Proceduri destinate gestiunii generatorului de sunet şi a intensităţii video

Procedura Sound

Procedura **Sound** asigură pornirea generatorului de sunet cu o frecvenţă dată. Frecvenţa reprezintă **numărul de oscilaţii generate într-o secundă**. Procedura se defineşte astfel **:**
Sound(Hz);
Hz – reprezintă frecvenţa sunetului exprimată în **Hertzi**.

Frecvenţa notelor muzicale

Octave	Do	Do#	Re	Re#	Mi	Fa	Fa#	Sol	Sol#	La	La#	Si
1	65	69	73	78	82	87	92	98	104	110	116	123
2	135	139	147	156	165	175	185	196	208	220	233	247
Do major	262	278	294	312	330	350	370	392	416	440	466	494
4	524	556	588	624	660	700	740	784	832	880	932	988
5	1048	1112	1176	1248	1320	1400	1480	1568	1664	1760	1864	1976
6	2096	2224	2352	2496	2640	2800	2960	3136	3328	3520	3728	3952
7	4192	4448	4704	4992	5280	5600	5920	6272	6656	7040	7456	7904

Procedura NoSound

Procedura **NoSound** decuplează generatorul de sunet. Dacă o procedură **Sound** nu este urmată, după procedura **Delay** de o procedură **NoSound**, atunci sunetul continuă, chiar dacă am părăsit mediul **Pascal**. Ea se defineşte prin **:**
NoSound;

Procedura Delay

Procedura **Delay** defineşte intervalul de timp destinat funcţionării generatorului de sunet. Folosită independent de procedurile **Sound** şi **NoSound**, stabileşte o pauză exprimată în **milisecunde**. O secundă este egală cu o mie de milisecunde. Deci 500 milisecunde reprezintă o jumătate de secundă. Combinate, cele trei proceduri pot produce un sunet cu o anumită durată de timp.

Program Prg_0012_Sound_Delay_NoSound;

```
Uses Crt;
Begin
TextBackGround(Blue);ClrScr;                {Fond albastru, Sterg ecran}
TextColor(Yellow);                          {Scris galben}
GoToXY(33,5);Write('Gama Do Major');
GoToXY(39,7);Write('Do');Sound(262);Delay(500); {Se  emit  sunete  cu }
GoToXY(39,8);Write('Re');Sound(294);Delay(500); { frecvenţa scrisa  în }
GoToXY(39,9);Write('Mi');Sound(330);Delay(500);  {paranteza procedurii }
GoToXY(39,10);Write('Fa');Sound(350);Delay(500); {Sound şi  cu  durata }
GoToXY(39,11);Write('Sol');Sound(392);Delay(500);{de  500  milisecunde }
GoToXY(39,12);Write('La');Sound(440);Delay(500); {scrisă în  paranteza }
GoToXY(39,13);Write('Si');Sound(494);Delay(500);        {procedurii  Delay    }
GoToXY(39,14);Write('Do');Sound(524);Delay(500);
GoToXY(34,16);Write(#7'Turbo Pascal'#7);       {Se emit 2 sunete datorită
                                                caracterului special #7 şi
                                                se afisează textul dintre
                                                apostroafe}
NoSound;                                        {Se opreste generatorul
                                                de sunet}
ReadLn;                                             {Se  aşteaptă  apăsarea
unei
                                                taste}
End.
```

Program Prg_0013_Afisare_precisa_a_numerelor;

```
Uses Crt;
Const
A=15;B=5;C=3;                   {Se asociază numerele intregi}
D=3.14;E=1.73;F=5.225;          {Se asociază numerele reale}

Begin                           {Începutul programului}
TextBackGround(Blue);ClrScr;     {Fond albastru, Şterg ecranul}
TextColor(Yellow);               {Culoarea scrisului galbenă}
Sound(294);Delay(100);    {      Se emit sunete}
Sound(330);Delay(100);    {      Se emit sunete}
Sound(350);Delay(100);    {      Se emit sunete}
Sound(392);Delay(100);    {      Se emit sunete}
Sound(440);Delay(100);    {      Se emit sunete}
Sound(494);Delay(100);    {      Se emit sunete}
Sound(524);Delay(100);    {      Se emit sunete}
NoSound;                        {Se opreste generatorul de sunet}

GoToXY(27,5);Write('Afişare precisă a numerelor');
GoToXY(27,6);Write('==========================');
```

```
GoToXY(35,8);Write(A*A:5);          {Se trimite cursorul in coloana 35}
GoToXY(35,9);Write(A*B:5);          {dupa care, datorita parametrului 5}
GoToXY(35,10);Write(A*A*B*C:5);     {din instructiunea 'Write', cursorul}
                                    {este deplasat 5 coloane spre dreapta}
                                  {şi apoi numărul este afişat de la}
                                  {dreapta spre stînga}


GoToXY(37,12);Write(D*E:7:3);       {Explicatia este ca mai sus, doar că }
GoToXY(37,13);Write(D*D*E:7:3);     {ne deplasam 7 coloane şi afisarea se}
GoToXY(37,14);Write(D*D*E*E*F*F:7:3);  {face cu 3 zecimale}
ReadKey;                            {Se aşteaptă apăsarea unei taste}
GoToXY(37,15);Write(D*D*E*E*F*F:7:3);  {Se tipăreşte încă odată ultimul rînd}
ReadKey;                            {Se aşteaptă apăsarea unei taste}
GoToXY(40,15);Write(',');                 {Se tipăreşte semnul virgulă peste
                                    punctul zecimal}
                                    {Se aşteaptă apăsarea unei taste}
ReadKey;
End.
```

Program Prg_0014_Proceduri_intensitate_video;

```
Uses Crt;

Begin
TextBackGround(Black);ClrScr;

TextBackGround(Blue);
TextColor(Yellow);
GoToXY(1,1);LowVideo;Write('LowVideo');
ReadKey;


TextBackGround(Blue);
TextColor(Yellow);
GoToXY(1,3);NormVideo;Write('NormVideo');
ReadKey;


TextBackGround(Blue);
TextColor(Yellow);
GoToXY(1,5);HighVideo;Write('HighVideo');
ReadKey;

TextBackGround(Black);ClrScr;
```

ReadKey**;**
End**.**

Tipuri de date

În **Turbo Pascal** există următoarele clase de tipuri **:**

- **Ordinal**
- **Real**
- **Mulţime**
- **Fişier**
- **Reper**
- **Şir de caractere**
- **Tablou**
- **Articol**
- **Procedură / funcţie**
- **Obiect**

Tipurile ordinale reprezintă mulţimi finite şi ordonate de valori. Se poate face referire la numărul de ordine al unei valori cu funcţia **Ord**, se poate specifica elementul succesor cu funcţia **Succ**, respectiv elementul predecesor cu funcţia **Pred** al unui element dat.

Tipurile ordinale sînt **:**

- **Tipuri logice**
- **Caracter**
- **De enumerare**
- **Tipuri întregi**
- **Tipuri interval**

Prin **tip simplu** se înţelege fie un **tip ordinal**, fie un **tip real.**

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

Tipuri logice

În **Turbo Pascal** sînt definite trei tipuri logice **: Boolean, WordBool** şi **LongBool**. Valorile de tip **Boolean** sînt reprezentate în memorie pe un octet, cele de tip **WordBool** pe doi octeţi (cuvînt), iar cele de tip **LongBool** pe patru octeţi (cuvînt dublu). Ultimele două tipuri au fost introduse în vederea asigurării compatibilităţii programelor **Pascal** cu programele destinate aplicaţiilor care se execută sub **Windows**.

Valorile tipului logic sînt desemnate de constantele predefinite **False** (Fals) şi **True** (Adevărat). În memorie constanta **False** se reprezintă cu **0** (zero), iar constanta **True** prin 1 (unu). **Tipul logic este ordinal**. Ordinul lui **False** este **0**,

iar ordinul lui **True** este **1**. Succesorul lui **False** este **True**, predecesorul lui **True** este **False** şi are loc relaţia **:**

ord(False)<ord(True)

Operaţiile care se pot face cu valorile tipului logic sînt **:**

- **And**

- conjuncţie logică (şi logic);
- operator binar;
- x **and** y, este **True** dacă şi numai dacă atît x cît şi y au valoarea **True**.

- **Or**

- disjuncţie logică (sau logic);
- operator binar;
- x **or** y este **False** dacă şi numai dacă atît x cît şi y au valoarea **False**.

- **Xor**

- *sau* exclusiv;
- operator binar;
- x **xor** y este **True** dacă şi numai dacă unul din operanzi este **True** şi celălalt este **False**.

- **Not**

- negaţie logică;
- operator unar;
- **not** x este **True** dacă şi numai dacă x este **False**.

Exemplu :

```
Program logica;

Var x,y,u,v:boolean;

Begin
      {...}
      x:=true;y:=false;
      u:=(not x) and y;
      v:=x or y;
      {...}
      u:= 2=1 {Valoarea lui u este False}
      v:= 2>1 {Valoarea lui v este True}
      {...}
End.
```

Tipul caracter

Valorile tipului **Char** (caracter) sînt cele **256** de caractere **ASCII** – set extins, numerotate de la **0** la **255**. Tipul **Char** este ordinal. În memorie se reprezintă pe un octet.

Valoarea acestui tip se poate scrie în diferite moduri **:**

- Caracter imprimabil cuprins între apostroafe, de exemplu **'a'**.
- Un număr de la **0** la **255**, precedat de caracterul # (simbol numeric). Numărul poate fi scris în **zecimal** (**#65** este codul caracterului **A**) sau în **hexazecimal** (**#$30** este codul cifrei zero).
- Un caracter precedat de caracterul **^**, pentru exprimarea caracterelor de control de la **0** la **31**, de exemplu **^A** (valoarea **ASCII 01**).

Funcţiile predefinite **Ord** şi **Chr** permit realizarea corespondenţei între setul de caractere **ASCII – extins** şi **numerele ordinale** ale caracterelor setului.

Astfel funcţia **Ord(c)** returnează rangul caracterului **c** în mulţimea ordonată a setului de caractere iar funcţia **Chr(i)** returnează valoarea caracterului de rangul **i**. Funcţiile **Ord** şi **Chr** sînt inverse şi sînt adevărate următoarele relaţii **:**

Chr(Ord(c))=c şi **Ord(Chr(i))=i**

Asupra valorilor de tip **Char** sînt permise operaţiile relaţionale. Fie **v1** şi **v2** două valori de tip **Char**, iar **r** una din relaţiile **<, <=, =, >=, >, <>**. Rezultatul operaţiei **: v1 r v2** este de tip logic. Acest rezultat este **True**, dacă şi numai dacă operaţia relaţională **:**

Ord(v1) r Ord(v2)

furnizează rezultatul **True.**

Program Prg_0015_Functiile_Chr_şi_Ord;

```
Uses Crt;
Const A=65;B=66;                {Constantele A şi B}

Begin
ClrScr;                         {Sterge ecranul}
WriteLn('Functia Chr');WriteLn; {Afişează textul dintre apostroafe}
WriteLn('65 = ',Chr(65));       {Afiseaza '65 = ' şi caracterul echivalent}
WriteLn('66 = ',Chr(66));       {Afişează '66 = ' şi caracterul echivalent}
WriteLn('67 = ',Chr(67));       {Afişează '67 = ' şi caracterul echivalent}

WriteLn;WriteLn('Functia Ord');  {Afişează textul dintre apostroafe}
WriteLn(Ord(Chr(65))<Ord(Chr(66)));  {Evaluează relaţia şi afişează True}
WriteLn(Ord(A)>Ord(B));          {Evaluează mărimea constantelor -False}
WriteLn(Ord(Chr(A))<Ord(Chr(B)));  {Evaluează mărimea ordinelor - True}
ReadKey;                         {Aşteaptă apăsarea unei taste}
End.
```

Programele **Prg_0016** afişează **setul de caractere al PC-ului, însoţit de codurile caracterelor** iar **Prg_0017** afişează o selecţie din **setul de caractere**, adică **caracterele româneşti** şi **caracterele pentru chenarul dublu**.

Caracterele cu codul **pînă la 31** şi semnul apostrof (‘) au fost introduse folosind afişarea cu #. Caracterele de la 32 la 255, inclusiv cele pentru chenar, au fost introduse prin apăsarea tastei Alt + codul caracterului. O altă modalitate de tipărire a datelor pe ecranul monitorului sau la imprimantă, este dată în programul **Prg_0018**.

Program Prg_0018_Afisare_cu_Chr;

```
Uses Crt;

Begin
TextBackGround(1);TextColor(14);ClrScr;
WriteLn(Chr(201),Chr(205),Chr(205),Chr(205),Chr(205),Chr(187));
WriteLn(Chr(186),Chr(32),Chr(65),Chr(66),Chr(32),Chr(186));
WriteLn(Chr(200),Chr(205),Chr(205),Chr(205),Chr(205),Chr(188));

ReadKey;
End.
```

În acest program s-a folosit afişarea prin coduri de control, folosind funcţia **Chr** care afişează caracterul al cărui cod se află între paranteze.

Tipuri întregi

Mulţimea valorilor tipurilor întregi este o submulţime a numerelor întregi. Există cinci tipuri de întregi **: integer, word, shortint, byte** şi **longint**. Domeniul de valori şi numărul de octeţi necesari pentru reprezentare depind de tipul utilizat.

Tip	Domeniu de valori		Octeţi
Integer	$-2^{15} \dots 2^{15}$	**- 32768 … 32767**	**2**
Word	$0 \dots 2^{16} - 1$	**0 … 65535**	**2**
Shortint	$-2^{7} \dots 2^{7} - 1$	**- 128 … 127**	**1**
Byte	$0 \dots 2^{8} - 1$	**0 … 255**	**1**
Longint	$-2^{31} \dots 2^{31} - 1$	**-2.147.483.648 … 2.147.483.647**	**4**

Prin constanta predefinită **MaxInt** se înţelege valoarea întreagă **32767**, adică **valoarea maximă** reprezentată pe **16 biţi**, cu semn. Tipul întreg este ordinal. **Succesorul** lui **x** este **x+1**. **Predecesorul** lui **x** este **x-1**. Ordinul întregului **x** este egal cu **x**. Operaţiile care se pot face cu valorile tipului întreg, sînt următoarele **:**

- **+** **adunare**
- **-** **scădere** sau schimbare de semn
- **Div** **împărţire întreagă**
  A Div B returnează cîtul împărţirii întregi între **A** şi **B**

Rezultatul este de tip întreg.

- **Mod** **restul împărţirii întregi**.
  A Mod B returnează restul împărţirii întregi a lui **A** cu **B**
  Rezultatul este de tip întreg.
- **/** **împărţire reală**
  A / B returnează cîtul împărţirii lui **A** la **B**
  Rezultatul este de tip real.
- **And** **şi aritmetic**
  A And B returnează o valoare întreagă.
  Rezultatul se obţine prin efectuarea conjuncţiei între biţii corespunzători ai lui **A** cu **B**.
- **Or** **sau aritmetic**
  A or B returnează o valoare întreagă
  Rezultatul se obţine prin efectuarea disjuncţiei între biţii corespunzători ai lui **A** şi **B**.
- **Xor** **Sau exclusiv aritmetic**
  A Xor B returnează o valoare întreagă
  Rezultatul se obţine prin efectuarea operaţiei **sau exclusiv** între biţii corespunzători ai lui **A** şi **B**.
- **Shl** **deplasare la stînga**
  A Shl B returnează o valoare întreagă care se obţine prin deplasarea la stînga, a lui **A** de **B** ori.
- **Shr** **deplasare la dreapta**
  A Shr B returnează o valoare întreagă care se obţine prin deplasarea la dreapta, a lui **A** de **B** ori.
- **Not** **negaţie aritmetică**
  Operator unar cu rezultat întreg
  Not A se obţine prin efectuarea operaţiei de negare asupra fiecărui bit al lui **A**.

Asupra valorilor de tip întreg sînt permise următoarele operaţii relaţionale **:**

- **<=** mai mic sau egal cu
- **<** mai mic
- **=** egal
- **>** mai mare
- **>=** mai mare sau egal cu
- **<>** diferit

Rezultatul operaţiei **v1 r v2** este de tip logic, unde **r** este o operaţie relaţională, iar **v1** şi **v2** sînt două valori întregi. Acest rezultat este adevărat, dacă şi numai dacă **Ord(v1) r Ord(v2)** este **True**.

Regulile referitoare la operaţiile aritmetice care utilizează operatori întregi sînt următoarele **:**

1. Tipul unei constante întregi se deduce din valoarea ei. Se alege acel tip predefinit, care are domeniul cel mai restrîns, dar care include valoarea respectivă.

2. La un operator binar, adică un operator care are doi operanzi, mai întîi operatorii sînt convertiţi în tipul lor comun şi apoi se efectuează operaţia. Prin tipul comun se înţelege acel tip predefinit, care are domeniul cel mai restrîns dar care include toate valorile posibile ale celor două tipuri. De exemplu, tipul comun al unui **integer** şi al unui **byte** este **integer**, tipul comun al unui **integer** şi al unui **word** este **longint**. Operaţia se efectuează cu precizia tipului comun, iar rezultatul este de tip comun.
3. Expresia din membrul drept al unei instrucţiuni de atribuire este evaluată independent de dimensiunea sau tipul variabilei din membrul stîng.
4. Un operand de tip **ShortInt** este convertit într-un tip **string** intermediar înaintea efectuării oricărei operaţii aritmetice.
5. O valoare de tip întreg poate fi convertită explicit într-un alt întreg, folosind conversia de tip.

Program Prg_0019_Operaţii_aritmetice;

```
Uses Crt;
Var
A,B:LongInt;
C,D:Real;

Begin
TextBackGround(1);TextColor(14);ClrScr;{Fond albastru,scris galben,şterg ecran}
GoToXY(30,2);Write('Operaţii aritmetice');
GoToXY(1,23);Write(' Introduceţi A > B');
GoToXY(1,24);
Write(' Introduceţi valoarea lui A ');
ReadLn(A);                          {Se citeşte valoarea lui A}
GoToXY(1,5);Write(' Valoarea lui A este ');
GoToXY(25,5);Write(A:5);             {Se afişează valoarea lui A}
GoToXY(1,24);ClrEol;                {Se şterg caracterele pînă la sfîrşitul liniei}
Write(' Introduceţi valoarea lui B ');
ReadLn(B);                          {Se citeşte valoarea lui B}
GoToXY(1,23);ClrEol;                {Se şterg caracterele pînă la sfîrşitul liniei}
GoToXY(1,24);ClrEol;                {Se şterg caracterele pînă la sfîrşitul liniei}
GoToXY(1,6);Write(' Valoarea lui B este ');
GoToXY(25,6);Write(B:5);             {Se afişează valoarea lui B}

GoToXY(1,9);Write(' Adunarea    (A + B)');
GoToXY(1,10);Write(' A + B = ',A,' + ',B,' = ',A+B);

GoToXY(1,12);Write(' Scăderea    (A - B)');
GoToXY(1,13);Write(' A - B = ',A,' - ',B,' = ',A-B);

GoToXY(1,15);Write(' Înmulţirea  (A * B)');
GoToXY(1,16);Write(' A * B = ',A,' * ',B,' = ',A*B);
```

```
GoToXY(1,18);Write(' Împărţirea întreagă cu rest (A Div B şi A Mod B)');
GoToXY(1,19);Write(' A Div B = ',A,' Div ',B,' = ',A Div B,' Rest ',A Mod B);

GoToXY(1,21);Write(' Împărţirea reală cu zecimale (A / B)');
C:=A;D:=B;
GoToXY(1,22);Write(' A / B = ',A,' / ',B,' = ',C/D:5:3);

ReadKey;
End.
```

Program Prg_0020_Operaţii_logice_pentru_numere_întregi;

```
Uses Crt;

Var
i,j,k,l,m,n,o,sl,sr:integer;

Begin
ClrScr;
WriteLn(' Operaţii logice pentru numere întregi');
I:=291;
j:=255;
WriteLn(' i = ',i,' j = ',j);
k:=i And j;
WriteLn(' And = ',k);
l:=i Or j;
WriteLn(' Or  = ',l);
m:=i Xor j;
WriteLn(' Xor = ',m);
n:=52;
o:=Not n;
WriteLn(' Not = ',o);
sr:=3 shr 4;
WriteLn(' Shr = ',sr);
sl:=3 shl 4;
WriteLn(' Shl = ',sl);

ReadKey;
End.
```

Tipuri reale

Mulţimea valorilor tipului real este o submulţime a numerelor reale. În timp ce rezultatele operaţiilor asupra unor valori întregi reproduc exact rezultatele,

operaţiile asupra valorilor reale sînt în general aproximative datorită erorilor de rotunjire. Se foloseşte un număr finit de cifre pentru reprezentarea unui număr real.

Există cinci tipuri reale **: real, single, double, extended** şi **comp.** Ultimele patru tipuri pot fi utilizate doar atunci cînd calculatorul este prevăzut cu coprocesorul matematic **80x87**. Domeniul de valori, numărul de octeţi necesari pentru reprezentare şi numărul de cifre semnificative depind de tipul real utilizat, conform tabelului de mai jos.

Tip	Domeniul de valori	Octeţi	Cifre semnificative
Real	**2.9E-39 … 1.7E38**	**6**	**1 – 12**
Single	**1.4E-45 … 3.4E38**	**4**	**8 – 8**
Double	**5.0E-324 … 1.7E308**	**8**	**15 – 16**
Extended	**3.3E-4932 … 1.1E4932**	**10**	**19 – 20**
Comp	**-2E63 + 1 … 2E63 – 1**	**8**	**19 – 20**

La primele patru tipuri au fost specificate doar intervalele numerelor pozitive din domeniul de valori. Intervalele numerelor negative sînt simetricele acestora. Deşi tipul **Comp** are numai valoriîntregi în intervalul **$[-2^{63} + 1,2^{63} - 1]$**, aproximativ **[–9.2E18 … 9.2E18]**, în calcule aceşti întregi vor fi consideraţi ca şi cum ar fi numere reale fără zecimale.

Tipul real este singurul tip simplu care **nu este de tip ordinal**. Astfel, valorile reale nu au numere ordinale, deci o valoare reală nu are nici succesor, nici predecesor.

Pentru generarea operaţiilor de tip real, compilatorul foloseşte două modele, care pot fi selectate din meniul de opţiuni, comanda opţiunii de compilare, grupa prelucrare numerică.

Cu comutatorul **8087 / 80287** nepoziţionat (sau cu directiva **{$N–}**, valoare implicită), toate operaţiile cu numere reale sînt efectuate prin apelarea unor subprograme de bibliotecă. Datorită vitezei şi lungimii codului, în acest model este admis numai **tipul real**, astfel folosirea tipurilor **single, double, extended, comp** va genera o eroare.

Cu comutatorul **8087 / 80287** poziţionat (sau cu directiva **{$N+}**), codul generat va conţine toate instrucţiunile coprocesorului numeric **80x87**. În acest caz sînt permise toate cele cinci tipuri reale, dar prezenţa coprocesorului **80x87** este obligatorie atît la compilare, cît şi la execuţie.

Cu comutatorul **Emulation** poziţionat (sau cu directiva **{$E+}**), este posibilă emularea unui coprocesor numeric **80x87** inexistent. Astfel, dacă sînt folosite simultan directivele **{$N+}** şi **{$E+}**, fişierul **.EXE** creat poate fi rulat pe orice calculator, indiferent de faptul dacă există sau nu un coprocesor numeric **80x87**. Dacă coprocesorul este prezent, atunci programul îl va utiliza efectiv, în caz contrar acest coprocesor va fi emulat, adică simulat pe cale soft.

Datorită faptului că **tipul real nu este ordinal**, variabilele reale nu pot fi folosite pentru a desemna **:**

- Un indice de tablou
- Un contor de ciclu pentru instrucţiunea **For**
- Un tip de bază pentru tipul **Set** (mulţime)
- O limită pentru tipul interval
- Un selector într-o instrucţiune **Case**.

Operaţiile care se pot face cu valorile reale sînt **: +** (adunare), **–** (scădere), ***** (înmulţire), **/** (împărţire). Rezultatul unei astfel de operaţii este de tip real, chiar dacă una din valori este de tip întreg.

Pentru valorile reale sînt permise operaţiile relaţionale **<=, <, =, >, >=** şi **<>**. Rezultatul unei astfel de operaţii este de tip logic, cu valorile **True** sau **False**.

Tipul enumerare

Tipul enumerare defineşte o mulţime ordonată de valori **:** se enumeră un şir de identificatori care desemnează valorile posibile. Primul identificator desemnează cea mai mică valoare, cu numărul de ordine **zero**. Ceilalţi identificatori desemnează succesorul valorii specificate de către identificatorul precedent. Identificatorii apar în ordinea crescătoare a valorii lor.

Tipul enumerare se defineşte în secţiunea **Type** astfel **:**

Type nume_tip = (identif,identif, … , identif);

Exemplu :

```
Type sex=(bărbat, femeie);
     Studii=(elementare, medii, superioare);
```

Variabilele de tip enumerare sînt declarate în secţiunea **var**. Ele pot lua una din valorile înşirate în lista de enumerare.

```
Var  s:sex;
     Pregătire:studii;
     Limba:(engleza,franceza);
```

Aici variabila **s** poate să ia una din valorile **bărbat** sau **femeie**, variabila **pregătire** poate sa ia una din valorile **superioare, medii** sau **elementare**.

Tipul lgic este de fapt un tip de enumerare **:**

Type boolean=(False,True);

Operaţiile care se pot face cu valorile unui tip de enumerare sînt următoarele **:**

- Atribuirea

  S:=bărbat; studii:=superioare;
  Limba:=germana;

- Determinarea numărului de ordine
  Se face cu funcţia **Ord**. De exemplu **Ord(bărbat)** returnează valoarea **0**, **Ord(medii)** returnează valoarea 1.

- Determinarea succesorului se face cu funcţia **Succ** sau a predecesorului se face cu funcţia **Pred**. De exemplu, prin instrucţiunea de atribuire **:**
  Limba:=Succ(engleza);
  Variabila **limba** va avea valoarea **franceza**, iar prin instrucţiunea **:**
  Pregătire:=Pred(medii);
  Variabila **pregătire** va avea valoarea **elemetare**.

Încercarea determinării **succesorului** ultimului element din listă sau a **predecesorului** primului element din listă, constituie o eroare.

- Comparaţia **(<, <=, =, >=, >, <>)**. Două valori **v1** şi **v2** sînt într-una din relaţiile anterioare dacă **Ord(v1)** şi **Ord(v2)** sînt în aceiaşi relaţie. De exemplu, comparaţia **"elementare < superioare"** furnizează rezultatul **True**, deoarece **Ord(elementare)**=0 este mai mic decît **Ord(superioare)=2.**

Tipul interval

Fiind dat un tip ordinal, din acest tip se poate genera un nou tip, numit **tipul interval**. Definiţia unui interval indică valoarea constantă cea mai mică şi cea mai mare din interval (în sensul numărului de ordine) şi cuprinde toate valorile dintre ele. Sintaxa unui **tip interval** este **:**

Type nume_tip=valoarea_minimă..valoarea_maximă;

Se subliniază faptul că nu este permisă definirea unui interval al tipului real, deoarece acesta nu este de tip ordinal. Valoarea minimă trebuie să fie mai mică sau egală cu valoarea maximă.

Exemplu :

```
Type
     Indice=1..10;                 {interval de integer}
     Litera='A'..'Z';              {interval de char}
     Zile=(Lu,Ma,Mi,Jo,Vi,Sî,Du);  {tip de enumerare}
     Zile_lucrat=Lu..Vi            {interval de tip enumerare}
Var
     I:indice;                     {valori posibile : 1, 2, …, 10}
     L:litera;                     {valori posibile : 'A','B', …,'Z'}
     Z:zile_lucrat;                {valori posibile : Lu,Ma, …, Vi}
```

O variabilă de tip interval moşteneşte proprietăţile variabilelor tipului de bază, dar valorile variabilei trebuie să fie numai din intervalul specificat. Dacă este validată opţiunea **Range checking** (meniul de opţiuni, comanda de opţiuni de compilare, grupa de erori de execuţie), sau dacă este prezentă directiva **{$R+}**, atunci în execuţia programului se va verifica apartenenţa valorii unei variabile de **tip interval**, la intervalul desemnat. În caz de neapartenenţă este semnalată o eroare de execuţie şi programul se opreşte. Implicit nu se efectuează nici o verificare, deoarece codul generat în acest caz, este mult mai scurt.

Program Prg_0021_Interval;

```
Uses Crt;

Type cifra=0..9;                    {Intervalul în care se pot lua valori}
Var C1,C2,C3:cifra;                 {Valorile posibile pt.C1, C2 si C3
                                        sînt 0,1, ...,9}
Begin
ClrScr;
{$R+}                               {Se activează directiva de verificare}
C1:=5;                              {Valoare validă}
C2:=C1+3;                               {Valoare validă}
{$R-}                               {Se dezactivează directiva de verificare}
C3:=15;                             {Valoare invalidă,dar nu se semnalează
                                        eroare}
{$R+}                               {Se activează directiva de verificare}
C3:=15;                             {Valoare invalidă,dar se semnalează
                                        eroare}

End.
```

Tipul mulţime

Un **tip mulţime (Set)** se defineşte în raport cu un tip de bază, care trebuie să fie un tip ordinal. Dîndu-se un asemenea tip de bază, valorile posibile ale tipului **Set** sînt formate din mulţimea tuturor submulţimilor posibile ale tipului de bază, inclusiv mulţimea vidă.

Tipul mulţime se defineşte astfel **:**

Type nume_tip=Set of Tip_de_bază;

unde **:**

Tip_de_bază este tip ordinal (**char, interval, enumerare, logic**).

Cu toate că tipurile întregi sînt ordinale, nu este permis decît tipul **set of byte**. Dacă tipul de bază are **'N'** valori, tipul mulţime va avea 2^N valori, cu restricţia ca **n<=256**.

Exemplu :

```
Type cifre=5..7; {tip interval}
     Mult=set of cifre;
Var m:mult;
```

Variabila **'m'** poate să aibe **8** valori **: [5], [6], [7], [5,6], [5,7], [6,7], [5,6,7]** şi **[]**, ultima valoare reprezentînd mulţimea vidă.

O valoare tip mulţime poate fi specificată printr-un **constructor** (generator) **de mulţime**. Un constructor conţine specificarea elementelor separate prin virgule şi închise între paranteze pătrate.

[element, element, …, element]

Un element de tip mulţime poate să fie **:**

- O valoare specificată
- Un interval de forma **:**

Inf..sup

Unde **:**

Valorile **inf** şi **sup** precizează valorile limitelor inferioare şi superioare.

Atît elementul, cît şi limitele de interval pot fi expresii. Construcţia **[]** reprezentînd mulţimea vidă. Dacă **sup<inf**, atunci nu se generează nici un element.

Exemplu :

```
Program tipmult;
Type octet=0..255; {tip interval}
     Numar=set of octet;
     Cuvint=set of char;
     Culoare=(alb,gri,negru); {tip enumerare}
     Nuanta=set of culoare;
Var n:numar;
     c:cuvint;
     a:nuanta;
     i:integer;

Begin
     N:=[2..4,8,10..12];
     {elementele din constructor : 2, 3, 4, 8, 10, 11, 12]
     i:=10;
```

```
    N:=[i-1..i+1, 2*i,30];
    {elemente : 9, 10, 11, 20, 30}
    c:=['A'..'C','K','S'];
    {elemente : 'A','B','C','K','S'}
    a:=[alb,gri];
End.
```

Dacă tipul de bază are **'n'** valor, o variabilă tip mulţime corespunzătoare tipului de bază, va fi reprezentată în memorie pe **'n'** biţi, depuşi într-o zonă de memorie contiguă (continuă) de **:**

- **(n div 8) +1** octeţi, dacă **'n'** nu este divizibil cu 8;
- **(n div 8)** octeţi, dacă **'n'** este divizibil cu 8.

De exemplu, o variabilă de tip **'set of char'** va fi reprezentat pe **256 div 8**, adică pe 32 octeţi.

Operaţii cu mulţimi

Operaţiile care se pot efectua cu valorile tip mulţime sînt **:**

- **+** 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 **c** este în **a** şi **c** nu este în **b**
- ***** intersecţie
  o valoare de tip ordinal
  c este în **a*b**, dacă **c** este în **a** şi în **b**

Relaţiile referitoare la mulţimi

Dacă **a** şi **b** sînt operanzi tip mulţime, atunci relaţia **:**

- **a = b**, este adevărată dacă **a** şi **b** conţin aceleaşi elemente, altfel **a<>b**
- **a<=b**, este adevărată dacă fiecare element al lui **a** este de asemenea un element al lui **b**
- **a>=b**, este adevărată dacă fiecare element al lui **b** este de asemenea un element al lui **a**
- **x in a**, este adevărată, dacă primul operand este elementul operandului al doilea (apartenenţă). Primul operand este de tip ordinal **t**, al doilea operand este de tip mulţime, al cărui tip de bază este compatibil cu tipul **t**.

În operaţiile şi relaţiile de mai sus, **a** şi **b** trebuie să fie tipuri mulţime compatibile. Dacă se notează cu **elmin** cea mai mică valoare ordinală a rezultatului unei operaţii cu mulţimi, iar cu **elmax** cea mai mare valoare ordinală a operaţiei, tipul rezultatului va fi **set of elmin..elmax** .

Constructorii pot fi folosiţi pentru scrierea mai comodă a unor condiţii. De exemplu, dacă **ch** este o variabilă tip caracter, condiţia **:**

If (ch='T') or (ch='U') or (ch='R') or (ch='B') or (ch='O') then {…}

poate fi exprimată prin **:**

if ch in ['T','U','R','B','O'] then {…}

iar condiţia **:**

if (ch>='0') and (ch<='9') then {…}

poate fi exprimată prin **:**

if ch in ['0'..'9'] then {…}

Exemplu : Verificarea operaţiilor cu mulţimi

```
Program operatiimultimi;
{Verificarea operatiilor cu multimi}
type
      multime=set of 1..10;
var
      a,b,int,reun,dif:multime;
      i:integer;

Begin
      a:=[1..3,7,9,10];
      WriteLn('Prima multime :');
      For i:=1 to 10 do
            If i in a then write(I:3);
      WriteLn;
      b:=[4..6,8..10];
      WriteLn('A doua multime :');
      For i:=1 to 10 do
            If i in b then write(i:3);
      WriteLn;
      Int:=a*b;   {9,10}
      Reun:=a+b;   {1..10}
      Dif:=a-b;   {1,2,3,7}
      WriteLn('Intersectie :');
      For i:=1 to 10 do
```

```
        If i in int then write(i:3);
    WriteLn;
    WriteLn('Reuniune :');
    For i:=1 to 10 do
        If i in reun then write(i:3);
    WriteLn;
    WriteLn('Diferenta :');
    For i:=1 to 10 do
        If i in dif then write(i:3);
    WriteLn;
End.
```

Tipuri pentru şir de caractere

În **Turbo Pascal** sînt definite două categorii de şiruri de caractere **:**

- Cele **"clasice"**
- Cele **"cu terminaţia nulă"**

Valoarea unei variabile de **tip şir de caractere** este formată dintr-o succesiune de caractere. Şirurile clasice au lungime maximă de 255 de caractere, iar lungimea unui astfel de şir este specificată într-un octet care precede şirul. Şirurile cu terminaţia nulă au **lungimea maximă** de **65535 octeţi (64Ko)**. La această categorie de şiruri, primul octet nu mai memorează lungimea şirului, ci un caracter nul **#0** semnalează sfîrşitul şirului. Ambele tipuri sînt considerate ca fiind tipuri compuse.

Tipul şir de caractere **"clasic"** se specifică prin construcţia **string[lungime]** sau numai prin cuvîntul cheie **string**. În primul caz **'lungime'** reprezintă lungimea maximă a şirului de caractere, avînd valori de la **1** la **255**. Un tip şir de caractere, fără specificarea atributului de lungime reprezintă un şir de lungime maximă, implicit egală cu **255**. Variabilele de tip **string[lungime]** pot avea ca valori orice succesiune de caractere a cărei lungime, nu depăşeşte lungimea declarată. Valoarea actuală a atributului de lungime, este returnată de funcţia standard **'Length'**.

Variabilele de tip şir de caractere **"clasice"** sînt memorate în locaţii succesive de memorie, pe **"Lungime+1"** octeţi, unde octetul de început conţine lungimea actuală a şirului. Această valoare poate fi modificată de utilizator, printr-o instrucţiune de atribuire de forma **:**

```
Sir[0]:=#nr;
```

sau

```
Sir[0]:=Chr(Ord(nr));
```

unde **:**

nr – este cuprins între **0** şi lungimea maximă admisă. Dacă **nr** are valoarea zero, atunci şirul este considerat vid.

O variabilă de tip şir **"clasic"** poate fi folosită în totalitatea ei, fie parţial, prin referirea unui caracter din şir. În primul caz referirea se face numai prin numele variabilei. În cel de-al doilea caz trebuie specificată între paranteze pătrate, poziţia caracterului din şir, printr-o construcţie de forma **" [expresie] "**, unde **expresia** trebuie să aibe o valoare întreagă în intervalul cuprins între **0** şi lungimea maximă declarată a şirului.

Asupra şirurilor de caractere se poate efectua operaţia de concatenare care se notează cu semnul plus (**+**). Dacă **s** şi **t** sînt doi operanzi de tip şir de caractere sau de tip **char**, rezultatul concatenării **s+t** este compatibil cu orice tip de şir de caractere, dar nu şi cu tipul **Char**. Dacă şirul rezultat depăşeşte 255 de caractere, şirul se trunchiază după caracterul al 255-lea.

Operatorii relaţionali **=, <>, <, >, <=** şi **>=** compară şiruri de caractere, în conformitate cu ordonarea setului extins de caractere ASCII. Deoarece toate şirurile de caractere sînt compatibile, pot fi comparate două valori arbitrare de tip şir.

O valoare de tip caracter este compatibilă cu o valoare de tip şir de caractere. Cînd aceste valori sînt comparate, valoarea de tip caracter este considerată ca şi cum ar fi un şir de lungime **1** (unu).

Program Prg_0022_Concatenare;

```
Uses Crt;

Var
S1:String[10];                  {Şirul S1 are lungimea maximă = 10}
S2:String[20];                  {Şirul S2 are lungimea maximă = 20}
L:Integer;                      {Lungimea şirului este un număr
întreg}

Begin
   ClrScr;
   S1:='Borland';               {Şirul S1 este 'Borland'}
   S2:=S1+' '+'Pascal';         {Şirul S2 se obţine prin adăugarea
                                unui spaţiu şi a şirului 'Pascal'}
   L:=Length(S2);               {Variabila L este egală cu lungimea
                                şirului S2}
   WriteLn(S2);                 {Se tipăreşte şirul S2}
   WriteLn('Lungime actuala = ',L);  {Se afişează lungimea şirului}

ReadKey;
End.
```

Introducerea **tipului şir de caractere cu terminaţia nulă** permite folosirea şirurilor de lungime mai mare ca 255. Primul octet nu mai memorează lungimea maximă actuală, ci conţine efectiv primul caracter din şir. Sfîrşitul şirului este semnalat cu caracterul **#0**. Lungimea maximă este **65535**. Conversia între şirurile clasice şi cele cu terminaţia nulă este realizată cu ajutorul funcţiilor **StrPCopy** şi **StrPas**. Aceste funcţii, precum şi toate acele funcţii care prelucrează şirurile cu terminaţia nulă, sînt depuse în unit-ul **Strings**.

Şirurile cu terminaţia nulă au următoarea declaraţie de tip **:**

Array[0..n] of char

Unde **'n'** desemnează numărul caracterelor din şir, deci fără aracterul de sfîrşit **#0**. Şirurile cu terminaţia nulă sînt utilizate sub forma variabilelor de tip reper. Pentru acest scop a fost introdus un tip predefinit **PChar** astfel **:**

Type Pchar=^char;

Acest tip desemnează un **tip reper**, care reperează un şir de caractere cu terminaţia nulă. Tipul **PChar** este compatibil din punct de vedere al atribuirii cu un şir de caractere clasice. O variabilă de acest tip poate să primească şi o valoare tip reper cu funcţia **Addr** sau @.

La variabilele de tip şir de caractere de terminaţie nulă, la fel ca la şirurile clasice şi la tabele, se pot folosi indici. Astfel **s[i]** desemnează un reper la caracterul al **i-**lea al şirului.

La transmisia de parametri se permite ca parametrului formal de tip şir clasic să-i corespundă un parametru actual tip şir cu terminaţia nulă.

Tipul tablou

Tipul tablou este un tip compus, care constă dintr-un număr fixat de componente, fiecare componentă avînd acelaşi tip. La definirea tipului tablou trebuie precizat atît tipul comun al componentelor, cît şi tipul indicilor, care stabileşte numărul componentelor tabloului.

Tipul tablou se defineşte printr-o construcţie de forma **:**

Type nume_tip=array[T1] of T2;

Unde **:**

T1 – este tipul indicelui, care trebuie să fie ordinal
T2 – este tipul componentelor şi care poate fi un tip oarecare.

T1 fiind ordinal, există un număr finit de valori, deci şi de componente.

Exemplu :

Type t=array[1..10] of integer;
Var a,b:t;

Fiecare componentă a unei variabile de tip tablou poate fi specificată explicit, prin numele variabilei urmat de indice, încadrat de paranteze pătrate, de exemplu **: A[7], B[9]**.

Fiind date două variabile de tip tablou, de acelaşi tip, numele variabilelor pot apărea într-o instrucţiune de atribuire.

Exemplu :

A:=B;

Această atribuire înseamnă copierea tuturor componentelor din membrul drept în membrul stîng, adică instrucţiunea precedentă este echivalentă cu ciclul **:**

Var i:integer;
{…}
For i:=Liminf to Limsup do a[i]:=b[i];

Unde **:**

Liminf şi **Limsup**, desemnează limita inferioară şi limita superioară a indicilor. Tablourile furnizează un mijloc de a grupa sub acelaşi nume mai multe variabile cu caracteristici identice.

Pentru specificarea tipului indicelui **T1** se folosesc de regulă intervale ale tipului întreg. Deoarece **T2** poate fi de orice tip, în particular poate fi tot un tip tablou. Astfel devine posibilă definirea tipului tablou multidimensional. Astfel **:**

Array[T1] of Array[T2] of T3

Reprezintă un tip tablou bidimensional (matrice). Accesul la o componentă oarecare, a unei variabile **tab** de acest tip, se realizează cu o construcţie de forma **:**

Tab[ind1][ind2]

Unde **:**

Ind1 – este o expresie de tip **T1**
Ind2 – este o expresie de tip **T2**

Această construcţie selectează elementul din linia **ind1** şi din coloana **ind2** a matricei **tab**. Se poate folosi şi o scriere simplificată **:** tipul poate fi scris sub forma **:**

Array [T1,T2] of T3;

Iar o referinţă sub frma **:**

Tab[Ind1,Ind2];

Exemplu :

```
Program Tablouri;
Type T1:Array[1..5] of Integer;
     T2:Array[0..3,0..5] of Real;
     T3:Array['a'..'c',2..5] of Char;
     T4:Array[Char,Boolean] of Char;
Var  V1:T1; {5 componente de tip întreg dintr-un vector}
     V2:T2; {4 x 6 = 24 componente de tip Real}
     V3:T3; {3 x 4 = 12 componente de tip Char}
     V4:T4; {256 x 2 = 512 componente de tip Char}
```

Se pot citi valorile de la tastatură sau se pot atribui în cadrul programului.

Exemplu :

```
V1[4]:=25;V1[5]:=V1[4]x2; {Vector}
V2[2,3]:=1.25
```

De la tastatură se citeşte cu **Read** (**Read(V2[2,3]);**)

Componentele unei variabile de tip tablou sînt memorate în locaţii consecutive de memorie, în stil linie după linie. Spaţiul de memorie ocupat se obţine prin înmulţirea lungimii în octeţi a unei componente cu numărul componentelor.

Program Prg_0023_MatriceBiDimensionala;

```
Uses Crt;

Type T1=Array[1..2,1..3] of Integer;

Var  V1:T1; {2 x 3 = 6 componente de tip intreg}

Begin
TextBackGround(1);TextColor(14);ClrScr;

GoToXY(29,3);Write('Matrice bidimensionala');
GoToXY(35,5);Write('Coloanele');
GoToXy(21,7);Write('ł              ł              ł');
GoToXy(21,8);Write('ł              ł              ł');
GoToXy(21,9);Write(' 1              2              3');
GoToXY(11,10);Write('╔═══════════════╦═══════════════╦═══
═══╗ ');
```

```
GoToXY(11,11);Write('ş V1[1,1] =       ş V1[1,2] =       ş V1[1,3] =       ş');
GoToXY(11,12);Write('╠═══════════════╬═══════════════╬═══════════════╣');
GoToXY(11,13);Write('ş V1[2,1] =       ş V1[2,2] =       ş V1[2,3] =       ş');
GoToXY(11,14);Write('╚═══════════════╬═══════════════╬═══════════════╝');
GoToXY(2,11);Write('Linia 1 >');
GoToXY(2,13);Write('Linia 2 >');

GoToXY(2,23);Write('Linia 1 Coloana 1 (V1[1,1]) = ');Read(V1[1,1]);
GoToXY(2,23);ClrEol;
GoToXY(25,11);Write(V1[1,1]:4);

GoToXY(2,23);Write('Linia 1 Coloana 2 (V1[1,2]) = ');Read(V1[1,2]);
GoToXY(2,23);ClrEol;
GoToXY(44,11);Write(V1[1,2]:4);

GoToXY(2,23);Write('Linia 1 Coloana 3 (V1[1,3]) = ');Read(V1[1,3]);
GoToXY(2,23);ClrEol;
GoToXY(63,11);Write(V1[1,3]:4);

GoToXY(2,23);Write('Linia 2 Coloana 1 (V1[2,1]) = ');Read(V1[2,1]);
GoToXY(2,23);ClrEol;
GoToXY(25,13);Write(V1[2,1]:4);

GoToXY(2,23);Write('Linia 2 Coloana 2 (V1[2,2]) = ');Read(V1[2,2]);
GoToXY(2,23);ClrEol;
GoToXY(44,13);Write(V1[2,2]:4);

GoToXY(2,23);Write('Linia 2 Coloana 3 (V1[2,3]) = ');Read(V1[2,3]);
GoToXY(2,23);ClrEol;
GoToXY(63,13);Write(V1[2,3]:4);
GoToXY(1,23);


ReadKey;
End.
```

Tipul articol

Tipul articol este un tip compus format dintr-un număr de componente, numite cîmpuri. Spre deosebire de tablouri, cîmpurile pot fi de tipuri diferite. Fiecare cîmp are un nume care este un identificator de cîmp. Numărul componentelor poate să fie fix sau variabil. În primul caz spunem că avem o structură cu articole fixe, în cel de-al doilea caz, avem o structură cu variante.

Forma generală a unei structuri cu articole fixe este **:**

```
TipArt=Record
      Nume_cîmp_1:Tip_1;
      Nume_cîmp_2:Tip_2;
      {…}
      End;
```

Exemplu :

```
Type data=Record
      An:1900..2100;
      Luna: (Ian,Feb,Mar,Apr,Mai,Iun,Iul,Aug,Sep,Oct,Noi,Dec);
      Ziua:1..31;
      End;
Var Astazi:Data;
```

Tipul reper

În **Turbo Pascal** variabilele pot fi **statice** sau **dinamice**. **Variabilele statice** sînt alocate în timpul compilării, iar spaţiul ocupat de ele în memorie nu poate fi modificat în execuţie. Ele există pe durata întregii execuţii a blocului (program, procedură, funcţie). **Variabilele statice** sînt declarate în secţiunea **Var**.

O variabilă poate fi creată şi distrusă dinamic în timpul execuţiei programului. O astfel de variabilă este denumită **variabilă dinamică**. **Variabilele dinamice** nu apar într-o declaraţie explicită, în secţiunea **Var** şi accesul la astfel de variabile nu se poate face direct.

Crearea şi distrugerea variabilelor dinamice se realizează cu procedurile **New** şi **GetMem** respectiv **Dispose** şi **FreeMem**. Aceste proceduri alocă, respectiv eliberează spaţiul de memorie pentru variabilele dinamice. Adresa zonei de memorie alocată unei variabile dinamice este depusă într-o variabilă de tip special, numită **reper**. Lungimea zonei de memorie atribuită unei variabile dinamice depinde de tipul variabilei dinamice **:** în funcţie de tip se alocă un număr variabil de de octeţi, variabilei respective. De exemplu, dacă tipul variabilei dinamice este **Integer**, se alocă 2 octeţi, iar dacă tipul este **Real**, se alocă 6 octeţi. În consecinţă, variabila de tip reper care va conţine adresa zonei alocate variabilei dinamice, trebuie să comunice procedurilor de alocare de memorie, tipul variabilei dinamice. Se menţionează că variabilele dinamice sînt alocate într-o zonă specială de memorie, numită **Heap**.

Definirea unui **tip reper** se poate face în secţiunea **Type**,în felul următor**:**

```
Type tip_reper=^ tip_variabilă_dinamică;
```

Unde semnul **^ (caret)** – semnifică o adresă.

Mulţimea valorilor **tip_reper** constă dintr-un număr delimitat de adrese. Fiecare adresă identifică o variabilă **tip_variabilă_dinamică**. La această mulţime de valori se mai adaugă o valoare specială, numită **nil**, care nu identifică nici o variabilă.

Limbajul permite ca în momentul întîlnirii tipului variabilei dinamice, aceasta să nu fi fost definită înainte, însă acest tip trebuie declarat mai tîrziu, într-o declaraţie de tip.

```
Type rep=^Art; {Referire înainte}
     Art=Record
           X,Y:integer;
           End;
Var  R1,R2:Rep;      {Pentru reperarea variabilei dinamice tip Art}
     R3:^integer;    {Pentru reperarea variabilei dinamice tip Integer}
     R4:^Char;       {Pentru reperarea variabilei dinamice tip Char}
```

O altă facilitate a limbajului constă în posibilitatea utilizării tipurilor care se autoreferă, adică sînt definite recursiv.

```
Type Lista=^Articol;
     Articol=Record
           A,B:Integer;
           Urmator:Lista;
        End;
Var  L:Lista;
```

Aici **tipul reper Lista**, reperează un **tip articol**, în care în cîmpul **următor** la rîndul lui este de asemenea de tip **Lista**. Această facilitate poate fi folosită de exemplu, la alcătuirea listelor înlănţuite.

După crearea unei variabile dinamice a cărei adresă este depusă într-o variabilă de tip reper, ea poate fi accesată prin operaţia numită **dereperare** adică **numele variabilei de tip reper** este urmat de semnul **^ (caret)**. Acest semn poate fi urmat şi de un alt calificator (de cîmp, de tablou). Dereperarea unei variabile de tip reper cu conţinut **nil** declanşează o eroare de execuţie.

Variabilele de tip reper sînt alocate pe 4 octeţi (2 octeţi pentru memorarea adresei de segment, 2 octeţi pentru memorarea deplasamentului).

Operaţiile relaţionale care sînt permise cu operanzii de **tipul reper**, compatibile sînt **egal (=)** şi **diferit (<>)**. Dacă **p1** şi **p2** sînt două valori tip reper compatibile, relaţia **p1=p2** este adevărată dacă **sînt egale părţile de segment** şi **deplasament**.

Tipul pointer

Tipul predefinit **pointer** este un tip reper care nu are tip de bază. Astfel, o variabilă de acest tip poate să repereze o variabilă de tip arbitrar şi din această cauză acest tip se mai numeşte şi **tip reper liber**. Declararea se face astfel **:**

Var ReperLiber:pointer;

Variabilele de **tip pointer** nu pot fi dereperate. Scrierea simbolului **‘ ^ ‘** după o astfel de variabilă constituie o eroare. Variabilele de **tip pointer** sînt utilizate pentru memorarea valorii unor variabile de **tip reper legat**. Unei variabile de acest tip îi poate fi atribuită şi valoarea predefinită **nil**.

Valori de tip reper pot fi create şi cu operatorul @ şi cu funcţia standard **Ptr**. Aceste valori sînt tratate ca şi cum ar fi repere pentru variabilele dinamice.

Compatibilitatea tipurilor

În anumite construcţii ale limbajului Turbo Pascal două tipuri trebuie să fie identice sau ele trebuie să fie compatibile sau compatibile doar din punct de vedere al atribuirii.

Tipuri identice

Identitatea de tip se impune între parametrii actuali şi parametrii formali variabili, adică cei specificaţi cu atributul **Var**.

Tipuri compatibile

Compatibilitatea de tipuri se impune în cazul expresiilor şi al operaţiilor de relaţie. Două tipuri sînt compatibile atunci cînd este adevărată cel puţin una din următoarele afirmaţii **:**

- Cele două tipuri sînt identice
- Ambele tipuri sînt reale (**real, simple, double, estended, comp**)
- Ambele tipuri sînt întregi (**integer, word, shortint, byte, longint**)
- Ambele tipuri sînt logice (**boolean, wordbool, longbool**)
- Un tip este un interval al celuilalt tip
- Ambele tipuri sînt intervale ale aceluiaşi tip de bază
- Ambele tipuri sînt tipuri mulţime, cu tipurile de bază compatibile
- Unul din tipuri este de tip **string** iar celălalt este de tip **string** sau **char**
- Unul din tipuri este **pointer**, iar celălalt este tip reper, dar două tipuri reper cu tipurile de bază diferite, nu sînt compatibile.
- Ambele tipuri sînt de tip procedură, unde numărul şi tipul parametrilor sînt identice

Vornicescu Silviu
vornicescus@yahoo.com
0741.145.430