u) = 1\ si\ P_1(u) = u$$

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

Puterea recursivă rezidă 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 număr întreg şi pozitiv să se scrie numărul obţinut prin citirea cifrelor numărului dat de la dreapta la stânga (numărul răsturnat).

```
EX38.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);
 write(' numarul rasturnat este ');
 invers(m);readln;
 end.
```

2. Fiind dat un număr întreg pozitiv să se calculeze factorialul său ca în EX36.PAS dar cu o funcţie cu apel recursiv. Să se folosească funcţia factorial pentru calculul combinărilor de m luate câte k.

```
EX39.PAS
program comb_mk_rec;
uses crt;
 var m,k:integer;
       c:real;
 function factorial(n:integer):longint;
  begin
   if n=0 then factorial:=1 else
               factorial:=n*factorial(n-1);
  end;
 begin
 clrscr;
 {program principal}
 write('introdu m,k,intregi,pozitive,m>k,m<12:');
 readln(m,k);
 c:=factorial(m)/factorial(m-k)/factorial(k);
 write(' combinari de ',m,' luate cate ',k);
 writeln(' = ',c);  readln;  end.
```

Recursivitatea poate fi întotdeauna transformată în iteraţie. În majoritatea cazurilor forma nerecursivă a unui program este mai eficientă decât cea 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 unui număr natural N ca o sumă de alte numere naturale, fiecare din ele nedepăşind o valoare M.

Algoritmul este următorul: 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)=P9n,m-1)+P(n-m,m) dacă n>m

Varianta iterativă ar fi greoaie în timp ce varianta recursivă este:

EX40.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 fct. p}
 begin   {program principal}
 clrscr;
 write(' introdu n si m: ');readln(l,k);
 write(' numar de partitii p( ',l,' , ',k);
writeln(' )= ',p(l,k));  readln;
 end.
```

Exemple cunoscute de apelare recursivă justificată sunt programele ce rezolvă probleme ca: turnurile din Hanoi, problema celor 8 regine,..., descrise în /1/ .

VII. Unit-uri, unit-ul GRAPH.

Primele versiuni ale compilatorului TURBO PASCAL nu permiteau scrierea programelor mai mari de 64 kbytes deoarece procesorul 8086 limitează dimensionarea unui segment la această valoare. Începând cu versiunea 4.0 a fost introdusă 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 clauză **uses**. Lungimea unui unit rămâne limitată la 64 Kb 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.

Există 8 unit-uri standard în TP, fiecare cu un profil şi o structură bine delimitată: **SYSTEM, DOS, OVERLAY, CRT, GRAPH, PRINTER, GRAPH3, TURBO3.**

. Unit-ul System conţine toate procedurile şi funcţiile standard din TP. 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 3 unit-uri se găsesc în fişierul Turbo.TPL

. Unit-ul Crt (character) permite utilizarea funcţiilor şi procedurilor pentru comanda funcţionării ecranului alfanumeric. Se foloseşte cu clauza **Uses Crt;**

Exemplu: pentru folosirea comenzii:
ClrScr (Clear Screen - sterge ecranul)
sau a funcţiilor:
KeyPressed:boolean; ReadKey:char;
e necesară scrierea clauzei Uses Crt;
după antetul progra-mului.

. Unit-ul Graph conţine procedurile şi funcţiile grafice (în fişierul GRAPH.TPU). Aceste

subprograme au nevoie de informaţiile din fişierele *.BGI şi *.CHR. 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 3.0

Exemplu: Uses Dos,Crt,Graph;

Programele în TP permit atât folosirea unit-urilor standard enumerate mai sus cât şi scrierea şi folosirea unit-urilor utilizator (scrise de utilizator).

Unit-ul GRAPH.

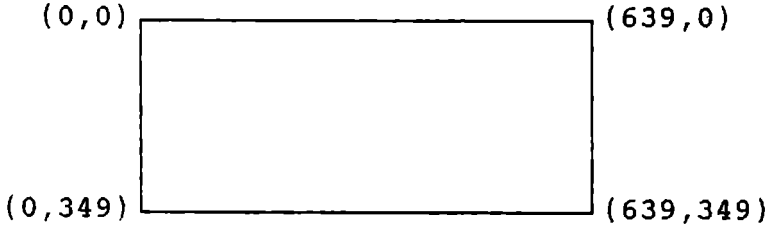
În cele ce urmează ne vom opri asupra unit-ului Graph. Programele care utilizează subprogramele din acest unit trebuie să conţină directiva Uses Graph; după antetul programului.

Subprogramele unit-ului Graph pot fi clasificate astfel:

1. iniţializare mod grafic
2. tratare erori grafice
3. definire de ferestre şi pagini
4. reprezentare puncte
5. reprezentare linii
6. reprezentare cercuri şi alte curbe
7. reprezentare poligoane şi haşurări
8. scriere de texte

...

Ecranul grafic pentru **placă VGA (640x350)** este



1. Inițializare mod grafic.

Pentru inițializarea modului grafic se poate folosi, de exemplu:

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

Procedura **InitGraph** folosită aici inițializează sistemul grafic prin încărcarea driverului grafic corespunzător echipamentului (la noi EGAVGA.BGI pentru placa IBM tip VGA), pune sistemul în mod grafic după care redă controlul programului apelant. Procedura se definește prin:

```
PROCEDURE InitGraph(var gd:integer;
                    vargm:integer;cd:string);
```

unde gd și gm vor conține valorile returnate de InitGraph și reprezintă codul corespunzător driverului și modului grafic. Variabila cd de tip string, în momentul apelului, conține calea spre driverul *.BGI unde sunt stocate fișierele corespunzătoare driverelor (la noi ea este 'F:\tp\BGI' sau '\tp\bgi' hard-discul curent fiind f:).

Variabila gd se inițializează cu constanta predefinită **const detect=0;** (gd:=detect;) pentru a se omite apelarea procedurii **DetectGraph** înaintea apelării procedurii InitGraph (DetectGraph testează hardware-ul și determină driverul și modul grafic utilizabil).

```
PROCEDURE DETECTGRAPH(VAR gd,gm:integer);
```

Funcțiile **GetMaxX** și **GetMaxY** dau valorile maxime pentru x și, respectiv, y în modul grafic actual (De exemplu pentru placă VGA cu rezoluția 640x350 pixeli aceste valori sunt respectiv 639 și 349).

Funcția **GetDriverName**, apelată după activarea procedurii InitGraph, dă numele driverului grafic actual

```
Function GetDriverName:string;
```

Funcţia **GetGraphMode** dă codul modului grafic actual

```
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, arcelor cu procedura **SetAspectRatio.**

```
GetAspectRatio(var xasp,yasp:word);
SetAspectRatio(xasp,yasp:word);
```

Exemplu:

EX43.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
  ggm:=getgraphmode;
  gdn:=getdrivername;
  getaspectratio(xaspect,yaspect);
  aspratio:=xaspect/yaspect;
  closegraph;
  writeln('graphdriver: ',gdn);
  writeln('graphmode: ',ggm);
  writeln(' aspectratio: ',aspratio:2:2);
  readln;
end.
```

2. Erori grafice.

Erorile interne ale unit-ului Graph sunt returnate de funcţia **GraphResult** sub forma unui **cod de eroare**. Dacă codul returnat este zero operaţia grafică a reuşit; dacă codul returnat este mai mic decât zero operaţia a eşuat.

```
Function GraphResult:integer;
```

```
Const
   grOK=0;    {operatie reusita; nici o eroare}
   grNoInitGraph=-1;
        {grafica BGI neinstalata cu InitGraph}
   grNotDetected=-2
        {grafica hardware nedetectata}
   grFileNotFound=-3
        {fisier driver *.BGI nu a fost gasit}
...
```

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

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
In continuare toate coordonatele vor fi relative la fereastră.

clip determină dacă liniile din afara ferestrei sunt sau nu vizibile.

Constantele predefinite pentru stabilirea valorii variabilei clip sunt:

```
const
  clipon=true;
```

```
      {taierea este activa; exterior invizibil}
  clipoff=false;
      {taierea este pasiva; 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 **Bar** pentru trasarea unui dreptunghi corespunzător ferestrei şi point-erul actual este mutat în colţul stânga-sus al ferestrei, punctul de coordonate relative (0,0).

Exemplu:

EX50P.PAS

```
program ex_viewport;
uses crt,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);
randomize;readln;
for i:=1 to 10 do
lineto(random(getmaxx)-180,random(getmaxy)-180);
readln;
rectangle(centrux+9,centruy+9,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.
```

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 0 şi valoarea dată de funcţia **GetMaxColor**. (max=15 pentru placă VGA).

Function GetMaxColor:word;

Exemplu:

EX57.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,gm,'\tp\bgi');
     if graphresult<>grok then halt(1);
    end;
function randcul: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.
```

5. Reprezentare linii.

Pentru a desena linii se folosesc procedurile:

Procedure 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:

Procedure 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 ca constante:

```
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:

```
Procedure
   SetLineStyle(linestyle:word;pattern:word;
                thickness:word);
```

Constantele pentru linestyle sunt:

```
CONST
solidln=0; dottedln=1; centerln=2; dashedln=3;
userbitln=4; {user-defined line style}
```

iar pentru grosime:

```
CONST
normwidth=1; thickwidth=3;
```

Variabila **pattern** este ignorată cât timp linestyle<>4. Când linestyle=4, linia e 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}
```

Exemplu:

EX58.PAS

```
program ex_line;
uses graph;
var gd,gm,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,'userbitline,thickwidth');
readln;
closegraph;
END.
```

Procedure 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 Line.

Procedure 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 Line. Deplasarea pointerului în punctul de coordonate (x,y) se face cu:

Procedure MoveTo(x,y:integer);

Dacă este definită o fereastră, valorile (x,y) sunt relative la fereastră.

Procedure MoveRel(dx,dy:integer);
mută pointerul din poziţia curentă la un punct definit de distanţele dx,dy.

Exemplu:

EX59.PAS

```
program linii;
uses graph;
var gd,gm:integer; info:viewporttype;
```

```
    pas,stil,x,y:word; s:string;
procedure grstart;
     begin
     gd:=detect; initgraph(gd,gm,'\tp\bgi');
     if graphresult<>grok then halt(1);
     end;
BEGIN
 grstart;
 getviewsettings(info);
 with info do pas:=(y2-y1) div 12;
 x:=35;y:=20;
 for stil:=0 to 3 do
  begin
  setlinestyle(stil,0,normwidth);
  line(x,y,x+100,y);
  str(stil,s);outtextxy(x,y+pas div 2-3,s);
  y:=y+pas;
  setlinestyle(stil,0,thickwidth);
  line(x,y,x+100,y);
  y:=y+pas;
  end;
  setlinestyle(userbitln,$c3,normwidth);
  line(x,y,x+100,y);
  outtextxy(x,y+pas div 2-3,'stil utilizator');
  y:=y+pas;
  setlinestyle(userbitln,$c3,thickwidth);
  line(x,y,x+100,y);
  readln;  closegraph;
END.
```

6. Reprezentare cercuri, arce de cerc...

Pentru a desena arce de cerc, cercuri și alte curbe se folosesc procedurile:

Procedure Circle(x,y:integer;raza:word);

desenează un cerc cu centru în punctul de coordonate (x,y) și 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) și raza dată începând de la unghiul ustart și sfârșind la unghiul ufinal, ambele date în grade în sensul trigonometric.

```
Procedure  Ellipse
(x,y:integer;ustart,ufinal:word;xraza,yraza:word);
```

desenează un arc de elipsă, (x,y) fiind coordonatele centrului, ustart,ufinal - unghiurile de început şi sfârşit în sens trigonometric, xraza, yraza - axele orizontale 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 haşurare sunt definite cu:

```
Procedure SetFillStyle(pattern:word;color:word);
```

Modelul implicit este compact (solid) şi culoarea implicită este culoarea maximă 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 ////}
bkslashfill=5;{umple aria cu modelul \\\\}
ltbkslashfill=6;{  -"-                \\\\}
hatchfill=7;   {   -"-                ||||}
xhatchfill=8;  {   -"-                xxxx}
interleavefill=9; widedotfill=10;
closedotfill=11; userfill=12;
```

```
Procedure SetFillPattern
        (pattern:fillpatternType;color:word);
```

umple figura cu un model definit de programator în culoarea definită de 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. Pentru fiecare bit al unui octet poziţionat la 1 este aprins un pixel. Următorul exemplu dă o

suprafaţă uniform gri (nuanţă de gri 50%) .

EX60.PAS

```
program ex_setmodel;
uses graph;
const
gray50:fillpatterntype=($AA,$55,$AA,$55,$AA,$55,
                        $AA,$55);
var gd,gm:integer;
BEGIN
 gd:=detect;
 initgraph(gd,gm,'\tp\bgi');
 if graphresult<>grok then halt(1);
 setfillpattern(gray50,white);
 pieslice(200,100,0,270,50);
 readln; closegraph;
END.
```

Incercaţi şi

EX60P.PAS

```
program ex_setmodel;
uses graph;
const gray50:fillpatterntype=($c3,$00,$c3,$00,
                              $c3,$55,$c3,$55);
var gd,gm:integer;
BEGIN
 gd:=detect; initgraph(gd,gm,'\tp\bgi');
 if graphresult<>grok then halt(1);
 setfillpattern(gray50,3);
 pieslice(200,100,0,270,50);
 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.

Exemplu:

EX61.PAS

```
program ex_raport_aspect;
uses graph;
var gd,gm,centrux,centruy,i:integer;
     info:viewporttype;
     raza,xasp,yasp,razapas:word;
procedure grstart;
   begin
    gd:=detect; initgraph(gd,gm,'\tp\bgi');
    if graphresult<>grok then halt(1);
   end;
BEGIN
  grstart;getviewsettings(info);
  with info do
   begin
    centrux:=(x2-x1) div 2;
    centruy:=(y2-y1) div 2;
    raza:=3*((y2-y1) div 5);
   end;
  razapas:=raza div 30;
  circle(centrux,centruy,raza);
  readln;
  getaspectratio(xasp,yasp);
  for i:=1 to 30 do
    begin
     setaspectratio(xasp,yasp+(i*getmaxx));
     circle(centrux,centruy,raza);
     raza:=raza-razapas;
    end;
  raza:=raza+25*razapas;
  for i:=1 to 25 do
    begin
    setaspectratio(xasp+(i*getmaxy),yasp);
    if raza > razapas then raza:=raza-razapas;
    circle(centrux,centruy,raza);
    end;
  readln;
  setaspectratio(xasp,yasp);
  closegraph;
END.
```

7. Reprezentare poligoane.

Pentru a desena poligoane se folosesc procedurile:

Procedure rectangle(x1,y1,x2,y2:integer);

deseneaza un dreptunghi cu
(x1,y1) coordonatele colţului stânga-sus
(x2,y2) coordonatele colţului dreapta-jos

Procedure Bar(x1,y1,x2,y2:integer);

deseneaza un dreptunghi şi-l coloreaza în culoarea şi modelul stabilite cu procedurile SetFillStyle sau SetFillPattern.(x1,y1) şi (x2,y2) au aceeaşi semnificaţie ca în procedura rectangle.

Procedure
Bar3d(x1,y1,x2,y2:integer;ad:word;top:boolean);

deseneaza un paralelipipeg dreptunghi şi-l haşurează în culoarea şi modelul curente. (x1,y1) şi (x2,y2) au aceeaşi semnificaţie ca în procedurile rectangle sau bar,
ad este adâncimea corpului,
top = TRUE deasupra paralelipipedului se poate aşeza un alt paralelipiped
FALSE deasupra paralelipipedului nu se poate aşeza alt paralelipiped
In secţiunea de interfaţă a unit-ului GRAPH sunt definite următoarele constante:

```
CONST
topon=TRUE;
topoff=FALSE;
```

Procedure DrawPoly(nrpct:word,var puncte);

deseneaza o linie poligonală în culoarea şi stilul curente.
nrpct reprezintă numărul vârfurilor poligonului,
puncte este un parametru variabilă fără tip care conţine coordonatele fiecărei intersecţii în poligon. De notat că pentru a desena o figură închisă cu n vertecşi trebuie să se transmită **n+1** coordonate şi
puncte[n+1]=puncte[n] .

Exemplu:

```
EX62.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)divsizeof(pointtype),
           triunghi);
  readln;  closegraph;
END.
```

NOTĂ: funcţia SizeOf(x) dă numărul de octeţi ocupat de argument; x este o referire de variabilă sau un identificator de tip.

Procedure
FillPoly(numpuncte:word;VAR punctepolinom);

desenează şi umple un polinom în culoarea şi modelul puse cu procedurile SetFillStyle sau SetFillPattern.

Exemplu:

```
EX62P.PAS
program ex_poligon;
  {exemplul anterior dar cu procedura FillPoly}
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);
  fillpoly(sizeof(triunghi)divsizeof(pointtype),
           triunghi);
  readln;  closegraph;
END.
```

8. Scrierea textelor.

Pentru scrierea textelor în modul grafic sunt incluse fonturi (tipuri de caractere) 8*8 bit-mapped şi câteva 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 fontul.

Scrierea şi selecţia fonturilor se face cu:

```
Procedure SetTextStyle
      (font:word;direction:word;charsize:word);
```

Constantele pentru font, 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:

```
Procedure
   SetUserCharSize(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 este rezultatul apelului uneia din următoarele proceduri:

Procedure OutText(text:string);

unde text este textul trimis la ieșire la poziția curentă a pointerului (CP) folosind parametri de formatare curenți puși cu SetTextStyle.

Procedure OutTextXY(x,y:integer;text:string);

cu care textul este scris începând din punctul de coordonate (în pixeli) (x,y).

Exemple:

```
EX63.PAS
program ex_dimens_var_font;
uses graph;
var gd,gm:integer;
BEGIN
  gd:=detect; initgraph(gd,gm,'\tp\bgi');
  if graphresult<>grok then halt(1);
  settextstyle(triplexfont,horizdir,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(10,10,' marit');
  readln;closegraph;
END.

EX64.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,'\tp\bgi');
  if graphresult <>grok then halt(1);
  with fer1 do
    begin     {cadru fereastra}
    setlinestyle(3,0,1);rectangle(succ(x1),
                 succ(y1),pred(x2),pred(y2));
    setviewport(x1,y1,x2,y2,clipon);
    outtextxy(10,10,' fereastra 1');
```

```
    end;
    {ecranul intreg}
    setviewport(0,0,getmaxx,getmaxy,clipon);
    with fer2 do
     begin
     setlinestyle(4,$c3,1); rectangle(succ(x1),
                  succ(y1),pred(x2),pred(y2));
     setviewport(x1,y1,x2,y2,clipon);
     outtextxy(10,50,' fereastra a doua');
     end;   readln;     closegraph;
END.
```

Incercați și:

```
EX105.pas
program EU;
{ARUNCAREA SUB UNGHI CU LOVIREA UNEI TINTE}
{Negret Alexandru, gr.108, anul univ.1995-1996}
{ programul cere coordonatele tintei; apoi}
{ dand viteza si unghiul se incearca lovirea}
{ tintei; iesirea din program se face dand v=0}
uses crt,graph;
var   gd,gm,f,k,v,x,y:integer;  a:real;
const pi=3.1415;
begin
write('x(10-490),y(10,190) :');readln(x,y);
gd:=detect;initgraph(gd,gm,'\tp\bgi');
if graphresult<>grok then halt(1);
setfillstyle(2,3); bar(0,0,639,349);
setviewport(50,50,550,250,clipon);
setfillstyle(4,0); bar(0,0,500,200);
outtext('v,a: ');  circle(x,200-y,3);
repeat
readln(v,a);if v=0 then halt;
a:=pi*a/180;
for f:=1 to 500 do
 begin
k:=trunc(200-(f*sin(a)/cos(a)-9.81*f*f/2/v/v/
         cos(a)/cos(a)));
putpixel(f,k,7);
if (f=x) and ((k>200-y-3) and (k<200-y+3)) then
                           outtextxy(x,y,'Au!');
 end;
until v=0;
repeat until keypressed;
end.
```

VIII. Elemente de prelucrare statistică a datelor experimentale cu programe în TP.

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.

Iată un exemplu dat în /5/ care arată de ce e necesar să estimăm erorile cu care facem măsurătorile. Se ştie, din măsurători anterioare, că viteza luminii este c=(3.09 +/- 0.15) 10^8 m/s. O nouă valoare măsurată dă 2.998 10^8 m/s; înseamnă că viteza luminii a variat cu 3% (ceea ce ar însemna o descoperire epocală) sau această nouă valoare e compatibilă, în cadrul erorii de măsură, cu vechea valoare?

Foarte adesea în procesul de măsurare a unei mărimi fizice obţinem valori diferite la repetarea măsurătorii în aceleaşi condiţii fizice. Aceasta fie pentru că mărimea fizică respectivă are un caracter statistic (exemplul numărului de dezintegrări într-o secundă ale unui element radioactiv, lungimea drumului între 2 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 (exemple: erorile de calibrare a ceasului, riglei, balanţei folosite la măsurarea intervalelor de timp, a lungimilor sau a maselor).

In 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 în exact aceleaşi condiţii ale mărimii fizice şi x_i valoarea obţinută într-o măsurătoare. Valoarea medie x a şirului finit de N măsurători va fi media aritmetică a valorilor x_i:

$$\overline{x} \equiv \frac{1}{N} \sum_{i=1}^{N} x_i \qquad (1)$$

iar abaterea valorii x_i de la x va fi Δ_i dată de:

$$\Delta_i = x_i - \overline{x} \quad \dashrightarrow \quad x_i = \overline{x} + \Delta_i \qquad (2)$$

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

$$\overline{\Delta} \equiv \frac{1}{N}\sum_{i=1}^{N} \Delta_i = \frac{1}{N}\sum_{i=1}^{N} (x_i - \overline{x}) = \qquad (3)$$

$$\frac{1}{N}\sum_{i=1}^{N} x_i - \overline{x} = \overline{x} - \overline{x} = 0$$

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

$$\sigma_x^2 = \overline{\Delta^2} \equiv \frac{1}{N}\sum_{i=1}^{N} \Delta_i^2 = \frac{1}{N}\sum_{i=1}^{N} (x_i - \overline{x})^2 = \qquad (4)$$

$$\frac{1}{N}\sum_{i=1}^{N} (x_i^2 - 2x_i\overline{x} + \overline{x}^2) = \overline{x^2} - \overline{x}^2 > 0$$

$$de\ unde\ si \quad \overline{x^2} > \overline{x}^2$$

adică media pătratelor este mai mare decât pătratul mediei unui şir de valori.

Pentru caracterizarea împrăştierii se mai foloseşte mărimea:

$$\sigma_x = \sqrt{\overline{\Delta^2}} \tag{5}$$

numită abatere standard şi abaterea standard relativă definită de:

$$\epsilon_x = \frac{\sigma_x}{\overline{x}} \tag{6}$$

numită şi fluctuaţie.

2. Propagarea erorilor.

Presupunem o funcţie f(x,y) de două variabile independente. Fie f, x şi 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 în serie Taylor; obţinem:

$$f(x,y) = \overline{f} + \Delta_{ij}f = f(\overline{x} + \Delta_i\ ,\ \overline{y_j} + \Delta_j) = \tag{7}$$
$$= f(\overline{x}\ ,\ \overline{y}) + \frac{\partial f}{\partial x}\Big|_{\overline{x},\overline{y}} \Delta_i + \frac{\partial f}{\partial y}\Big|_{\overline{x},\overline{y}} \Delta_j$$

neglijând termenii de ordin mai înalt.

Identificăm:

$$\overline{f} = f(\overline{x}\ ,\ \overline{y})$$
$$\Delta_{ij}f = \frac{\partial f}{\partial x}\Delta_i + \frac{\partial f}{\partial y}\Delta_j \tag{8}$$

Abaterea medie pătratică a funcţiei f este, prin definiţie:

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

$$= \frac{1}{N}\left[\left(\frac{\partial f}{\partial x}\right)^2 \sum_i (\Delta_i)^2 + \left(\frac{\partial f}{\partial y}\right)^2 \sum_j (\Delta_j)^2\right] \qquad (9)$$

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

conform definiţiei (4) şi a faptului că suma produselor $\Delta_i\Delta_j$ conţine termeni ce se vor anula. Generalizând, pentru o funcţie $f(x_1, x_2, \ldots, x_m)$ de m variabile x_i (i=1,m):

$$\sigma_f^2 = \sum_{k=1}^{m} \left(\frac{\partial f}{\partial x_k}\right)^2 \sigma_k^2 \qquad (10)$$

Relaţia (10) reprezintă **teorema de propagare a abaterilor standard** sau mai obişnuit teorema de propagare a erorilor.

Exemplu.

Presupunem că se fac N măsurători ale înălţimii h de la care cade liber un corp şi ale timpului de cădere şi că vrem să determinăm astfel acceleraţia gravitaţională

$$g = \frac{2h}{t^2} = f(h,t) \qquad (11)$$

şi abaterea standard a valorii obţinute.

EX74.PAS

```
program prop_er;
 {programul calculeaza valoarea medie}
 { si abaterea standard pentru f(x1,x2...xm)}
 {m<=20 folosind n, n<=100, masuratori pentru}
 {setul de variabile x1,x2...xm}
```

```
{ si teorema propagarii erorilor}
{aici este folosit pentru g=f(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
   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 {main program}
 write('introdu nr. de variabile,m=');readln(m);
 write('introdu numar seturi de masura,n=');
 readln(n);
 for i:=1 to m do
   begin
   writeln('introdu valorile x[',i,']');
   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:=
dfh(xmediu[1],xmediu[2])*dfh(xmediu[1],xmediu[2]);
```

```
sig2fm2:=
dft(xmediu[1],xmediu[2])*dft(xmediu[1],xmediu[2]);
  sigfm2:=sig1fm2*sigx2[1]+sig2fm2*sigx2[2];
write(' h=',xmediu[1]:10:2,' +/- ');
writeln(sqrt(sigx2[1]):10:2);
write(' t=',xmediu[2]:10:2,' +/- ');
writeln(sqrt(sigx2[2]):10:2);
write(' g=',fmediu:10:2,' +/- ');
writeln(sqrt(sigfm2):10:2);
 END.
```

3. Distribuţii

In 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:

$$\int_{-\infty}^{\infty} p(x)\,dx=1 \quad \textit{pentru x continuu}$$

$$\textit{sau} \qquad (12)$$

$$\sum_{i=1}^{n} p(x_i)=1 \quad \textit{pentru x discret}$$

n fiind numărul de valori posibile pentru x.

Parametrii importanţi ce caracterizează o distribuţie sunt:

valoarea medie μ definită ca

$$\mu = \int_{-\infty}^{\infty} x\, p(x)\, dx$$

sau (13)

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

și
abaterea pătratică medie (varianţa) definită ca

$$\sigma^2 = \int_{-\infty}^{\infty} (x-\mu)^2\, p(x)\, dx = \langle x^2 \rangle - \mu^2$$

cu

$$\langle x^2 \rangle = \int_{-\infty}^{\infty} x^2\, p(x)\, dx \qquad (14)$$

sau

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

i) Distribuţia normală sau **distribuţia Gauss** este distribuţia cea mai des folosită în măsurătorile experimentale. Forma ei este:

$$p(x) = \frac{1}{\sigma \sqrt{2\pi}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}} \qquad (15)$$

și conţine 2 **parametri independenţi**, μ și σ, (Figura 1) .

Ea are următoarele **proprietăţi**:
- este normată la unitate:

$$\int_{-\infty}^{\infty} p(x)\ dx = 1 \qquad (16)$$

- este simetrică faţă de dreapta x=μ în sensul că $P(x_1)=P(x_2)$ dacă $\mu - x_1 = x_2 - \mu$.
- admite un singur maxim, la $x = \mu$, $p(\mu)=1/(\sigma\sqrt{2\pi})$
- prin integrare numerică se poate verifica faptul că probabilitatea ca o valoare măsurată x_i să aparţină intervalului $(\mu - \sigma, \mu + \sigma)$ este:

$$r(\mu-\sigma \le x \le \mu+\sigma) = \int_{\mu-\sigma}^{\mu+\sigma} p(x)\ dx = 0.693 \qquad (17)$$

analog, pentru intervalele de 2σ *si* 3σ,

$$r(\mu-2\sigma \le x \le \mu+2\sigma) = 0.954$$

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

De unde vedem că pentru o mărime, ale cărei valori sunt distribuite normal, 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 aruncăm valorile ce nu aparţin intervalului $(\mu-3\sigma,\mu+3\sigma)$.

Exerciţiu: un program care dă valorile numerice de mai sus pentru integrale pe intervalele considerate folosind un set de valori pentru care calculaţi μ şi σ.

EX75.PAS

```
PROGRAM distr_Gauss;
uses crt;
type fct=FUNCTION(x:real):real;
const pi=3.14159;
VAR    x:array[1..100] of real;
var    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
   begin
    if i mod 2 = 0 then s:=s+2*f(a+i*h)
                   else s:=s+4*f(a+i*h);
   end;
 simpson:=s*h/3;
 end;   {end Simpson}
{$F+}
Function Gauss(x:real):real;
 begin
Gauss:=exp(-(x-xmediu)/2/sigx*(x-xmediu)/sigx);
 end;   {end Gauss}
{$F-}
BEGIN     {program principal}
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;
 WRITELN('xmediu=',xmediu);
 s:=0; for i:=1 to n do
       s:=s+(x[i]-xmediu)*(x[i]-xmediu);
 sigx:=sqrt(s/n); writeln('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
write('r(xmediu-',i,'*sigx<=x<=xmediu+',i);
writeln('*sigx)=',r[i]);
write(' r este probabilitatea ca o valoare x');
writeln(' sa cada in');
write(' intervalul de 1, 2 sau 3 abateri');
writeln(' standard');
writeln(' in jurul valorii medii');
   repeat until keypressed;
END.
```

Pentru seminar:
1. Fiind dat un set de valori x[1]..x[n], rezultate din măsurarea unei mărimi fizice, să se scrie programul care
- calculează valoarea medie şi abaterea standard şi le scrie
- găseşte şi elimină valorile ce se abat cu mai mult de 2 abateri standard de la valoarea medie şi scrie câte astfel de valori a găsit
- în caz că au fost găsite şi eliminate astfel de valori, reface, pe valorile rămase, calculul valorii medii şi abaterii standard şi le scrie; în caz contrar scrie că toate valorile date sunt în intervalul xmediu +/- 2*sigx.

2. Se dau 2 seturi de măsurători x1[1]..x1[n1] şi x2[1]..x2[n2]. Să se scrie programul care
- calculează valorile medii şi abaterile standard şi le scrie
- găseşte dacă cele 2 seturi sunt rezultatul măsurării aceleaşi mărimi fizice, adică dacă
x1mediu - 3*sigx1<= x2mediu <=x1mediu + 3*sigx1
dacă da, reface calculul valorii medii şi abaterii standard pentru setul n1 + n2 de valori şi scrie noile valori pentru xmediu şi sigx, dacă nu, scrie mesajul că cele două seturi de valori provin din măsurarea a 2 mărimi fizice diferite.

ii) 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:

$$p(k) = \frac{m^k}{k!} e^{-m} \qquad (18)$$

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

$$m=\frac{\sum_{i=0}^{k_{max}} n_i\, k_i}{\sum n_i} \qquad (19)$$

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

$$N=\sum_{i=0}^{k_{max}} n_i \qquad (20)$$

este numărul total de măsurători.

Proprietăţi ale distribuţiei Poisson:

- Condiţia de normare la unitate:

$$deoarece \sum_{k=0}^{\infty} \frac{m^k}{k!}=e^m \text{ , } rezulta \sum_{k=0}^{\infty} p(k)=1 \qquad (21)$$

- Valoarea medie:

$$\overline{k} \equiv \sum_{k=0}^{\infty} k\, p(k) = \sum_{k=0}^{\infty} k\, \frac{m^k}{k!} e^{-m} =$$

$$=m\sum_{k=0}^{\infty} \frac{m^{k-1}}{(k-1)!} e^{-m} \quad deci \quad \overline{k}=m \qquad (22)$$

- Abaterea standard:

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

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

$$rezulta \ ca \ \sigma_k = \sqrt{\overline{k^2} - \overline{k}^2} = \sqrt{m} \qquad (23)$$

- $p(0) = e^{-m} <> 0$ adică 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.

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:

$$p(m-1) = \frac{m^{m-1}}{(m-1)!}e^{-m} = \frac{m}{m}\frac{m^{m-1}}{(m-1)!}e^{-m} = \frac{m^m}{m!}e^{-m} = p(m) \qquad (24)$$

Distribuţia Poisson are un singur parametru independent: m.

EX76.PAS

```
PROGRAM distrib_Poisson;
Uses crt,graph;
{verifica distributia Poisson si reprezinta}
{histograma; valori k intre 0 si 10, valori nk}
{ 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;
  sknk1:=0;
  for i:=0 to kmax do
   sknk1:=sknk1+(k[i]-kmediu)*(k[i]-kmediu);
```

```
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: {reprezentare 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 hist.}
   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
     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((1+kmax)*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.
```

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 şi cum obţinem parametrii ei?

Din considerente statistice, pe care le veţi afla în anul II, se spune că dreapta care descrie cel mai bine punctele este cea care satisface condiţia:

$$S = \sum_{i=1}^{n} \frac{(y_i - a*x_i - b)^2}{\sigma_i^2} = \min \qquad (25)$$

Din

$$\frac{\partial S}{\partial a} = 0 \quad rezulta \quad a\sum_{i=1}^{n} \frac{x_i^2}{\sigma_i^2} + b\sum_{i=1}^{n} \frac{x_i}{\sigma_i^2} = \sum_{i=1}^{n} \frac{x_i y_i}{\sigma_i^2}$$

$$si\ din \qquad (26)$$

$$\frac{\partial S}{\partial b} = 0 \quad rezulta \quad a\sum_{i=1}^{n} \frac{x_i}{\sigma_i^2} + b\sum_{i=1}^{n} \frac{1}{\sigma_i^2} = \sum_{i=1}^{n} \frac{y_i}{\sigma_i^2}$$

un sistem de 2 ecuaţii cu 2 necunoscute: a şi b.

Notând $1/\sigma_i^2 = p_i$ = ponderea unei măsurători, sistemul se scrie:

$$\begin{aligned} a\sum_{i=1}^{n} p_i x_i^2 + b\sum_{i=1}^{n} p_i x_i &= \sum_{i=1}^{n} p_i x_i y_i \\ a\sum_{i=1}^{n} p_i x_i + b\sum_{i=1}^{n} p_i &= \sum_{i=1}^{n} p_i y_i \end{aligned} \qquad (27)$$

Soluţiile sunt:

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

$$b = \frac{\sum p_i x_i^2 \sum p_i y_i - \sum p_i x_i \sum p_i x_i y_i}{\sum p_i \sum p_i x_i^2 - \left(\sum p_i x_i\right)^2} \qquad (28)$$

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

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

$$b = \frac{\sum x_i^2 \sum y_i - \sum x_i \sum x_i y_i}{n\sum x_i^2 - \left(\sum x_i\right)^2} \qquad (29)$$

Pentru a găsi erorile cu care sunt determinaţi parametrii a şi b aplicăm teorema propagării erorilor (relaţia 10):

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

$$b = b(x,y) \Rightarrow \sigma_b^2 = \left(\frac{\partial b}{\partial x}\right)^2 \sigma_x^2 + \left(\frac{\partial b}{\partial y}\right)^2 \sigma_y^2 \qquad (30)$$

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

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

$$\sigma_b^2 = \sum_{i=1}^{n} \left(\frac{\partial b}{\partial y_i}\right)^2 \sigma_y^2 \qquad (31)$$

$$cu \quad \sigma_{y_1} = \sigma_{y_2} = \ldots = \sigma_{y_n} = \sigma_y$$

Cum

$$\left(\frac{\partial a}{\partial y_i}\right)^2 = \left[\frac{n\,x_i - \sum x_i}{n\sum x_i^2 - \left(\sum x_i\right)^2}\right]^2 \qquad (32)$$

rezultă:

$$\sigma_a = \sqrt{\frac{n}{n\sum x_i^2 - \left(\sum x_i\right)^2}}\ \sigma_y \qquad (33)$$

La fel:

$$\left(\frac{\partial b}{\partial y_i}\right)^2 = \left[\frac{\sum x_i^2 - x_i \sum x_i}{n\sum x_i^2 - \left(\sum x_i\right)^2}\right]^2 \qquad (34)$$

rezultă:

$$\sigma_b = \sqrt{\frac{\sum x_i^2}{n\sum x_i^2 - \left(\sum x_i\right)^2}}\ \sigma_y \qquad (35)$$

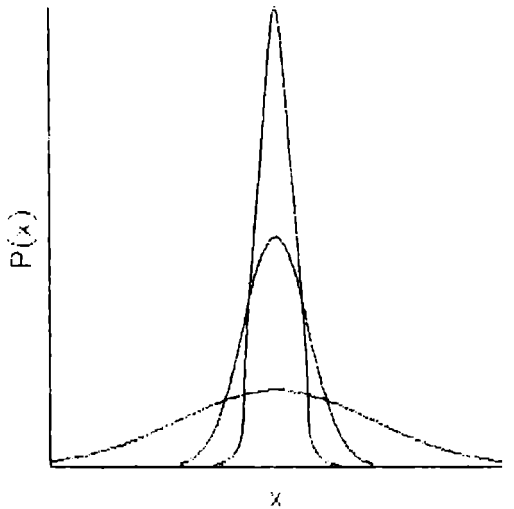


Fig.1 Distributii Gauss
pentru diferite abateri standard.

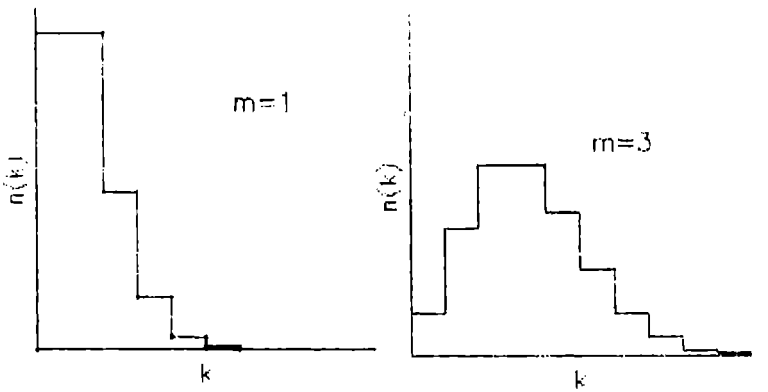


Fig.2 Histograme Poisson
pentru 2 valori m

EX77.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 num. de puncte(x,y) masurate');
  write(' n='); readln(n);
  writeln('introdu valorile x(i),y(i), i=1...n,');
  write('x(i) ordonate crescator adica');
  writeln(' x(1)<x(2)<...<x(n)');
  writeln(' cate o pereche x,y 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 verificarea cu 0}
  if num=0 then goto unu;
               {calcul parametrii dreptei}
  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');
                        {initializare mod grafic}
  line(5,5,5,300); line(5,300,600,300);
  for i:=1 to n do
    begin
    px:=ROUND(scx*x[i]);
    py:=300-ROUND(scy*y[]);
    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
```


Tiparul s-a efectuat sub c-da nr. 247/1996
la Tipografia Editurii Universităţii Bucureşti

ISBN - 973 - 575 - 083 - x

Lei 2680