P R O G R A M O V A N I E

TURBO PASCAL 6.0

Jazyk Turbo Pascal firmy Borland International je tak prínosnou a komerčne úspešnou implementáciou Wirthovho Pascalu, že je možné prehlásiť programátorovu neznalosť tejto implementácie za podobný poklesok, akým je vo všeobecnej rovine neznalosť práce s osobným počítačom vôbec . (Martin Kvoch, Programovanie v Turbo Pascale 6.0, KOPP 1992)

Profesor Niklaus Wirth pôsobil v rokoch 1963-1967 na stanfordskej univerzite, potom dva roky na univerzite v Zürichu. Od roku 1968, už ako profesor informatiky, prednášal na Vysokej škole technickej a na univerzite Zürichu.

Štruktúra pascalovského programu:

PROGRAM <meno programu>;
{direktívy prekladača}
USES <zoznam použitých knižníc (jednotiek, unitov)>
LABEL <deklarácia návestí>
CONST <deklarácia konštánt>
TYPE <definícia dátových typov>
VAR <deklarácia premenných>
PROCEDURE
FUNCTION
<deklarácia užívateľských procedúr a funkcií>
BEGIN
  <telo hlavného programu>
END.

Štruktúra pascalovského programu je pevne daná a nie je dovolené prehadzovať jednotlivé časti. V prípade, ak chceme zadeklarovať viac premenných alebo konštánt, stačí použiť len raz VAR, alebo CONST.

Deklarácia jednotiek: napr.:**USES** CRT, DOS, GRAPH;

Klauzula **uses** v programe uvádza zoznam jednotiek (knižníc), ktoré má prekladač prehľadať pri preklade programu, to znamená, že má zmysel uvádzať len tie jednotky, ktorých príkazy program využíva. Všetky pascalovské príkazy sú kvôli rýchlosti spracovania rozdelené do skupín - jednotiek. Je rýchlejšie prehľadať zopár jednotiek, ako zoznam všetkých príkazov. Navyše Pascal umožňuje vytvárať jednotky vlastné, v ktorých si môžem zadefinovať vlastné príkazy (procedúry a funkcie) a vlastné premenné.

Ak môj program používa príkazy určené pre textovú obrazovku, stačí ak napíšem len **uses crt;**. Ak začnem používať aj grafickú obrazovku, stačí deklaráciu jednotiek zmeniť na **uses crt, graph;**.

Deklarácia návestí, sekcia label:

napr.:
```
LABEL 1,2,3;
BEGIN
........
GOTO 5;
........
5: WRITELN('vykonal sa skok na 5');
........
END.
```

Návestie je postupnosť číslic 1 až 9999, alebo akýkoľvek identifikátor.. Návestie definuje miesto, na ktoré sa vykoná skok príkazom `GOTO`. Znamená to asi toľko, že program sa nemusí nutne vykonávať riadok po riadku, ale môžem príkazom `GOTO` niektoré riadky vynechať, prípadne ich nechať vykonať viackrát. V Turbo Pascale nie je povolený skok do alebo z bloku. Znamená to, že nie je možný skok do alebo z podprogramu. Skok do zloženého príkazu nie je Pascalom definovaný ako chyba, môže však spôsobiť vážne problémy. Skok je zvodná, ale nebezpečná štruktúra. Ak sa to dá, treba sa vyvarovať použitiu skokov (i keď niekedy je to riešenie najjednoduchšie).

Deklarácia konštánt: napr.:**`CONST`**

```
A = 1;          {číselná konštanta}
B = 'd';        {znaková konštanta}
C = 'retazec';  {reťazcová konštanta}
```

Konštanta: konštanta je dátový objekt, ktorého hodnota sa v priebehu výpočtu nemení. Aj z matematiky poznáme konštanty ako napr.: e=2.72, π=3.14 atď. Hodnotu konštanty nie je dovolené v priebehu výpočtu meniť (ak ju predsa chcem zmeniť, namiesto konštanty použijem premennú).

Deklarácia premenných:

napr.:**`VAR`**

```
a : integer;      {premenná a je typu celé číslo}
ab : real;        {premenná ab je typu reálne číslo}
c : char;         {premenná c je typu znak}
d : string[15]; {premenná d je typu reťazec maximálnej dĺžky 15 znakov}
```

Premenná: premenná je dátový objekt, ktorého hodnota sa v priebehu výpočtu môže meniť. Na rozdiel od matematiky dovoľuje Pascal používať aj iné ako číselné premenné. Použiť sa dajú aj znakové, reťazcové, dokonca aj nami zadefinované štruktúrované premenné.

Poznámka: meno premennej alebo konštanty (identifikátor, od identifikovať) je postupnosť znakov a číslic začínajúca znakom. Všeobecne platí, že dva rôzne objekty (premenné, konštanty, procedúry ...) nemôžu používať to isté meno, ten istý identifikátor.

Deklarácia: deklarácia slúži k zavedeniu a pomenovaniu určitých súčastí programu. V časti deklarácií uvádzam zoznam všetkých premenných, konštánt, procedúr, funkcií, ktoré v programe používam. Všeobecne platí, že v programe môže použiť len to, čo som zadeklaroval alebo zadefinoval.

Výraz: výraz je operačná štruktúra, pomocou ktorej sa vo vyšších programovacích jazykoch popisuje výpočet hodnoty. Výrazom môže byť napr.: `(-b-SQRT(d))/(2*a)`. Každý výraz sa skladá z operátorov (operácií) a operandov. Operátory sú v našom príklade `-,*,/` a operandy sú `b,d,a`. Pri zápise výrazov je potrebné uvedomiť si prioritu operátorov, napr. `a+b*c` sa vyhodnotí nasledovne: najskôr súčin `b*c`, ktorý sa potom sčíta s `a`. Ak chceme najskôr vyhodnotiť sčítanie, musíme náš výraz upraviť nasledovne: `(a+b)*c`. V tomto prípade sa vyhodnotí súčet `a+b`, ktorý sa potom vynásobí hodnotou `c`. V Pascale (podobne ako v matematike) platí nasledovná priorita operátorov:

Operátory v tom istom riadku majú rovnakú prioritu.

Operátor	Priorita
not	najvyššia
* / div mod and	⇓
+ - or xor	⇓
= <> <= >= < > in	najnižšia

+ operácia sčítania	**NOT** logická negácia	**=,<>,<=,>=,<,>** operácia rovný,
- operácia odčítania	**AND** logické a	rôzny, menší alebo rovný, väčší
***** operácia násobenia	**XOR** logické buď alebo	alebo rovný, menší, väčší
/ operácia delenia	**OR** logické alebo	**IN** patriaci do
DIV celočíselné delenie	**MOD** zvyšok po celočíselnom delení	

Príkaz: jednotlivé výpočtové akcie a ich náväznosti sa popisujú pomocou príkazov. Pascal rozlišuje dva druhy príkazov: jednoduché a zložené. Jednoduché príkazy sú napr.: `a:=1`, `WRITE(x)`. Zložené príkazy majú nasledovný tvar: `BEGIN príkaz1; príkaz2; ...END`. Všimnite si, že telo programu tvorí jeden zložený príkaz. Rezervované slová `BEGIN` a `END` môžeme chápať ako ľavú a pravú zátvorku. Použitie zloženého príkazu má svoj význam na miestach, kde Pascal dovoľuje použiť len jeden príkaz.

Rezervované slová: (týmto pojmom sa označujú slová v Pascale, ktoré majú presne definovanú funkciu a nesmú sa použiť k iný účelom) **AND, CASE, DIV, END, GOTO, OR, TYPE, VAR, CONST, DO, IF, NOT, REPEAT, STRING, UNIT, WHILE, ARRAY, DOWNTO, FOR, LABEL, THEN, UNTIL, BEGIN, ELSE, FUNCTION, MOD, PROGRAM, TO, USES, XOR.**

Typ definuje množinu prípustných hodnôt. Ak si zadefinujem premennú `a` typu `integer`, množinu prípustných hodnôt tvorí interval celých čísel <-32768,32767>. Žiaden typ nemôže definovať nekonečnú množinu hodnôt. Je to obmedzené hranicami intervalu, počtom platných číslic alebo jednoducho veľkosťou pamäte počítača.

Štandardné typy dát:

Celočíselné typy:

typ	**rozsah**
byte	0..255
shortint	-128..127
word	0..65535
integer	-32768..32767
longint	-2147483648..2147483647

Množina operácií: aritmetické: +, -, *, **DIV, MOD**
relačné: >, >=, <, <=, <>, =
(operácia / sem nepatrí, dôvodom je fakt, že napr.: výraz 4/2 má v Pascale hodnotu 2.000000000 čo je číslo reálne a to aj napriek tomu, že jeho desatinná časť je nulová)

Reálne typy:

typ	**rozsah**
real	$2.9*10^{-39}..1.7*10^{38}$
single	$1.5*10^{-45}..3.4*10^{38}$
double	$5.0*10^{-324}..1.7*10^{308}$
extended	$3.4*10^{-4932}..1.1*10^{4932}$
comp	$-2^{63}+1..2^{63}-1$

Množina operácií: aritmetické: +, -, *, /
relačné: >, >=, <, <=, <>, =

Konverzia medzi typmi INTEGER (int) **a REAL**(real)**:**

a	**b**	**a+b**	**a-b**	**a*b**	**a/b**	**a DIV b**	**a MOD b**	**a >,>=,<,<=,<>,= b•**
int	**int**	int	int	int	real	int	int	boolean
int	**real**	real	real	real	real	---	---	boolean

real	**int**	real	real	real	real	---	---	boolean
real	**real**	real	real	real	real	---	---	boolean

REAL := INTEGER - konverzia tohto typu je dovolená
INTEGER := REAL - konverzia tohto typu nie je dovolená

Typ char: premenná typu **char** je určená pre prácu so znakmi a obsahuje práve jeden znak v ASCII (American Standard Code for Information Interchange, zjednodušene sú to všetky znaky, ktoré môžeme napísať pomocou klávesnice) kóde. Konštanty typu char sú uzavreté medzi apostrofmi, napr. 'd', 'D'. Pozor: 'd' <>'D'.

Typ string: premenná typu **string** je určená pre prácu s reťazcami (postupnosťami) znakov dĺžky 1 až 255. Ak nie je uvedená dĺžka, predpokladá sa dĺžka 255 znakov.

Konverzia medzi typmi CHAR (chr) **a STRING** (str)**:**

a	**b**	**a+b** (zreťazenie)	**a > , >= , < , <= , <> , = b**
char	**char**	**string**	boolean
char	**string**	**string**	boolean
string	**chrar**	**string**	boolean
string	**string**	**string**	boolean

STRING := CHAR - konverzia tohto typu je dovolená
CHAR := STRING - konverzia tohto typu nie je dovolená

Typ boolean: premennej typu **boolean** môže byť priradená práve jedna z dvoch hodnôt: true (pravda) alebo false (nepravda). Je to logická premenná a používa sa pri práci s logickými premennými.

Význam logických spojok	
AND	logická spojka **a**
OR	logická spojka **alebo**
XOR	logická spojka **buď alebo**
NOT	logická **negácia**

Pravdivostná tabuľka logických spojok					
A	B	A and B	A or B	A xor B	NOT(A)
T	T	T	T	F	F
T	F	F	T	T	F
F	T	F	T	T	T
F	F	F	F	F	T

Množina operácií: logické: **AND**, **OR**, **NOT**, **XOR**
relačné: >, >=, <, <=, <>, =

Jednoduché typy pracujú s lineárne usporiadanou množinou znakov M, pre ktoré platí: "**a,b** Î **•M => (a > b)** Ú **(a = b)** Ú **(a < b)** (trichotómia). Na základe tejto vlastnosti sú v Pascale pre jednoduché typy (real nie je jednoduchý typ) definované funkcie **SUCC, PRED, ORD** (viď. ordinárne funkcie). Jednoduché typy, pre ktoré sú definované tieto funkcie, nazývame **ordinárne**.

Štandardné procedúry a funkcie Pascal-u:

Pre všetky procedúry, funkcie a príkazy jazyka Pascal platí, že pred ich použitím v programe je vhodné si preštudovať nápovedu jazyka Pascal. Všeobecnú nápovedu získame stlačením klávesu <F1>, nápovedu ku konkrétnemu príkazu kombináciou klávesov <*Ctrl F1*> (stačí sa nastaviť kurzorom na príslušné slovo, ktorého nápovedu chceme použiť a stlačiť <*Ctrl F1*>).

Nasleduje zoznam procedúr a funkcií, rozdelený do jednotlivých kategórií. Pri každej procedúre a funkcii je uvedený stručný popis jej činnosti a jej deklarácia. V časti deklarácie časti v hranatých zátvorkách [v týchto] sú nepovinné. Je dôležité dodržiavať presnú syntax daných podprogramov (procedúry a funkcie sa súhrnne nazývajú podprogramy). Všímajte si počet parametrov a ich typy.

Štandardné procedúry a funkcie

DISPOSE procedúra uvoľní dynamickú premennú z operačnej pamäti
deklarácia: `Dispose(var p : pointer)`

EXIT procedúra spôsobí návrat z podprogramu (ak je volaná z procedúry alebo funkcie) alebo ukončí program, ak je volaná z hlavného programu
deklarácia: `Exit`

HALT procedúra ukončí program a vráti riadenie operačnému systému
deklarácia: `Halt`

NEW procedúra vytvára novú dynamickú premennú a nastavuje ukazovateľ tak, aby ukazoval na túto premennú, referenciu (odkaz) na novo vzniknutú premennú možno previesť napr.: pomocou p^
deklarácia: `New(var p : pointer)`

Konverzné funkcie

CHR funkcia vracia znak, ktorý je reprezentovaný číselnou hodnotou, konvertuje celočíselnú typ na typ char
deklarácia: `Chr(x : byte) : byte`

ORD funkcia vracia ordinárne číslo špecifikovanej hodnoty ordinárneho typu, konvertuje typ ordinárny na typ integer
deklarácia: `Ord(x) : longint`

ROUND funkcia zaokrúhľuje reálne číslo na celočíselnú hodnotu, konvertuje typ real na typ celočíselný
deklarácia: `Round(x : real) : longint`

TRUNC funkcia orezáva desatinnú časť reálneho čísla a získanú hodnotu vracia ako celočíselnú, konvertuje typ real na typ celočíselný
deklarácia: `Trunc(x : real) : longint`

Aritmetické funkcie

ABS funkcia vracia absolútnu hodnotu, výsledok je rovnakého typu ako parameter x
deklarácia: `Abs(x : y) : y`

ARCTAN funkcia vracia arcustangens (výsledok je v radiánoch)
deklarácia: `ArcTan(x : real) : real`

COS funkcia vracia kosínus (argument je v radiánoch)
deklarácia: `Cos(x : real) : real`

EXP funkcia vracia hodnotu e^x
deklarácia: `Exp(x : real) : real`

FRAC funkcia vracia desatinnú časť reálneho argumentu (`x-Int(x)`)
deklarácia: `Frac(x : real) : real`

INT funkcia vracia celú časť reálneho argumentu (výsledok je typu real)
deklarácia: `Int(x : real) : real`

LN funkcia vracia prirodzený logaritmus argumentu (logaritmus pri základe e=2.718281828)
deklarácia: `Ln(x : real) : real`

PI funkcia vracia hodnotu 3.1415926535897932386 (π)
deklarácia: `Pi : extended`

SIN funkcia vracia sínus argumentu (argument v radiánoch)
deklarácia: `Sin(x : real) : real`

SQR funkcia vracia druhú mocninu argumentu
deklarácia: `Sqr(x : real) : real`

SQRT funkcia vracia druhú odmocninu argumentu
deklarácia: `Sqrt(x : real) : real`

Ordinárne funkcie

DEC procedúra zmenšuje ordinárnu premennú x o zadaný počet n, ak nie je n zadané, predpokladá sa jeho hodnota 1, platí `Dec(x,n) <=> x := x - n`
deklarácia: Dec(var x [; n : longint])

INC procedúra zväčšuje ordinárnu premennú x o zadaný počet n, ak nie je n zadané, predpokladá sa jeho hodnota 1, platí `Inc(x,n) <=> x := x + n`
deklarácia: Inc(var x [; n : longint])

ODD funkcia vracia true ak je celočíselný argument párny, inak vráti false
deklarácia: `Odd(x : longint) : boolean`

PRED funkcia vracia predchodcu argumentu, ktorý je ordinárneho typu, výsledok je rovnakého typu ako argument
deklarácia: Pred(x : ordinal) : ordinal

SUCC funkcia vracia nasledovníka argumentu, ktorý je ordinárneho typu, výsledok je rovnakého typu ako argument
deklarácia: Succ(x : ordinal) : ordinal

Procedúry a funkcie pre prácu s reťazcami

STR procedúra konvertuje numerickú hodnotu na jej reprezentáciu v reťazci znakov
deklarácia: `Str(x [: dlzka[: desatinne]]; var s : string)`

VAL procedúra konvertuje hodnotu typu string do zodpovedajúcej numerickej hodnoty, premenná code je po úspešnom prevedení nastavená na 0, inak ukazuje index znaku, pri ktorom došlo k chybe, premenná v je typu integer alebo real
deklarácia: `Val(s : string; var v; code : integer)`

Ostatné procedúry a funkcie

RANDOM funkcia vracia náhodné číslo, ak nie je zadaný parameter funkcia generuje reálne číslo x v rozsahu $0 \leq x \leq 1$, ak je zadaný parameter celočíselného typu, tak generuje celé číslo x z intervalu $0 \leq x \leq n$
deklarácia: `Random(n : word) : word`
`Random : real`

RANDOMIZE procedúra inicializuje generátor náhodných čísel
deklarácia: `Randomize`

Štandardné precedúry a funkcie pre súbory

ASSIGN procedúra priraďuje meno externého súboru premennej typu súbor
deklarácia: `Assign(var f : file; meno : string)`

APPEND procedúra otvára existujúci súbor (pozor, len typu text) pre zápis a nastavuje pozíciu zápisu na koniec súboru
deklarácia: `Append(var f : text)`

CLOSE procedúra uzatvára otvorený súbor
deklarácia: `Close(var f : file)`

EOF funkcia nadobudne hodnotu true , ak sa dosiahne koniec súboru, inak je false
deklarácia: `Eof [(var f : file)]`

EOLN funkcia má hodnotu true, ak je pozícia súboru (pozor, len textový súbor) na znaku konca riadku
deklarácia: `Eoln [(var f : text)]`

READ, READLN procedúra umožňuje načítať hodnotu z textového súboru (z klávesnice), ln umožní prechod na nový riadok v súbore (na obrazovke)
deklarácia: `Read([var f : text;] v1 [,v2, ...,vn])`
deklarácia: `Readln[([var f : text;] v1 [,v2, ...,vn])]`

RESET procedúra otvára existujúci súbor pre čítanie
deklarácia: `Reset(var f : file)`

REWRITE procedúra otvára nový súbor pre zápis, starý maže
deklarácia: `Rewrite(var f : file)`

WRITE, WRITELN procedúra umožňuje zapísať hodnotu do súboru (na obrazovku), ln zapíše aj znak konca riadku (presunie kurzor do nového riadku)
deklarácia: `Write([var f : text;] v1 [,v2, ...,vn])`
deklarácia: `Writeln[([var f : text;] v1 [,v2, ...,vn])]`

Procedúry a funkcie jednotky CRT:

CLREOL procedúra vymaže znaky od kurzora do konca riadku vrátane
deklarácia: `CleEol`

CLRSCR procedúra vymaže obrazovku a kurzor umiestni do ľavého horného rohu, výsledná farba obrazovky je daná nastavením farby pozadia procedúrou TextBackground, ak je na obrazovke okno, týka sa zmena len okna
deklarácia: `ClrScr`

DELAY procedúra pozastaví výpočet na špecifikovaný počet milisekúnd
deklarácia: `Delay(pocet_milisekund : word)`

DELLINE procedúra vymaže riadok, na ktorom je kurzor, a zvyšok textu posunie smerom hore
deklarácia: `DelLine`

GOTOXY procedúra umiestni kurzor na špecifikovanú pozíciu v aktuálnom okne
deklarácia: `GotoXY(x,y : byte)`

KEYPRESSED funkcia váti true v prípade, že bol stlačený nejaký kláves
deklarácia: `KeyPressed : boolean`

NOSOUND procedúra vypína vnútorný tónový generátor, inak zostáva zvukový generátor zapnutý aj po skončení programu
deklarácia: `NoSound`

READKEY funkcia prečíta znak z klávesnice bez jeho výpisu na obrazovku
deklarácia: `Readkey : char`

SOUND procedúra zapne vnútorný tónový generátor na frekvenciu zadanú parametrom f v Hz
deklarácia: `Sound(f : byte)`

TEXTBACKGROUND procedúra určí farbu pozadia textu (v rozsahu 0 .. 7), pri použití clrscr sa zmena prevedie v aktuálnom okne, bez clrscr len na práve písaných znakoch
deklarácia: `TextBackground(farba : byte)`

TEXTCOLOR procedúra určuje farbu textu (v rozsahu 0 .. 15)
deklarácia: `TextColor(farba : byte)`

WHEREX funkcia vracia x-ovú súradnicu kurzoru na obrazovke (vzhľadom na aktívne okno)
deklarácia: `WhereX : byte`

WHEREY funkcia vracia y-ovú súradnicu kurzoru na obrazovke (vzhľadom na aktívne okno)
deklarácia: `WhereY : byte`

WINDOW procedúra definuje textové okno na obrazovke, kde x1, y1 je ľavý horný roh a x2, y2 je pravý dolný roh
deklarácia: `Window(x1,y1,x2,y2 : word)`

Procedúry a funkcie jednotky GRAPH:

Grafický výstup nie je pre Pascal štandardný, preto ho treba zvlášť inicializovať. Môžeme to spraviť postupnosťou príkazov:

```
DETECTGRAPH(gd,gm);
INITGRAPH(gd, gm, 'cesta k suborom BGI')
```
grafická obrazovka je inicializovaná a môžeme používať grafické príkazy
```
CLOSEGRAPH;
```

Keď je inicializovaná grafická obrazovka, nedajú sa použiť napr. príkazy write a read.

BAR procedúra nakreslí obdĺžnik aktuálnym typom výplne a aktuálnou farbou (pozri setfillstyle)
deklarácia: `Bar(x1,y1,x2,y2 : integer)`

BAR3D procedúra nakreslí 3-rozmerný stĺpec aktuálnym typom výplne (pozri setfillstyle) a aktuálnou farbou (pozri setcolor), ktorého tretí rozmer má veľkosť hĺbka a kreslenie hornej časti povoľuje (zakazuje) hodnota logickej premennej vrchol
deklarácia:
`Bar3D(x1,y1,x2,y2:integer;hlbka:word;vrchol:boolean)`

CIRCLE procedúra nakreslí kružnicu s polomerom r a stredom x,y
deklarácia: `Circle(x,y : integer; r : word)`

CLEARDEVICE procedúra vymaže grafickú obrazovku
deklarácia: `ClearDevice`

CLOSEGRAPH procedúra ukončí prácu v grafickom režime, je potrebné ju uviesť vždy pred ukončením grafického režimu
deklarácia: `CloseGraph`

DETECTGRAPH procedúra zistí, aký grafický driver a aký grafický režim potrebujeme pre činnosť grafického programu
deklarácia: `DetectGraph(var gd, dm : integer)`

ELLIPSE procedúra nakreslí eliptický oblúk so stredom x,y, začiatočným uhlom zu, koncovým uhlom ku, veľkosťou vodorovnej polosi vo a veľkosťou zvislej polosi zo
deklarácia: `Ellipse(x,y:integer; zu,ku:word; vo,zo:word)`

FILLELLIPSE procedúra nakreslí elipsu so stredom x,y, veľkosťou vodorovnej polosi vo a veľkosťou zvislej polosi zo vyplnenú aktuálnym výplňovým vzorom
deklarácia: `FillEllipse(x,y:integer; vo,zo:word)`

FLOODFILL procedúra zaplní ohraničenú oblasť aktuálnym typom výplne, je potrebné zadať bod x,y ležiaci vo vnútri oblasti a farbu hranice uzatvorenej oblasti
deklarácia: `FloodFill(x,y:integer; farba_hranice:word)`

GETX funkcia vracia x-ovú súradnicu aktuálnej polohy grafického kurzora
deklarácia: `GetX : integer`

GETY funkcia vracia y-ovú súradnicu aktuálnej polohy grafického kurzora
deklarácia: `GetY : integer`

INITGRAPH procedúra inicializuje grafický systém a nastavuje grafický režim (gm) na grafickom adaptéry (gd), pozri detectgraph
deklarácia:
`InitGraph(var gd:integer;var gm:integer;cesta:string)`

LINE procedúra nakreslí čiaru ako spojnicu dvoch bodov [x1,y1]a [x2,y2]
deklarácia: `Line(x1, y2, x2, y2 : integer)`

LINEREL procedúra nakreslí čiaru od aktuálnej polohy grafického kurzora do bodu, ktorého súradnice sú dané prírastkom dx v smere osi x a prírastkom dy v smere osi y k aktuálnej pozícií kurzora
deklarácia: `LineRel(dx, dy : integer)`

LINETO procedúra nakreslí čiaru z pozície aktuálneho kurzora do definovaného bodu [x,y]
deklarácia: `LineTo(x,y : integer)`

MOVEREL procedúra posunie grafický kurzor do bodu, ktorého súradnice sú dané prírastkom dx v smere osi x a prírastkom dy v smere osi y k aktuálnej pozícii kurzora
deklarácia: `MoveRel(dx, dy : integer)`

MOVETO procedúra premiestni grafický kurzor na definovanú pozíciu [x,y]
deklarácia: `MoveTo(x,y : ineteger)`

OUTTEXT procedúra zobrazí textový reťazec na aktuálnu pozíciu grafického kurzora
deklarácia: `OutText(text : string)`

OUTTEXTXY procedúra zobrazí textový reťazec na definované súradnice [x,y]
deklarácia: `OutTextXY(x,y : integer; text : string)`

PUTPIXEL procedúra nakreslí na pozícii [x,y] bod zadanej farby
deklarácia: `PutPixel(x,y : integer; farba : word)`

SETCOLOR procedúra nastaví farbu pre kreslenie čiar a iných obrazcov
deklarácia: `SetColor(farba : word)`

SETBKCOLOR procedúra nastaví farbu pozadia
deklarácia: `SetColor(farba : word)`

SETFILLSTYLE procedúra nastaví nový výplňový vzor a jeho farbu
deklarácia: `SetFillColor(vzorka, farba : word)`

SETTEXTSTYLE procedúra nastaví znakový font, smer písania a koeficient zväčšenia grafického textu
deklarácia:
`SetTextStyle(font:word; smer:word; zvatsenie: word))`

Štandardné príkazy PASCALu:

Priraďovací príkaz: :=. Realizáciou priraďovacieho príkazu sa priradí premennej na ľavej strane príkazu hodnota výrazu na pravej strane príkazu.

Napr.:

príkaz	efekt
`a := 1;`	hodnota `a` je 1
`b := 21 - a;`	hodnota `b` je 20
`c := 'ret' + 'azec';`	hodnota `c` je 'retazec'

Príkaz výstupu: WRITE, WRITELN (writeln - prechod na nový riadok). Príkaz výstupu zabezpečuje výstup údajov z programu na monitor alebo do súboru.
Napr.: (a=2, b=3)

príkaz	výstup na monitor
`WRITE(1);`	`1`
`WRITE(1,2);`	`12`
`WRITE(a,b);`	`23`
`WRITE(a,' ',b);`	`2 3`
`WRITE('a','b');`	`ab`
`WRITE('a'+'b');`	`ab`
`WRITE(a+b);`	`5`

Formátovanie výstupu: Príkaz WRITE má všeobecný tvar WRITE(*parameter*). Pre *parameter* sú povolené celkom tri rôzne tvary:
výstupný údaj
výstupný údaj : počet znakov

výstupný údaj : počet znakov : počet desatinných miest

Napr.: (I=-253, R=345.25, Z='A', B=true)

príkaz:	výstup na monitor
`WRITE(I:4);`	`- 2 5 3`
`WRITE(ABS(I):5);`	`  2 5 3`
`WRITE(I:2);`	`- 2 5 3`
`WRITE(R:8:3);`	`  3 4 5 . 2 5 0`
`WRITE(R:9);`	`  3 . 4 5 E + 0 2`
`WRITE(Z:5);`	`  A`
`WRITE(B:5);`	`  t r u e`

Poznámka: číslo `3.45E+02` je počítačový zápis čisla 3,45 . 10^2.

Príkaz vstupu: READ, READLN (`readln - prechod na nový riadok`). Príkaz vstupu zabezpečuje vstup údajov z klávesnice alebo zo súboru do programu.

Napríklad: príkaz	vstup z kláv.	efekt
`READ(a)`	`23`	hodnota a je 23
`READ(a,b,c)`	`1 aa 3.14`	hodnota a je 1, b je 'aa' a c je 3.14

Príkaz **IF** slúži na podmienené vetvenie programu. To znamená, že v závislosti od platnosti nejakej podmienky môžeme zmeniť postup výpočtu. Všimnite si program v úvode, kde na základe platnosti (alebo neplatnosti) podmienky D>=0 program nájde dve riešenia kvadratickej rovnice alebo vyhlási, že rovnica riešenie nemá. Príkaz IF má dva tvary: príkaz IF úplný (zariadi to, čo sa má spraviť ak podmienka platí a čo sa má spraviť ak podmienka neplatí) a príkaz IF neúplný.(ten zariadi len to, čo sa má spraviť ak podmienka neplatí).

Príkaz IF (neúplný): **IF** *podmienka* **THEN** *príkaz*; alebo

IF *podmienka* **THEN** **BEGIN**
príkaz$_1$;
príkaz$_2$;
...
príkaz$_n$;
END;

Príkaz sa vykoná, resp. *príkaz*$_1$ až *príkaz*$_n$ sa vykoná len v tom prípade, ak *podmienka* je splnená. Ak podmienka nie je splnená, nevykoná sa ani jeden z uvedených príkazov. Podmienka je výraz typu boolean (ide teda o logický výraz, ktorý buď platí, alebo neplatí).

Napr.:
```
WRITELN('Zadaj vstupne heslo');
READLN(a);
IF  a = 'tajne'  THEN  WRITELN('Heslo je spravne');
```

Príkaz IF (úplný):

```
IF podmienka THEN príkaz1
               ELSE príkaz2;
```

alebo

```
IF podmienka THEN BEGIN
                  príkaz1;
                  príkaz2;
                  ...
                  príkazn;
                  END
               ELSE BEGIN
                  príkazn+1;
                  príkazn+2;
                  ...
                  príkazk;
                  END;
```

Príkaz$_1$ sa vykoná, resp. *príkaz*$_1$ až *príkaz*$_n$ sa vykoná len v tom prípade, ak *podmienka* je splnená. *Príkaz*$_2$ sa vykoná, resp. *príkaz*$_{n+1}$ až *príkaz*$_k$ sa vykoná len v tom prípade, ak *podmienka* nie je splnená. Pred ELSE sa ';' nepíše. Nemôže sa stať, aby sa vykonali všetky príkazy *príkaz*$_1$ až *príkaz*$_n$ a súčasne príkazy *príkaz*$_{n+1}$ až *príkaz*$_k$, alebo aby sa nevykonal ani jeden príkaz. V tomto prípade by podmienka musela platiť a neplatiť súčasne, čo je však nemožné.

Napr.
```
WRITELN('Zadaj vstupne heslo');
READLN(a);
IF a = 'tajne'  THEN WRITELN('Heslo je spravne')
                ELSE WRITELN('Heslo je nespravne');
```

Príkaz CASE: Príkaz case umožňuje viacnásobné podmienené vetvenie programu do niekoľkých alternatív v závislosti na hodnote premennej alebo výrazu.

```
CASE premenna OF
   hodnota1 : prikaz1;
   hodnota2 : prikaz2;
   ...
   hodnotan : prikazn
ELSE prikazn+1;
END;
```

Napr.:
```
WRITELN('Zadaj znak');
READLN(a);
CASE a OF
 'a'..'z'   : WRITELN('Male pismeno');
 'A'..'Z'   : WRITELN('Velke pismeno');
   '0'..'9' : WRITELN('Cislica');
 '<'        : WRITELN('Znak mensi')
ELSE          WRITELN('Iny znak');
END;
```

Príkaz$_i$ (i je z intervalu <1,n>) sa vykoná len vtedy, ak hodnota *premennej* je *hodnota*$_i$. *Premenna* a *hodnota*$_i$ musia byť rovnakého ordinárneho typu. V prípade, že hodnota *premennej* je rôzna od *hodonota*$_1$ až *hodnota*$_n$ vykoná sa *príkaz*$_{n+1}$. Vetva **ELSE** nie je povinná (ak by sme vetvu ELSE vynechali a hodnota *premennej* by bola rôzna od *hodonota*$_1$ až *hodnota*$_n$ nevykonal by sa ani jeden z daných príkazov). *Premenná* musí byť ordinálneho typu (napr. integer, char).

Príkazy cyklu

Pomocou príkazu cyklu sa predpisuje opakované vykonávanie príkazov alebo postupnosti príkazov (niekedy potrebujeme viackrát opakovať tie isté alebo podobné príkazy, máme v podstate dve možnosti: buď daný príkaz 100 krát opíšeme, alebo vytvoríme cyklus a povieme mu, aby príkaz 100 krát zopakoval). Okrem príkazov, ktoré sa majú opakovane vykonávať, je súčasťou každého príkazu cyklu aj špecifikácia riadiaca toto opakovanie. Pri príkazoch `repeat`, resp. `while` je táto špecifikácia daná podmienkou (výrazom typu boolean), ktorej splnenie alebo nesplnenie znamená opakované vykonávanie cyklu. V príkaze `for` sa udáva interval hodnôt ordinárneho typu, pre ktoré sa príkaz postupne vykonáva. Často býva základným problémom rozhodnúť sa, ktorý z cyklov použiť. Pomôcť mi môže toto: ak presne viem koľkokrát chcem príkaz opakovať, použijem FOR, ak opakovanie príkazu závisí od platnosti nejakej podmienky, ale aspoň raz sa zopakovať musí, použijem REPEAT, ak opakovanie príkazu závisí od platnosti nejakej podmienky, ale príkaz se nemusí zopakovať ani raz, použijem WHILE.

Príkaz cyklu FOR TO: `FOR` i := *zac_hod* `TO` *konc_hod* `DO` *príkaz*;

V prípade, že *zac_hod* <= *konc_hod* sa *príkaz* vykoná *konc_hod* - *zac_hod* + 1 krát. V opačnom prípade sa *prikaz* nevykoná. Premenná i sa nazýva riadiacou premennou cyklu a je ordinárneho typu (napr. integer, char). Táto premenná nadobúda počas práce cyklu hodnoty od *zac_hod* po *konc_hod* (pri každom prechode cyklom sa zväčší o 1). Namiesto príkazu môžeme použiť aj zložený príkaz.

Napr.:
```
WRITELN('Zadaj cele cislo');
READLN(cislo);
z:=1; k:=10;
WRITELN('Nasobky cisla:',cislo);
FOR i:=z TO k DO WRITELN(i,'*',cislo,'=',i*cislo);
```

Príkaz cyklu FOR DOWNTO: `FOR` i := *zac_hod* `DOWNTO` *konc_hod* `DO` *príkaz*;

V prípade, že *zac_hod* >= *konc_hod* sa *príkaz* vykoná *zac_hod* - *konc_hod* + 1 krát. V opačnom prípade sa *prikaz* nevykoná. Premenná i sa nazýva riadiacou premennou cyklu a je ordinárneho typu (napr. integer, char). Táto premenná nadobúda počas práce cyklu hodnoty od *zac_hod* po *konc_hod* (pri každom prechode cyklom sa zmenší o 1).

Napr.:
```
WRITELN('Mala anglicka abeceda odzadu:');
z:='z';
FOR i:=z DOWNTO 'a' DO WRITELN(i,' ');
```

Poznámka: Všimnite si rozdiel medzi `z` a `'z'`. Zápis: `z` označuje premennú z a zápis: `'z'` označuje znak malého písmena z. Hodnotou premennej z je znak `'z'` (premenná `z` teda musí byť typu `char`). Ako koncovú hodnotu sme nepoužili premennú, ale znak malé písmenko a.

Príkaz `FOR` bude opakovať len jeden príkaz. Vždy je to prvý príkaz uvedený za rezervovaným slovom `DO`. V prípade, ak chceme v cykle opakovať viac ako jeden príkaz, uzatvoríme tieto príkazy medzi rezervované slová `begin` a `end`., inými slovammi použijeme zložený príkaz. Príkaz `FOR` sa nazýva aj príkaz cyklu s pevným počtom opakovaní. Vždy vieme presne povedať, koľko krát sa daný cyklus bude opakovať (pozri vyššie).

Príkaz cyklu WHILE: **WHILE** *podmienka* **DO** *prikaz*; alebo

WHILE *podmienka* **DO BEGIN**
prikaz$_1$;
prikaz$_2$;
...
prikaz$_n$;
END;

V prípade, že podmienka platí, tak sa *prikaz* resp. *prikaz*$_1$, *prikaz*$_2$, ..prikaz$_n$ bude opakovane vykonávať pri každom prechode cyklom. V prípade, že podmienka neplatí, príkaz while sa ukončí. Telo cyklu (príkazy sa slovom DO) musí obsahovať aj príkazy, ktoré majú vplyv na platnosť podmienky. V opačnom prípade bude vykonávanie cyklu nekonečné. Keďže najskôr sa testuje platnosť podmienky, cyklus sa nemusí zopakovať ani raz (stane sa to v tom prípade, ak podmienka hneď na začiatku neplatí, pozri repeat). Pretože podmienka je na začiatku, nazýva sa tento cyklus aj cyklus s podmienkou na začiatku. Cyklus WHILE končí, ak podmienka prvýkrát neplatí.

Tento program (presnejšie povedané časť programu) vypíše všetky znaky, ktoré sa dajú zobraziť pomocou klávesnice (pozor, niektoré znaky sú neviditeľne, napr.: medzera). Všetky znaky sú zoradené jeden za druhým. Každému z nich je pridelené poradové číslo. Všetkých znakov je 256 a majú poradové čísla od 0 do 255. Funkcia chr (pozri konverzné funkcie) priradí poradovému číslu znak s týmto poradovým číslom. Druhý cyklus nikdy neskončí, pretože hodnota premennej i bude stále 255, čo je vždy väčšie ako 0.

Napr. správne

```
WRITELN('Vypis ASCII znakov:')
i := 255;
WHILE i >= 0 DO BEGIN
                 WRITELN(CHR(i));
                 i := i - 1;
                END;
```

nesprávne

```
WRITELN('Vypis ASCII znakov:')
i := 255;
WHILE i >= 0 DO  BEGIN
                  WRITELN(chr(i));
                 END;
```

Príkaz cyklu REPEAT: **REPEAT**
prikaz$_1$;
prikaz$_2$;
...
prikaz$_n$;
UNTIL *podmienka*;

Príkaz cyklu REPEAT zabezpečí opakované vykonávanie príkazov *prikaz*$_1$, *prikaz*$_2$, ... *prikaz*$_n$. Príkazy sa opakujú, pokiaľ *podmienka* neplatí. Príkaz cyklu REPEAT skončí, ak *podmienka* po prvýkrát platí. Telo cyklu musí obsahovať príkazy, ktoré majú vplyv na platnosť podmienky. V opačnom prípade sa cyklus REPEAT môže opakovať nekonečne veľakrát (bude teda nekonečným cyklom). Keďže najskôr sa vykonajú príkazy *prikaz*$_1$, *prikaz*$_2$, ... *prikaz*$_n$ a až potom sa testuje platnosť podmienky, vykonajú sa príkazy cyklu vždy aspoň raz (ak sa príkazy cyklu zopakovali práve raz, nastal prípad, že hneď pri prvom testovaní podmienky podmienka platí, pozri while). Cyklus REPEAT sa nazýva aj cyklus s podmienku na konci. Všimnite si, že ak chceme v tele cyklu zopakovať viac ako jeden príkaz, tak nemusíme použiť begin a end.

Tento algoritmus zistí, či zadané číslo x je prvočíslo (t.j. také číslo, ktoré má práve dva rôzne delitele, jedničku a samé seba). Algoritmus skúša deliť číslo x každým celým číslom z intervalu $<2,\sqrt{x}>$. Na začiatku má premenná je_delitel hodnotu true (predpokladáme, že číslo x je prvočíslo) V prípade, že

algoritmus nájde číslo, ktorým je x deliteľné ((x mod delitel)=0), hodnota premennej je_delitel sa zmení na false. Cyklus REPEAT skončí ak premenná je_prvočíslo má hodnotu false (not(false)=true) alebo delitel>$\sqrt{x}$.
Pr.:

```
WRITELN('Zadaj prirodzene cislo');
READLN(x);
je_prvocislo:=true;
delitel:=2;
REPEAT
IF (x mod delitel)=0 THEN je_prvocislo:=false;
delitel:=delitel+1;
UNTIL NOT(je_prvocislo) OR (delitel>SQRT(x));
IF je_delitel THEN WRITELN('Cislo ',x,' je prvocislo')
              ELSE WRITELN('Cislo ',x,' nie je prvocislo');
```

Užívateľom definované typy

Pri spracovaní dát nie vždy vystačíme s typmi premenných (dát), ktoré ponúka Pascal. Navyše údaje, ktoré potrebujeme spracovať, môžu byť rôzne štruktúrované (napríklad informácia o jednom človeku môže byť hodnotou jednej premennej, ale tá ako taká sa môže skladať z jeho mena, priezviska, adresy, telefónneho čísla, čísla občianskeho preukazu atď.) a jednoduché pascalovské typy už nestačia. Pascal aj tu ponúka možnosť. Dovoľuje definovať vlastné typy (pozri deklaráciu typov pri štruktúre programu). Základnými metódami štruktúrovania sú **array** (pole), **record** (záznam), **set** (množina) a **postupnosť** (súbor). Vybrať správnu štruktúru reprezentovania dát v počítači patrí medzi kľúčové problémy programátorského umenia.

Typ pole: Pole je homogénna dátová štruktúra skladajúca sa so zložiek rovnakého typu, ktoré sa vzájomne rozlišujú pomocou indexu. Využíva sa najmä vtedy, ak potrebujeme pomocou premenných uchovať veľa hodnôt rovnakého typu (mená všetkých ľudí v triede, známky žiakov, záznamy o ľuďoch a pod.). Na sprístupnenie prvku poľa je potrebné poznať meno celej štruktúry a index (indexom zvyčajne býva interval, alebo viac intervalov) prvku v poli.
Popis typu pole (jeho definícia) má základný tvar:

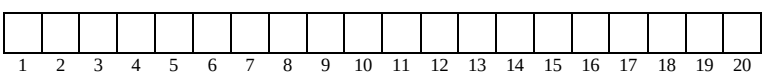
TYPE meno_pola =**ARRAY**[typ indexu] **OF** typ položky

Týmto sme definovali len typ (len nejakú štruktúru), Aby sme ju mohli aj využívať, potrebujeme si ju sprístupniť. Deklarujeme teda premennú, ktorá bude typu "meno_pola" (všimnite si, že typ meno_pola sme definovali, ale premennú ktorá je typu meno_pola sme deklarovali, definovať znamená slovne vymedziť pojem uvedením jeho základných vlastností a deklarovať znamená vyhlásiť, že v programe použitá premenná má vlastnosti definované v definícii poľa).

Počet položiek poľa je daný počtom rôznych hodnôt, ktoré patria do typu indexu. Často tým intervalom býva nejaký intervel z typu integer, ale môže to byť ľubovoľný ordinárny typ (znamená to, že hodnoty tohto indexu musia byť usporiadané).

Pr.: Nasledovný príklad ukazuje praktické využitie typu pole. V prvom riadku definujeme typ pole, ktoré má maximálne 20 položiek (pole má jednu nevýhodu, vždy treba dopredu definovať maximálny počet položiek, i keď ich v programe nemusíme využiť všetky). Ďalej si deklarujeme premennú a, ktorá je typu pole. To čo sme zatiaľ vytvorili, vyzerá v reprezentácií počítača asi nasledovne:

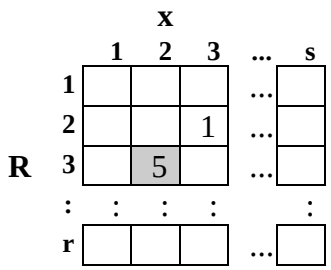
premenná **a** vyzerá takto:
jednotlivé časti sú indexované



Je zatiaľ prázdne, ale do každého okienka môžeme zapísať jedno celé číslo. V cykle REPEAT sa načítavajú položky pola, pričom po každom načítaní zväčšíme index o 1 (inc(index)). Cyklus (načítavanie) skončí ak zadáme nulu, alebo ak bolo zadaných 20 čísel (po 20 čísle je hodnota indexu 21). Potom v cykle od 1 do index-1 bude počítač hrať príslušné tóny (sound).

```
TYPE pole=ARRAY[1..20] OF WORD;
VAR a:pole;
....
WRITELN('Zadavaj tony, zadavanie ukonci 0');
index:=1;
REPEAT
READLN(a[index]);
INC(index);
UNTIL a[index]=0 OR index=21;
FOR i:=1 to index-1 do BEGIN
                        SOUND(a[index]);
                        DELAY(200);
                       END;
NOSOUND;
```

Predchádzajúci príklad ukazuje použitie jednorozmerného poľa. Pascal však dovoľuje zadefinovať aj viacrozmerné pole (v princípe si môžeme zadefinovať n rozmerné pole). Ak k prvkom jednorozmerného poľa sme pristupovali pomocou jedného indexu, tak na prvky dvojrozmerného poľa sa budeme odvolávať pomocou dvoch indexov. Dvojrozmerné pole si môžeme predstaviť ako tabuľku, ktorá má niekoľko stĺpcov (**s**) a niekoľko riadkov (**r**). V každom okamihu výpočtu môžeme pracovať s ľuboľným prvkom poľa (nech je to prvok v stĺpci **i** a riadku **j**).



Definícia dvojrozmerného poľa môže vyzerať nasledovne:

```
TYPE dpole=ARRAY[1..10,1..10] OF INTEGER;
VAR a:dpole;
```

Pri používaní dvojrozmerného poľa je potrebné dávať si pozor na to, čo nám reprezentuje riadky a čo nám reprezentuje stĺpce. Podľa predchádzajúceho obrázku platí a[i,j]=5, ale a[j,i]=1 (v tomto prípade platí i=2 a j=3). Načítanie prvkov poľa a ich nasledovný výpis vyzerá nasledovne:

```
WRITELN('Zadaj pocet stlpcov a riadkov');
READLN(s,r);
WRITELN('Zadavaj prvky pola');
FOR y:=1 TO r DO BEGIN
                  FOR x:=1 TO s DO READ(a[x,y]);
                  READLN;
                 END;
WRITELN('Vypis prvkov pola:');
FOR y:=1 TO r DO BEGIN
                  FOR x:=1 TO s DO WRITE(a[x,y]);
                  WRITELN;
                 END;
```

Typ záznam: niekedy potrebujeme v počítači reprezentovať údaje, ktoré síce nie sú rovnakého typu, ale istým spôsobom patria k sebe. Napríklad reprezentácia informácií o ľuďoch. Každý človek môže byť reprezentovaný nasledujúcimi údajmi:

meno : string[20]
priezvisko : string[20]
pohlavie : (muz,zena)
ak pohlavie=muz tak vojna : boolean
ak pohlavie=zena tak rodena : string[20]

Typ záznam, **record**, obsahuje definovaný počet položiek, hovoríme o položkách záznamu. Položky môžu byť rôznych typov. Pri definícií typu záznam sa musí definovať identifikátor a typ každej premennej. Definícia premenných môže mať aj premenlivú časť. V našom prípade, ak pohlavie má hodnotu muz, tak záznam obsahuje položku vojna (áno znamená absolvoval, nie znamená že ho to ešte čaká) a ak pohlavie má hodnotu zena, tak záznam obsahuje položku rodená (meno za slobodna). Všimnite si ešte (ďalej v texte), že časť záznamu: `a.pohlavie`, `a.vojna` nemôžeme načítať priamo, ale pomocou premennej `pom`. Typ premennej `pohlavie` je **vymenovaný typ**, kde pri deklarácií vymenujeme množinu prípustných hodnôt.

```
type clovek = record
              meno:string[20];
              priezvisko : string[20];
              case pohlavie:(muz,zena) of
                           'muz' : (vojna : boolean);
                           'zena' : (rodena : string[20]);
              end;
var a : clovek;
    pom : string;
begin
writeln('zadaj meno');
readln(a.meno);
writeln('zadaj priezvisko');
readln(a.priezvisko);
writeln('zadaj pohlavie');
readln(pom);
if pom='muz' then begin
                   a.pohlavie  := muz;
                   writeln('Absolvoval vojnu');
                   readln(pom);
                   if pom=’ano’ then a.vojna:=true
                                else a.vojna:=false;
                  end
             else begin
                   a.pohlavie := zena;
                   writeln('zadaj rodena');
                   readln(a.rodena);
                  end;
```

Typ množina: pascal vie pracovať aj s dátovými objektmi, ktorých hodnoty sú množiny. Typ množina je zložený z ordinárnych typov. Množina nesmie mať viac ako 255 prvkov. V rámci triedy množina sú definované množinové a relačné operácie:

`A+B <=>` zjednotenie množín A a B
`A-B <=>` rozdiel množín A a B
`A*B <=>` prienik množín A a B

`A=B <=>` rovnosť množín A a B
`A<>B <=>` nerovnosť množín A a B
`A<=B <=>` A je podmnožinou B
`A>=B <=>` A obsahuje B
x in A `<=>` prvok x je prvkom A

Medzi jednoduché prípady použitia množín patrí vyjadrovanie podmienok príslušnosti nejakej hodnoty do určitej množiny hodnôt, kde dlhé logické výrazy nahradím reláciou s operátorom **in**.V praxi môže použitie množín vyzerať nasledovne:

```
USES crt;
TYPE znaky = SET OF char;
CONST velke : znaky = ['A'..'Z'];
      pismenko : znaky = ['A'..'Z'] +['a'..'z'];
      cislica : znaky = ['0'..'9'];
VAR a : char;
BEGIN
WRITELN('Stlac lubovolny klaves');
a:=READKEY;
IF a IN velke THEN WRITELN('Velke pismenko');
if a IN pismenko-velke THEN WRITELN('Male pismenko');
IF a IN cislica+pismenko THEN WRITELN('Cislica, pismenko');
END.
```

Typ súbor: Je síce pekné, ak program spracuje množstvo dát, vyhodnotí údaje a poskytne výsledky. Ako náhle počítač vypneme, výsledky sú nenávratne preč. Bolo by teda vhodné mať možnosť, uchovať výsledky aj po vypnutí počítača. A na to sú stvorené súbory, súhrny informácií uložené na pevnom disku počítača, ktoré sa uchovávajú aj po vypnutí počítača.
Turbo Pascal rozpoznáva celkom tri druhy premenných typu súbor (**file**). Sú to súbory **s udaným typom**, **bez udaného typu** a súbory **typu text**. Ukážme si, ako sa pracuje so súborom s udaným typom a so súborom typu text. Súbor s udaným typom pracuje s položkami, ktoré sú všetky rovnakého typu (integer, real, record...). To znamená, že zápis ale aj čítanie so súbory prebieha vždy po položkách. Takýto súbor je bez toho aby sme niečo vedeli o type položky v podstate nečitateľný (napr. v Norton Commanderi príkazom F3). Súbor typu text obsahuje textové informácie (premenná typu string je definovaná ako array[0..255] of char). Tento súbor sa dá veľmi jednoducho prezerať aj v NC. Nasledujúci program demonštruje prácu so súbormi:

```
uses crt;
var c:file of integer;
    t:text;
var cislo:integer;
    retazec:string;
    znak : char;
begin
assign(c,'subor.num'); rewrite(c);
assign(t,'subor.txt'); rewrite(t);
repeat
 writeln('Zadaj cele cislo'); readln(cislo);
 write(c,cislo);
 writeln('Dalsi vstup? a/n'); znak := readkey;
until znak='n';
repeat
 writeln('Zadaj text'); readln(retazec);
 writeln(t,retazec);
 writeln('Dalsi vstup? a/n'); znak := readkey;
until znak='n';
close(t);
append(t);
writeln(t,'Text na konci suboru');
writeln('Pripojenie na koniec suboru');
repeat
 writeln('Zadaj text');readln(retazec);
```

```
 writeln(t,retazec);
 writeln('Dalsi vstup? a/n'); znak := readkey;
until znak='n';
reset(c);
reset(t);
writeln('Obsah subor.num:');
while not eof(c) do begin
                     read(c,cislo);
                     writeln(cislo);
                    end;
writeln('Obsah subor.txt:');
while not eof(t) do begin
                     read(t,znak);
                     write(znak);
                     if eoln(t) then begin
                                     readln(t);
                                     writeln;
                                     end;
                    end;
close(t); close(c);
end.
```

PROCEDÚRY a FUNKCIE: každý zložitejší mechanizmus je potrebné riadiť hierarchickým spôsobom. Hierarchia spočíva v tom, že horné vrstvy (šéfovia, vedúci, velitelia) sa zaoberajú všeobecnými problémami a informáciami a dolné vrstvy (podriadení, robotníci, vojaci) sa zaoberajú detailmi. Generál vie ako vyhrať boj, ale nie veľmi ho zaujíma ako ženista postaví most cez rieku. Jedno však spraviť musí: prikázať ženistovi "**postav_most**". Takže, náš generál začal používať príkaz: "**postav_most**". Všetko bolo v poriadku až do chvíle, keď sa mali vojaci preplaviť v mieste, kde tiekli dve rieky. Chudák ženista nevedel na ktorej rieke má most postaviť. Uvedomil si to aj generál a zmenil svoj príkaz na tvar "**postav_most cez rieku R**". Istú dobu to fungovalo, ale občas sklerotický ženista zabudol kde naposledy most staval a tak ho nemohol nájsť a postaviť znova na inom mieste. Generál porozmýšľal a vymyslel príkaz: "**postav_most na rieke R, most nájdeš na mieste M**"., kde M bolo miesto, kde sa most naposledy staval. Samozrejme že vždy keď sa postavil most na novom mieste, zmenila sa aj hodnota M, takže generál mal vždy prehľad o tom, kde sa most staval naposledy. Zdalo sa, že všetko bude v poriadku, ale nebolo to tak. Je rozdiel v tom, či je most postavený na pokojnej malej riečke, alebo na veľkej rýchlo tečúcej rieke. Ak bol most na malej riečke, mohlo cez most prejsť viac vojakov a rýchlejšie. Ak to bola veľká rieka, vojaci museli chodiť opatrne, po jednom, čo značne spomaľovalo presun. Generál preto vydal nariadenie, podľa ktorého príkaz pre stavbu mosta cez rieky znie: "**postav most na rieke R, most nájdeš na mieste M, podaj hlásenie ako rýchlo sa vojaci môžu po moste prepravovať, P**". Ak ženista dostal tento príkaz, vedel na ktorej rieke (R) má most postaviť, vedel kde bol most postavený naposledy (M), takže ho rýchlo našiel, navyše podal generálovi hlásenie o priepustnosti mosta (P). Ak ženista vykonal príkaz, vedel generál kde je novopostavený most a akú má priepustnosť.

Presne tak to funguje aj v programovaní, ale nestaviame mosty a nie je tu generál a ženista, ale počítame hodnoty premenných podľa istých vzťahov, je tu hlavný program a podprogramy, ktoré nazývame **procedúry** a **funkcie**. Ak by sme chceli generálove príkazy prepísať do jazyka Pascal mohlo by to vyzerať nasledovne:

1 príkaz:
```
procedure postav_most ;
begin
najdi most;
presun most;
postav most;
end;
```

2 príkaz:
```
procedure postav_most(R);
begin
najdi most;
presun most k rieke R;
postav most;
end;
```

3 príkaz:
```
procedure postav_most(R,var M);
begin;
chod na miesto M;
presun most k rieke R;
postav most;
zmen hodnotu M;
end;
```

4 príkaz
```
function postav_most(R,var M):P;
begin
chod na miesto M;
presun most k rieke R;
postav most;
zmen hodnotu M;
podaj hlasenie o priepustnosti mosta P;
end;
```

Procedúry a funkcie tvoria postupnosti inštrukcií, ktoré potrebujeme v programe opakovať viackrát a na rôznych miestach. Procedúry a funkcie musia byť deklarované v časti deklarácií procedúr a funkcií. Po deklarácii môže byť procedúra alebo funkcia použitá kdekoľvek v nasledujúcom bloku programu. Rozdiel medzi procedúrou a funkciou je v tom, že funkcia vracia hodnotu a môže sa použiť priamo vo výrazoch. Procedúra sa vyvolá príkazom volania procedúry a vykoná jednu alebo viac inštrukcií. Všimnite si kapitolu "Štandardné procedúry a funkcie Pascalu". Príkazy, ktoré bežne používate, sú procedúry a funkcie, ale už sú dopredu zadeklarované. Nemusíte teda vedieť ako pracujú, ale čo robia. Procedúry a funkcie môžu byť bez parametrov (príkaz 1), s parametrami volanými hodnotou (príkaz 2), s parametrami volanými odkazom (príkaz 3: var M). Samozrejme, že je možná ich ľubovoľná kombinácia.

Pr.: Vytvorme opäť program, ktorý hľadá korene kvadraticke rovnice, a použime procedúry a funkcie.

```
var a,b,c,d:real;
function determinant(kvad,lin,abs:real):real;
begin
determinant:=sqr(lin)-4*a*c;
end;
function ma_riesenie(x:real):boolean;
begin
if x<0 then ma_riesenie:=false
       else ma_riesenie:=true;
end;
procedure korene(kvad,lin,det:real);
var x1,x2:real;
begin
x1:=(-lin+sqrt(det))/(2*kvad);
x2:=(-lin-sqrt(det))/(2*kvad);
writeln('Korene rovnice su: ',x1,' ',x2);
end;

procedure vstup(var kv,li,ab:real);
begin
writeln('Zadaj koeficienty kvadratickej rovnice: ');
readln(kv,li,ab);
end;

begin
vstup(a,b,c);
```

```
d:=determinant(a,b,c);
if ma_riesenie(d) then korene(a,b,d)
                  else writeln('Rovnica nema riesenie');
end.
```