ZÁKLADY PROGRAMOVACÍHO JAZYKA TURBO PASCAL

Obsah

Úvod

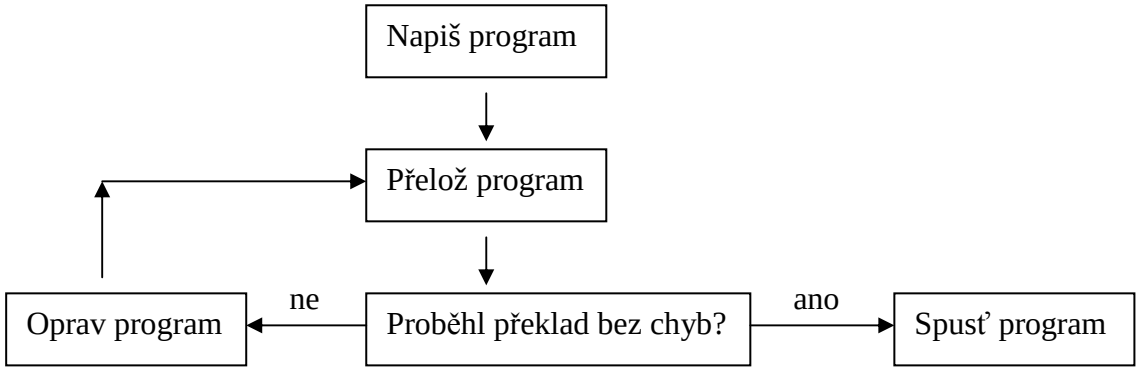
Pascal byl navržen v roce 1971 profesorem Niklausem Wirthem na počest Blaire Pascala, francouzského matematika. Návrhem jazyka prof. Wirth sledoval:

- vytvořit jazyk vhodný pro výuku programování,
- navrhnout strukturu jazyka tak, aby jej bylo možno jednoduše implementovat na většině počítačů.

Turbo Pascal je velice úspěšný dialekt standardního Pascalu navržený firmou Borland.

Zpracování programu počítačem

Zpracování programu počítačem můžeme znázornit diagramem:

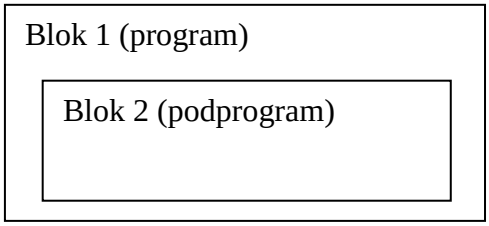


Program v jazyku Pascal se do počítače zapisuje konversačním způsobem tzv. editováním. K tomu slouží program nazvaný editor. Text programu se nazývá zdrojový program. Po zapsání programu se musí spustit překladač Pascalu. Při překladu se obvykle odhalí chyby, které se musí opravit. Vrátíme se zpět k editování a opravíme zdrojový program a znovu jej přeložíme. Pokud překlad proběhne bez chyb, je možné program spustit. Při spuštění programu mohou nastat další chyby – tzv. chyby běhové.

Je-li program přeložen bez chyb a nejsou-li hlášeny chyby běhové, je třeba se přesvědčit, zda pro různé vstupní údaje dává program očekávané výsledky – **ladění** programu. Vytvoříme vzorky vstupních údajů, pro které určíme předem údaje výstupní. Jsou-li výsledky programu chybné nebo vznikne-li běhová chyba, je třeba opravit zdrojový kód programu, spustit překladač a opět ladit program. Odladěný program se uloží na disk a je připraven k dalšímu využití.

Turbo Pascal je integrované vývojové prostředí, integruje v sobě několik částí – editor, překladač, ladící prostředky a nápovědný systém.

Struktura a zápis programu v jazyku Pascal



Jazyk Pascal patří mezi jazyky s blokovou strukturou programu. Každý blok obsahuje nejprve deklarace a definice všech objektů, které se v bloku vyskytují a pak následují příkazy. Charakteristickou vlastností bloku je, že může obsahovat v sobě další blok nebo více bloků – tzv. vnořené bloky, viz obrázek níže.

Příklad jednoduchého programu:

```
Program Nasobeni;            {1}
const pi=3.14;               {2}
var a:real;                  {3}
begin                        {4}
      read(a);               {5}
      write(a*pi);           {6}
end.                         {7}
```

Řádek {1} je jeho záhlaví. Určuje, že program se jmenuje Nasobeni. Řádek {2} obsahuje deklaraci konstanty a řádek {3} obsahuje deklaraci proměnné. Konstantu označujeme **pi** a proměnnou **a**. Konstanta bude mít hodnotu 3.14 a proměnná a bude reálné číslo (real – reálné). Vlastní program začíná na řádku {4}. Na dalším řádku je příkaz read pro načtení hodnoty do proměnné a. Poté příkazem write program vypíše součin konstanty pi a proměnné a.

Čísla ve složených závorkách tvoří komentář, jsou určena pouze jako pomůcka pro jeho popis. Komentář můžeme taktéž zapsat mezi znaky **(*** a ***)**.

Každý program musí obsahovat následující části:

1) hlavičku – řádek 1
2) deklarační část – řádky 2 a 3
3) příkazovou část – řádky 4 až 7

Hlavička programu začíná vždy klíčovým slovem program, za kterým následuje jméno programu. Hlavička je ukončena středníkem

Deklarační část může dle potřeby obsahovat pět úseků deklarací (v uvedeném pořadí):

1) úsek deklarací návěští
2) úsek deklarací konstant (viz příklad výše)
3) úsek deklarací typů
4) úsek deklarací proměnných (viz příklad výše)
5) úsek deklarací funkcí a procedur.

Jednotlivé úseky jsou opět odděleny středníkem.

Hlavička programu nemusí být uvedena vůbec, protože má pouze dokumentační charakter.

> Po hlavičce programu následuje seznam použitých programových jednotek. Ten je uveden za klíčovým slovem **uses**. Jednotlivé položky jsou odděleny čárkou a seznam je ukončen **středníkem**. Není li použita žádná jednotka, vše se vynechává. Příklad:
> `uses crt;`

Po této části následují definice a deklarace návěští, konstant, typů, proměnných, funkcí a procedur. Tyto úseky nemají pevné pořadí a mohou být dokonce i několikrát opakovány. Vždy však musíme dodržet pravidlo, že před každým použitím syntaktického objektu musí být tento objekt ji definován nebo deklarován. Výjimku má pouze definice typu ukazatel (o té ale mnohem později).

Po deklarační části následuje složený příkaz, kteří tvoří tělo programu. Za závěrečným **end** je uvedena tečka. (**begin … end.**)

Program zapsaný v Pascalu je z formálního hlediska text vytvořený podle tzv. syntaktických pravidel jazyka. Tato pravidla stanoví, jaké symboly lze v textu použít a jak tyto symboli v rámci programu skládat (řadit). Mezi symboly patří:

1) písmena, tj. znaky velké i malé abecedy,
2) číslice 0 až 9,
3) speciální symboly + – . , ‘ ( ) / * <= >= <> = := .. [ ] { } a klíčová slova.

Klíčová slova jsou: and, array, asm, end, case, const, constructor, destruktor, div, do, downto, else, end, exports, file, for, function, goto, if, implementation, in, inherited, inline, interface, label, library, mod, nil, not, object, of, or, package, procedure, program, rekord, repeat, set, shl, shr, string, then, to, type, unit, until, uses, var, while, with, xor.

Číselný datový typ (integer a real)

Čísla se zadávají i vypisují s desetinnou tečkou místo obvyklé desetinné čárky. Pro zápis čísla v exponenciálním tvaru používáme pro oddělení mantisy a exponentu písmeno E. Mantisou může být buď celé číslo nebo číslo desetinné.

V programech pracujeme s čísly typu real – racionální čísla v desetinném tvaru (např. 12.5 -81.6 0.7 5.7E9) a s čísly typu integer – čísla celá (např. 13 -48 123)

Příklad:

1E6 = 1 000 000 2.2E3 = 2 200

Konstanty

1) konstanty si můžeme v programu pojmenovat – viz příklad s pi
2) nebo můžeme konstantu v programu přímo využít (taková konstanta se nazývá **literál**), viz příklad níže:

```
program Pouziti_konstanty;
begin
      write(3.5 * 4);
end.
```

Výhodou pojmenovaných constant proti literálům je to, že se zvýší čitelnost zapsaného programu a hlavně to, že změnu hodnoty konstanty můžeme provést pouze jednou změnou její deklarace, na jediném místě programu.

Proměnné

Algoritmus by měl být použitelný pro různá data na vstupu. Z toho důvodu používáme v programu proměnné. Obsah proměnné závisí na konkrétní hodnotě přečtené ze vstupu nebo

hodnotě přiřazené v průběhu chodu programu.

Stejně jako konstanty musí být proměnná před svým prvním použitím deklarována. Tím se určí její typ a také množina přípustných hodnot, které do ní mohou být ukládány. Typem proměnné jsou současně určeny i operace, které lze pro tuto proměnnou předepisovat.

```
A: real
```

Sdělujeme počítači, že budeme v programu pracovat s veličinou **A**, jejímž obsahem bude reálné číslo. Deklarace je počítačem akceptována jednak jako příkaz k vymezení prostoru v paměti, jednak jako příkaz k pojmenování tohoto místa jménem A a dále k zajištění přístupu obsahu tohoto paměťového místa, kdykoliv napíšeme jeho jméno, tj. **A**.

Po deklaraci je obsah proměnné nedefinován až do okamžiku, kdy proměnné přiřadíme nějaký obsah. To lze dvěma způsoby:

- přečtením hodnoty do proměnné (ze vstupu, příkaz **read** nebo **readln**)
- dosazením hodnoty do proměnné – **přiřazovací příkaz**, např. A:=7 (dosazení sedmičky do proměnné **A**)

Aritmetické operátory a aritmetické standardní funkce:

Operátor	Operace	Typ operandu	Typ výsledku
+	Sčítání	I nebo R	I nebo R
–	Odčítání	I nebo R	I nebo R
*	Násobení	I nebo R	I nebo R
/	Dělení	I nebo R	Vždy R
Div	Celočíselné dělení	I	I
Mod	Zbytek po celočíselném dělení	I	I

I = Integer, R = real

Celočíselné dělení

Celočíselné dělení dvou čísel typu Integer provádíme pomocí operátoru **div**. Dává vždy celočíselný výsledek, i když čísla nejsou beze zbytku dělitelná. Výsledkem je pouze celá část podílu. Pro čísla typu real **není** celočíselné dělení definováno (např. 25 div 4 = 6)

Zbytek po celočíselném dělení

Operátor **mod** předepisuje operaci, jejímž výsledkem je zbytek po celočíselném dělení. Tato operace může být předepsána pouze pro operandy typu Integer, z nichž druhý musí být větší než 0. Pro nezáporné **A** a kladné **B** platí:

```
A mod B = A – (A div B) * B (např. 25 mod 4 = 1)
–A mod B = –A – (–A div B) * B (např. –25 mod 4 = –1)
```

Aritmetické standardní funkce:

Funkce	Význam	Typ argumentu	Typ výsledku
Abs(x)	Absolutní hodnota x	I nebo R	Jako x
Sqr(x)	Druhá mocnina x	I nebo R	Jako x
Sqrt(x)	Druhá odmocnina x	I nebo R	R
Ln(x)	Přirozený logaritmus x	I nebo R	R
Exp(x)	e^x	I nebo R	R
Sin(x)	Sinus x (x v radiánech)	I nebo R	R
Cos(x)	Cosinus x (x v radiánech)	I nebo R	R
Arctan(x)	Arcus tangens x	I nebo R	R

a) celočíselné typy:

Typ	Rozsah	Zabírá B v paměti
Byte	0 .. 255	1
Shortint	–128 .. 127	1
Word	0 .. 65535	2
Integer	–32768 .. 32767	2
longint	–2147483648 .. 2147483647	4

b) typy reálných čísel:

Typ	Rozsah	Zabírá B v paměti
Real	$2{,}9*10^{-39}$ .. $1{,}7*10^{38}$	6
Single	$1{,}5*10^{-45}$ .. $3{,}4*10^{38}$	4
Double	$5{,}0*10^{-324}$ .. $1{,}7*10^{308}$	8
Extended	$3{,}4*10^{-4932}$ .. $1{,}1*10^{4932}$	10
Comp	$-2^{63}+1$ .. $2^{63}-1$	8

Příkazy

- jednoduché
  - přiřazovací příkaz
  - příkaz procedury
  - příkaz skoku
  - prázdný příkaz
- strukturované
  - složený příkaz
  - podmíněný příkaz

- příkaz if
- příkaz case

- příkaz cyklu
  - příkaz repeat
  - příkaz white
  - příkaz for
- příkaz with

Příkazy vstupu a výstupu

```
Read(x)
Read(x1, …xn)
```

kde x proměnná (či proměnné). Kolik znaků se tímto příkazem přečte a jak se přečtená posloupnost znaků zpracuje, závisí na typu proměnné:

- je-li proměnná typu **char**, přečte se jediný znak a jeho hodnota se přiřadí proměnné. Přečte-li se přitom oddělovač řádků, do proměnné se uloží znak mezera.
- Je-li proměnná typu **integer**, přečte se posloupnost znaků tvořících dekadický zápis celého čísla s připadným znaménkem, vytvoří se příslušná hodnota typu integer a ta se přiřadí proměnné. Libovolný počet mezer a oddělovačů řádků se při čtení ignorují.
- Je-li proměnná tyty **real**, přečte se posloupnost znaků tvořících dekadický zápis čísla s případným znaménkem, vytvoří se příslušná hodnota typu real a ta se přiřadí proměnné. Libovolný počet mezer a oddělovačů řádků se při čtení ignorují.

Příkaz **readln** (read line = číst řádku) má podobné vlastnosti jako **read**, odlišnost spočívá v tom, že po načtení údaje do poslední proměnné je připraveno čtení z nového řádku.

```
Write(p)
Write(p1, …, pn)
```

Parametry mohou mít jeden z následujících tvarů:

H

H:PZ

H:PZ:PDM

H – vstupní hodnota

PZ – počet znaků výstupu

PDM – počet desetinných míst výstupní hodnoty

Pro příkaz **write** platí následující pravidla:

- není-li počet znaků výstupu určen, vystoupí v případě typu char a řetězec celkem tolik znaků, kolik jich obsahuje hodnota výrazu. Pro ostatní typy je počet znaků stanoven implementací. Hodnota typu **real** vystoupí v exponenciálním tvaru.
- Je-li počtem znaků PZ určen větší počet než je nutné, doplní se vystupující hodnoty zleva mezerami.
- Je-li počtem znaků PZ určen menší počet než je nutné, vystoupí v případě typů integer a real minimální nutný počet znaků a z hodnot ostatních typů pouze zadaný počet znaků zleva. Hodnota typu real vystoupí v exponenciálním tvaru.
- Hodnota typu real vystoupí v pevné řádové čárce jen v případě, že je určen počet desetinných míst PDM.
- Hodnota true vystupuje jako řetězec „true“ a hodnota typu false jako řetězec „false“

Příkaz **writeln** (write line = psát řádku) je podobný jako **write**. Rozdíl je v tom, že zapíše parametry a přidá označení konce řádku hned za poslední znak.

Má-li být vypsán text, zapíše se do apostrofů (např. write(‘Ahoj svete!‘);)

Složený příkaz

Složený příkaz má tvar:

```
Begin
      Prikaz 1;
      Prikaz 2;
      …
      Prikaz n;
End.
```

Je tvořený posloupností příkazů, které jsou odděleny středníkem. Posloupnost je uzavřena do příkazových závorek **begin** a **end**. Počet příkazů v posloupnosti není omezen. Složený příkaz se používá všude tam, kde je dovoleno zapsat pouze jeden příkaz.

Snažíme se v Pascalu psát zdrojový program do co nejpřehlednější formy.

Podmíněné příkazy

Příkaz if

- alternativní postupy v programu, testování podmínky a na základě jejího vyhodnocení pokračování v příslušné části programu.

Příkaz **if – then – else** (jestliže – pak – jinak) umožňuje předepsat v programu výběr jedné ze dvou možností. Je-li podmínka splněna, provede se postup zapsaný za klíčovým slovem **then**, není-li splněna, provede se postup zapsaný za klíčovým slovem **else**. Za klíčovým slovem **then** a **else** smí být zapsán **pouze jeden** příkaz (v opačném případě musíme použít příkaz složený).

```
program obvod;
var r, obvod: real;
begin
      readln(r);
      If r<0 then writeln(‘chybne zadany polomer’)
            else  begin
                        obvod:=2*pi*r;
                        writeln(‘obvod = ’,obvod);
                  end;
end.
```

Je možné vložit několik podmíněných příkazů do sebe, např.:

```
program maximum;
var a, b, c: integer;
begin
      readln(a,b,c);
      if a>b then
            If a>c then writeln(a)
                  else writeln(c)
            else
                  If b>c then writeln(b)
                        else writeln(c);
end.
```

Příkaz case

Selektivní příkaz case je zobecnění příkazu **if**. Provede jeden z většího počtu alternativních příkazů v závislosti na hodnotě předepsaného výrazu. Poslouží všude tam, kde se ve vývojovém diagramu objeví vícenásobné větvení programu.

Obecný zápis příkazu case:

```
case V of
      K1:P1;
      K2:P2;
      …
      Kn:Pn
    else Pn+1;
end;
```

Část **else** nemusí být použita. Příkaz **case** je ukončen klíčovým slovem **end** – v tomto případě není end do páru s klíčovým slovem begin.

V – výraz ordinálního typu – tzv. selektor větvení

K – konstanta ordinálního typu (hodnota výrazu V)

P – příkaz

Příkaz se provede tak, že se nejdříve vyhodnotí výraz V, který musí být ordinálního typu a jehož všechny hodnoty musí být uvedeny v seznamu konstant K1, …, Kn. Počítač provede příkaz, který je předznačen hodnotu rovnající se hodnotě výrazu V. Každá konstanta může předznačovat pouze jeden příkaz. Naopak jeden příkaz může být předznačen více konstantami. Je to zkrácený zápis pro případ, že příkaz se má provést pro několik hodnot výrazu V.

```
program prikaz_case;
var cislo: integer;
begin
      writeln('Zadej cislo od 1 do 10.');
      read(cislo);
      case cislo of
            1,3,5,7,9:writeln('Cislo je liche.');
            2,4,6,8,10:writeln('Cislo je sude.');
         else writeln('Bylo zadano jine cislo.');
      end;
end.
```

Příkaz skoku

Příkazem skoku se udává návěští, od něhož bude provádění programu pokračovat. Má tvar:

```
Goto n;
```

kde n je návěští. Návěští jsou od příkazů oddělena dvojtečkou a musí být v úseku deklarace návěští deklarována. Číselným návěštím mohou být pouze celá čísla bez znaménka zpravidla nejvýše čtyřciferná (0 … 9999).

Příklad použití návěští:

```
Program skok;
label 12;
…
begin
      …
      if B then goto 12;
      …
      12: prikaz;
      …
end.
```

Používání příkazu skoku je v Pascalu velmi omezeno. Je to dáno požadavkem dobře strukturovaných programů. V podstatě lze příkaz **goto** použít jen v těchto případech:

- pro předčasné ukončení cyklu typu **for** a skok buď na konec cyklu nebo opuštění cyklu a skok až za jeho konec,
- předčasné ukončení některého strukturovaného příkazu,
- předčasné ukončení procedury nebo funkce.

Příkazem skoku nesmí být předáno řízení z vnějšku, dovnitř strukturovaného příkazu (jedná se o příkazy **if, case, repeat, while, for** a **složené příkazy**). Řídící struktury Pascalu jsou však definované tak, že se dá každý program zapsat bez příkazu skoku. Proto se příkaz skoku v programech **nepoužívá** a zde je uveden pouze jako existující možnost.

Cykly (opakování postupu v programu)

Doposud jsme po spuštění programu zadávali data pouze jednou. Pokud jsme chtěli zadat nová data, bylo nutné program opět spustit. Požadujeme-li však několikanásobné opakování

celého programu nebo jeho části, je takový postup nevýhodný a pracný. Použijeme proto příkazu cyklu, který způsobí automatické opakování programu nebo jeho části, a to tolikrát, kolikrát je zadáno.

Cyklus while

```
program cyklus1; {Program napise 10krat Dobry den.}
var a: integer;
begin
      a:=0;
      while A<10 do
            begin
                  writeln('Dobry den');
                  a:=a+1;
            end;
end.
```

V programu je použita konstrukce **while – podmínka – do** (dokud platí – podmínka – dělej). Ta zajišťuje opakování příkazů tak dlouho, dokud je splněna podmínka mezi klíčovými slovy **while** a **do**. Jakmile tato podmínka splněna není, opakované provádění složeného příkazu, uvedeného za klíčovým slovem **do**, končí. Cyklus **while** se vyznačuje tím, že pokud již před začátkem cyklu podmínka není splněna, cyklus vůbec neproběhne.

Cyklus for

```
program aritmeticky_prumer;
var   pocet,i: integer;
      soucet, a: real;
begin
      write('Zadej pocet cisel:');
      readln(pocet);
      soucet:=0;
      for i:=1 to pocet do
            begin
                  write('Zadej hodnotu ',I,' cisla: ');
                  readln(a);
                  soucet:=soucet+a;
            end;
      writeln('Aritmeticky prumer = ',soucet/pocet);
end.
```

Cyklus **for** (pro) předepisuje postupné ukládání určených hodnot jdoucích po jedničce po sobě, zadaných první a poslední (koncovou) hodnotou do řídící proměnné. V našem případě je první hodnotou jednička, poslední je uložena v proměnné počet a řídící proměnná je **i**. Příkaz za klíčovým slovem **do** se tedy provede počet-krát. Při použití klíčového slova **downto** se hodnota řídící proměnné postupné o jedničku zmenšuje (např. `for i:=10 downto 1 do …`)

Cyklus repeat

Třetí a poslední cyklus se nazývá **repeat**. Je stejně jako cyklus **while** řízen libovolným logickým výrazem. Má tvar:

```
repeat
      Prikaz 1;
      …
      Prikaz n;
until podmínka;
```

Příkazy uzavřené mezi klíčovými slovy **repeat** (opakuj) a **until** (dokud ne) tvoří složený příkaz a představují tělo cyklu. Tělo cyklu se provádí tak dlouho, dokud není splněna podmínka za klíčovým slovem **until**. Jakmile je podmínka splněna, provádění cyklu se zastaví. Z konstrukce cyklu je patrno, že cyklus proběhne vždy alespoň jednou.

```
program hvezdicka;
var pocet: integer;
Bbgin
      write('Kolikrat opakovat? ');
      readln(pocet);
      repeat
            write('*');
            pocet:=pocet-1;
      until pocet<=0;
      repeat
      until keypressed;
end.
```

V programu je použito dvou cyklů **repeat**. Druhý cyklus, **repeat until keypressed** probíhá tak dlouho, dokud nestiskneme nějakou klávesu. Potom je ukončen. V našem případě jej používáme k tomu, aby na monitoru zůstaly po spuštění programu zobrazené výsledky a my je nemuseli znovu vyvolávat stiskem kláves **Alt-F5**.

```
program hvezdicka_2;
var   Konec: char;
begin
      repeat
            writeln('*');
            writeln('Další hvězdička? (A/N)');
            Konec:=ReadKey;
      until (konec='n') or (konec = 'N');
end.
```

V tomto programu nevíme kolikrát budeme chtít hvězdičku tisknout. Program hvězdičku vytiskne a zeptá se nás, jestli chceme tisknout další. Pokud nestiskneme klávesu **N** (musíme ale stisknout libovolnou jinou) vytiskne program hvězdičku.

Logický datový typ

Každé dvě hodnoty datových typů real a integer je možné porovnat. V Pascalu můžeme použít tyto relační operátory:

=	test na rovnost
<>	test na nerovnost
<	menší
>	větší
<=	menší nebo rovno
>=	větší nebo rovno

Výsledkem porovnání je výrok, zda je tvrzení pravdivé (true) nebo nepravdivé (false). V tabulkách se pravdivost nejčastěji označuje znakem 1 (jedna) a nepravdivost znakem 0 (nula).

Datový typ mající pouze dvě hodnoty (true a false) je v Pascalu na počest matematika Georga Boolea pojmenován **boolean**.

Příklad deklarace logických proměnných:

```
Var x,y: boolean;
```

Takto deklarované proměnné mohou nabývat pouze dvou hodnot: **true** nebo **false**, přičemž platí: false < true. Hodnoty proměnných lze i vypisovat – vypíše se slovo **true** či **false**.

Pro použití v logickém výrazu jsou definovány tři operace s operátory **not** (negace), **and** (logický součin) a **or** (logický součet), výsledky operací vidíte níže v tabulce:

X	Y	X and Y	X or Y	Not X	Not Y
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	1
1	1	1	1	0	0

Provedení všech operací předepsaných výrazem se nazývá **vyhodnocení výrazu**. Výsledkem vyhodnocení je **hodnota výrazu**. Jednotlivé operace předepsané výrazem se provádějí postupně zleva doprava s ohledem na prioritu operátorů a na závorky. Podle priority lze operátory rozdělit do čtyř skupin:

1. not nejvyšší priorita
2. *, /, div, mod, and
3. +, -, or
4. =, <>, <, > <=, >= nejnižší priorita

Prioritu operátorů je možno ovlivnit závorkami. Platí, že výrazy uvedené v závorce se vyhodnocují vždy jako první.

Správně: (A >= 0) and (A <= 9)
Špatně: A >= 0 and A <= 9 bude označen jako chyba

Znakový datový typ (char)

Znakovou konstantu představuje jeden znak uzavřený mezi apostrofy – např. ‘n’, ‘7’. Deklarace:

```
Var A1, B3, S: char;
```

Do takto deklarované proměnné můžeme uložit jeden znak – **literál** – např. `A1:='n';` sám apostrof se zapíše takto: `S:='''';` (dva krajní apostrofy ohraničují literál a dva prostřední vyjadřují jediný apostrof). Více znaků tvoří řetězec, ten ale musíme ukládat do jiného typu proměnné (string), ale o tom až později.

Jednoduché datové objekty, jejich typy a možné operace

Každý datový typ určuje nejen množinu přípustných hodnot, ale současně také množinu operací, které je možno s datovými objekty daného typu provádět.

Typy jednoduchých datových objektů

Jednoduché typy rozdělujeme do dvou tříd:

1) ordinální typy (integer, boolean, char, výčet, interval)
2) typ real (neordinální)

Typy real, integer, boolean a char jsou předem definovány jazykem Pascal. Typy výčet a interval musíme zavádět (deklarovat) za klíčovým slovem **type** a označit je jménem. Jméno pak slouží k deklaraci tohoto typu.

Vlastnosti ordinálních typů

1. Všechny možné hodnoty ordinálních typů souvisí s pořadím – ordinalitou. S výjimkou hodnot typu integer první hodnota každého ordinálního typu má pořadí 0, další má pořadí 1 atp. U typu integer jsou hodnoty a pořadí shodné.
   V ordinálním datovém typu má každá hodnota svého **předchůdce** (predecessor) a **následovníka** (successor), kromě první či poslední hodnoty.
   Příklad:

6	**7**	**8**
Předchůdce		**následovník**
Pá	**So**	**Ne**

2. Standardní funkce ord nám udává pořadí hodnoty libovolného ordinálního typu.
3. Standardní funkce **pred** nám udává hodnotu předchůdce a může být použita pro hodnotu libovolného ordinálního typu.
4. Standardní funkce **succ** nám udává hodnotu následovníka a použijeme ji pro hodnotu libovolného řádového typu. Při použití pro poslední hodnotu ordinálního typu funkce **succ** hlásí chybu.

```
Program ordinal;
Uses crt;
Var znak:char;
Begin
      clrscr;
      writeln(‘Poradi znaku A az z’);
      writeln;
      for znak:=’A’to ‘z’ do
            Write(znak,’ = ’,ord(znak):1);
      writeln; writeln;
      writeln(’Zadej znak’); writeln;
      readln(znak);writeln;
      writeln(’pred(’,znak,’) = ’,pred(znak)); writeln;
      writeln(’succ(’,znak,’) = ’,succ(znak));
      repeat until keypressed;
end.
```

Typ interval

Z každého ordinálního typu umožňuje Pascal deklarovat jeho část – **interval**. Typ interval vymezuje souvislou část, která se při deklaraci určuje první a poslední hodnotou typu, z něhož se interval odvozuje.

```
Type int1 = 1..50;
int2 = ’a’..’z’;
```

zavádí dva typy interval pojmenované int1 a int2. Proměnné typu int1 zavedeme následovně

```
var a,b:int1;
```

potom proměnné **a** a **b** mohou nabývat celočíselných hodnot od 1 do 50 včetně. Proměnné **a** a **b** můžeme deklarovat též přímo a to deklarací:

```
var a,b: 1..50;
```

Typ výčet

Typ výčet je jednoduchý nestandardní datový typ. Vznikl z potřeby vytvořit typ jehož množina hodnot je malá.

Typ výčet tedy specifikuje uspořádanou množinu hodnot výčtem všech identifikátorů, které dané hodnoty reprezentují. Jejich uspořádání je dáno pořadím ve kterém jsou v popisu tyto hodnoty uvedeny, přitom prvnímu identifikátoru je přiřazeno pořadové neboli ordinální číslo 0.

Výčtový typ zavádíme většinou jako pojmenovaný typ, tedy v úseku `type`.

```
Type  den = (PO, UT, ST, CT, PA, SO, NE);
      predmet = (CJ, AJ, NJ, M, B, IVT);
      trida = (1A, 2A, 3A, 4A);
var   den_v_tydnu: den;
      hodina: predmet;
      mistnost: trida;
```

a je tedy možné přiřazení:

```
      mistnost:=1A;
```

Operace s hodnotami jednoduchých typů

Operace lze předepisovat

a) pomocí operátorů
b) s použitím standardních funkcí (např. funkce sinus)

Operátory jsou základní symboly jazyka s přesně vymezeným významem, mohou předepisovat operaci s jedním operandem (operátory unární) nebo se dvěma operandy (binární operátory).

Každý z operátorů může být použit pouze pro určitý typ operandů:

- aritmetické operátory
- operátory pro celočíselný typ
- logické operátory apod.

Standardní funkce pro argumenty jednoduchých typů

Typ argumentu standardní funkce	**Standardní funkce použitelné pro daný typ argumentu**	**Skupina funkcí**
Integer	succ pred ord char abs sqr sqrt ln exp odd	Ordinální Aritmetické logické
Real	abs sqr sqrt ln exp sin cos arctn trunc round	Aritmetické Transformační
Char	succ pred ord	Ordinální
Interval	Podle typu ze kterého je odvozen	

Ordinální standardní funkce

Funkce	**Succ(x)**	**pred(x)**	**Ord(x)**	**Chr(x)**
Význam	Následující hodnota	Předchozí hodnota	Ordinální číslo	Znak
Typ argumentu (x)	Libovolný ordinální	Libovolný ordinální	Libovolný ordinální	Integer
Typ funkcí hodnoty	Jako argument	Jako argument	Integer	Char

Funkce chr(x) je definována pro typ char. Je to inverzní funkce k funkci ord(x). Pro libovolný znak ZN platí:

```
chr(ord(ZN)) = ZN
```

Aritmetické standardní funkce

Viz kapitola 3.2.2

Logická standardní funkce odd

Standardní funkce **odd** testuje, zda argument je liché číslo (argument musí být typu integer). Je-li argument číslo liché, vrací logickou hodnotu **true**, v opačném případě vrací **false**.

Transformační standardní funkce

Tyto funkce slouží pro transformaci hodnot typu real na hodnoty typu integer.

Funkce **trunc** odstraňuje desetinnou část reálného čísla.

Příklad:

trunc(0.8) = 0 trunc(3.5) = 3

trunc (-0.8) = 0 trunc(-3.5) = -3

Funkce **round** zaokrouhluje argument podle běžných pravidel pro zaokrouhlování reálných čísel, to znamená:

round(x) = trunc(x + 0,5) pro x >= 0

round(x) = trunc(x – 0,5) pro x < 0

round(0.8) = 1 round(3.5) = 4

round(–0.8) = –1 round(–3.5) = –4

Programové jednotky

V Turbo Pascalu můžeme používat programové jednotky, z nichž pak můžeme využívat procedury a funkce. Zde se zmíníme pouze o některých jednotkách (a některých funkcích a procedurách), podrobněji vše probereme později.

Jednotka system

Procedura **dec** snižuje hodnotu proměnné

```
Dec(var X{; N:longint];
```

X – proměnná ordinálního typu

N – výraz typu longint

Příklad:

`Dec(x)`	odpovídá výrazu	`x:= x – 1;`
`Dec(x,n)`	odpovídá výrazu	`x:= x – n;`

Procedura **inc** zvyšuje hodnotu proměnné

```
inc(var X{; N:longint];
```

X – proměnná ordinálního typu

N – výraz typu longint

Příklad:

`Inc(x)`	odpovídá výrazu	`x:= x + 1;`
`Inc(x,n)`	odpovídá výrazu	`x:= x + n;`

Funkce **dec** a **inc** generují optimalizovaný kód, a proto je vhodné ji používat v cyklech místo výrazu `x:= x + 1;` či `x:= x + n;`

Funkce **Int** vrací celočíselnou část argumentu

```
Int(x:real):real
```

X – výraz typu real

Výsledkem funkce je celočíselná část parametru x.

Procedura **MkDir** vytvoří na disku podadresář

```
MkDir(S:string)
```

S – výraz typu řetězec, který definuje cestu přístupu a jméno nového adresáře

Funkce **Pi** vrátí hodnotu Pí (3.141592653589793285)

```
Pi:real
```

Výsledek funkce je typu real

Funkce **Random** vrací náhodné číslo

```
Random[(Rozsah:word)]:word;
```

Pokud se nedefinuje rozsah, je výsledek náhodné číslo typu real, které patří do intervalu 0 <= x < 1.

Pokud se rozsah uvádí, musí být výraz typu integer a výsledek je náhodné číslo typu word, které patří do intervalu 0 <= x < rozsah.

Pokud se uvede rozsah <= 0, funkce vrací hodnotu 0.

Generátor náhodných čísel se musí před použitím inicializovat vyvoláním **Randomize**.

Procedura **Randomize** inicializuje vestavěný generátor náhodných čísel náhodnou hodnotou. Náhodné číslo se získává ze systémových hodin.

```
Program nahodna_cisla;
Uses crt;
Begin
      Clrscr;
      Randomize;
      Repeat
            Write (Random,‘ ‘);
            Writeln(Random(10),‘ ‘);
            Writeln(Random(100),‘ ‘);
            Writeln(Random(1000),‘ ‘);
            Delay(200); {Pocka 200 milisekund}
            Writeln; writeln;
      Until keypressed;
End.
```

Funkce **UpCase** převede malé písmeno na velké

```
UpCase(znak:char):char
```

Znak je výraz typu char, výsledek je také typu char, znakové hodnoty, které nejsou v intervalu malých písmen se nepřevádějí a růstávají beze změny

Jednotka crt

Jednotka crt obsahuje programové prostředky pro řízení režimu zobrazovače, nastavení barev obrazu, vytváření okének, ovládání zvukového generátoru a klávesnice. Zde uveden pouze základ, bližší informace v manuálu.

Procedura **ClrScr** vymaže obsah okna a umístí kurzor do levého horního rohu.

```
ClrScr;
```

Procedura **Delay** pozastaví provádění programu na definovaný počet milisekund

```
Delay(Ms:word);
```

Ms – počet milisekund, čas se nedodrží zcela přesně, protože procedura čas zaokrouhlí na strojové cykly.

Procedura **GoToXY** nastaví polohu kurzoru v aktuálním okně

```
GoToXY(X,Y:byte);
```

X a Y definují polohu kurzoru, X poloha sloupce, Y poloha řádku uvnitř okna. Souřadnice levého horního rohu okna má souřadnice (1,1)

```
Program presun_kurzor;
Uses crt;
Begin
      Clrscr;
      GoToXY(10,20);
      Write('Ted budu psat do 10 sloupce a 20 radku.');
      Repeat until keypressed;
End.
```

Funkce **KeyPressed** vrací logickou hodnotu **True**, když se stiskla libovolná klávesa na klávesnici. Ve všech ostatních případech funkce frací logickou hodnotu False.

```
KeyPressed:boolean;
```

Funkce nereaguje na stisk rozšiřujících kláves Shift, Alt, NumLock, Ctrl, …

Stisknutý znak zůstane ve vyrovnávací paměti klávesnice a musí se vyjmout pomocí funkce **ReadKey** (jinak při opakovaném volání funkce **KeyPresse**d vrací neustále hodnotu True).

Funkce **ReadKey** přečte znak z klávesnice, přečtený znak nezobrazí na obrazovce

```
ReadKey:char;
```

Pokud měla funkce **KeyPressed** hodnotu **True** před vyvoláním funkce **ReadKey**, funkce vrací znak okamžitě. V opačném případě funkce čeká na stisk klávesy.

Procedura **Sound** spustí zvukový generátor.

```
Sound(Hz:word);
```

Parametr definuje kmitočet (Hz), na kterém bude generátor pracovat. Zvukový generátor pracuje, dokud se nevyvolá procedura **NoSound**.

Procedura **NoSound** vypne zvukový generátor.

Strukturované datové typy

Data, která mají určitou společnou vlastnost a jsou určitým způsobem organizována – říkáme, že tvoří struktury

1) **Pole** – je homogenní (stejnorodá) datová struktura, skládá se ze složek stejného typu. Složky se rozlišují pomocí indexů. Jako indesy složek slouží hodnoty určitého typu. Typ indexu spolu s typem složek pole je stanoven popisem typu pole.

2) **Záznam** – je nehomogenní (nestejnorodá) datová stuktura, která se skládá z určitého počtu pojmenovaných složek, tzv. položek záznamu. Příkladem může být osobní karta obsahující např. jméno, příjmení, datum narození apod.
3) **Soubor** – množina dat uložených mimo operační paměť počítače (např. na disku)
4) **Množina** – je strukturovaný datový objekt, jehož hodnotami jsou množiny.

Typ pole

Jednorozměrné pole

Mějme pole nazvané A se složkami $A_1 = 5$, $A_2 = 3$, $A_3 = 7$, $A_4 = 10$. Takové pole nazýváme jednorozměrné a můžeme si je představit takto:

index	1	2	3	4
složka pole A	5	3	7	10

V jazyku Pascal zapisujeme indexy pole dohranatých závorek, např. A[1] = 5.

Popis typu pole má základní tvar:

```
ARRAY [typ indexu] OF typ složky
```

Deklarace proměnné:

```
var A: array [1..4] of integer;
```

říká, že deklarujeme pole, které se jmenuje A a používá indexy 1 až 4. Složky tohoto pole jsou typu integer.

Každý typ pole může být, stejně jako ostatní typy, popsán v úseku deklarací typu

```
type vektor = array [1..10] of real;
```

a poté může být použit v deklaraci proměnné

```
var B: vektor;
```

Rozsah pole je možno vymezit i následujícím způsobem:

```
type DEN = (po, ut, st, ct, pa, so, ne);
   var TEPLOTA: array[DEN] of real;
```

pak přístup k jednotlivým složkám přes `TEPLOTA[po]` apod.

Příklad:

```
{Načtení pole o 10 složkách a tisk libovolné složky}
var   i: integer;
      a:array [1..10] of real;
begin
      writeln('Nacteni pole – 10 slozek:');
      for i:=1 to 10 do
begin       write ('Zadej ',i,'. slozku:');
      readln(a[i]);
end;
      write('Kterou slozku chces vytisknout? ');
      readln (i);
writeln('Hodnota ',i,'te slozky: ',a[i]:1:4);
end.
```

Vícerozměrné pole

Podíváme-li se na popis jednorozměrného pole, zjistíme, že na typ složek není kladeno žádné omezení. Typ složky může tedy být jak jednoduchý, tak i strukturovaný. Může to tedy být **opět typ pole**. Tímto se v Pascalu zavádějí typy **vícerozměrných polí**.

Příklad pole polí:

		sloupce			
A =	2	5	4	7	
	1	8	4	8	řádky
	3	8	12	1	

pole obsahuje 3 řádky a 4 sloupce, k jednotlivým složkám se tedy můžeme dostat přes indexy řádku a sloupce, tedy např. A[1,4] = 7 … ve složce na prvním řádku a ve čtvrtém sloupci je uložena 7.

Výše uvedné dvourozměrné pole si můžeme představit jako čtverec nebo obdelník, toto pole nazýváme **matice**.

deklarace typu

```
type pole1 = array [1..3] of array [1..4] of integer;
```

nebo zkrácený zápis

```
type pole1 = array [1..3,1..4] of integer;
```

a poté deklarace proměnné

```
var m: pole1;
```

nebo zkrácený zápis bez deklarace typu

```
var m: array [1..3,1..4] of integer;
```

Přístup ke složce pole přes indexy, např.: `m[1,4]` – složka na prvním řádku a čtvetém sloupci.

Úvahu o zavedení dvoudimenzionálního pole (matice) můžeme dále zobecnit a získat pole o **n** dimenzích, např. deklarací

```
var pole3: array[1..3,1..3,1..10] of real;
```

získáme pole o 3 dimenzích typu real (pro názornost si představce krychli nebo kvádr).

Příklad:

```
{Matice 20x20, vynulovani matice, jednotkova matice}
var   i,j:integer
      A,B:array [1..20,1..20] of real;
begin
      for j:=1 to 20 do A[1,j]:=0;  {Vynulovani prvniho radku matice}
      for i:=2 to 20 do A[i]:=A[1]; {Vynulovani dalsich radku – kopie 1.
radku}

      B:=A;                         {Kopie A do B – vynulovani matice B}
      for i:=1 to 20 do B[i,i]:=1;  {Nastaveni slozek na 1 na diagonale}
end.
```

Typ řetězec

Datový objekt typu řetězec může obsahovat řetězec znaků. Typ řetězec nemá pevně určenou délku, můžeme si ji definovat sami. Pokud definice délky chybí, je za implicitní délku vzata maximální možná délka 255 znaků.

Pokud je délka definována, může datový typ obsahovat právě tolik znaků. S datovým typem můžeme pracovat jako z celkem a odvolávat se na něj pomocí identifikátoru (pak je přístupný celý řetězec), nebo jej lze i indexovat (pak je přístupný pouze jeden znak z řetězce). Znaky jsou indexovány od 1 do N (N je délka řetězce). Přes index [0] je přístupná délka řetězce, lépe je však zjišťovat délku řetězce přes funci LENGTH.

deklarace

```
var   S:string
reterez: string[20]
```

Příklad:

```
{Zjisteni delky promenne typu retezec}
uses crt;
var   r: string[20];
      i: integer;
begin
      write(,Zadej text, maximalne 10 znaku: ‘);
      readln(r);
      for i:=1 to 20 do write(i,‘ = ‘,ch(i),‘ | ‘); {Tisk casti ASCII
tabulky}
writeln;
writeln(,Retezec obsahuje znaku: ‘ord(r[0]));
writeln(,Delce retezce odpovida znak: ‘,r[0]);
repeat until keypressed;
end.
```

Další použitelné funkce při práci s řetězci:

UpCase(c:char):char;

- převede jeden znak na velké písmeno

Concat(s_1,[s_2, …, s_n]):string;

- spojuje řetězce s_1, …, s_n v jeden, je-li výsledek delší než 255 znaků, je zkrácen na 255 znaků

Copy(s:string; index,pocet:integer):string;

- vrací podřetězec z řetězce **s**, **index** udává počáteční index a **pocet** udává počet znaků ve výsledku
- je-li **index** větší než pocet vrátí funkce prázdný řetězec
- je-li **pocet** větší než zbývající počet znaků řetězce, dojde ke zkrácení

Delete(var s:string; index,pocet:integer) ;

- vymaže z řetězce daný podřetězec udaný počátečním znakem **index** a délkou **pocet**

Length(s: string):integer;

- vrací délku řetězce

Insert(src:string, var s:string, idx:integer)

- vkládá řetězec **src** do řetězce **s**, **idx** udává počáteční hodnotu v řetězci **s**, kam bude vloženo **src**

Pos(substr:string, s: string):byte;

- vyhledává v řetězci **s** podřetězec **substr**
- vrací hodnotu indexu prvního znaku v řetězci **s**, kde byl **substr** nalezen

Str(x [:width[:decimal]], var s:string);

- konvertuje numerickou hodnotu X na řetězec znaků
- **width** udává celkový počet znaků
- **decimal** udává počet desetinných míst

Val(s:string; var v; code:integer);

- konvertuje hodnotu typu string na numerickou hodnotu
- synstaxe **s** musí odpovídat celému či reálnému číslu
- **code** je nastaven na nulu při úspěšném převodu, jinak je v něm index znaku, kde došlo k chybě při převodu.

Typ množina

Strukturovaný datový typ množina se vytváří z prvků báze (základu). Bázovým typem smí být pouze typ ordinální. Všechny možné kombinace prvků báze pak jsou hodnotami typu množina. Popis typu množina má tento obecný tvar:

```
SET OF bázový typ;
```

Počet takových množin je: 2^n, kde n je počet prvků báze.

V Borland Pascalu může mít bázový typ maximálně 256 hodnot.

Příklad množiny:

```
type  I = 1..3;          {interval, ktery tvori bazi – zaklad mnoziny}
      mnozina = set of I;     {typ mnozina s bazovym typem I}
var   m: mnozina;        {promenna typu mnozina}
      j: I;
```

pak proměnná **j** typu **I** může nabývat hodnot: **1**, **2** nebo **3**, hodnotami proměnné **m** typu **mnozina** mohou být:

[]	prázdná mnozina
[1], [2], [3]	jednoprvkové množiny
[1, 2], [2, 3], [1, 3]	dvouprvkové množiny
[1, 2, 3]	tříprvková množina

Pozor: [1, 2] je totéž co [2, 1]

Konstrukce hodnoty typu množina (konstruktor množiny) spočívá ve **výčtu** množinových elementů, které jsou **odděleny čárkami** a **uzavřeny** do hranatých závorek. Elementy mohou být dány výrazy bázového typu nebo ve tvaru n..m, který představuje množinu všech prvků i

bázového typu takových, že n<=i<=m, [] je prázdná množina.

Příklad:

```
[], [i+j,i-j], [0..9], [1,3,5..9,45,89..87]
```

zápis 89..87 je přípustný a obsahuje prázdnou množinu

Množinové operace

S hodnotami množinového typu jsou přípustné tyto množinové operace:

1) **sjednocení** množin
   a. předpisuje se operátorem **+** (plus)
   b. výsledkem sjednocení množin **A + B** je množina obsahující všechny prvky sjednocovaných množin
2) **průnik** množin
   a. předepisuje se operátorem ***** (krát)
   b. výsledkem průniku množin **A * B** je množina obsahující společné prvky obou množin
3) **rozdíl** množin
   a. předepisuje se operátorem **–** (mínus)
   b. rozdílem množin **A – B** je množina obsahující prvky množiny A, které nejsou obsaženy v množině B
4) **relace**
   a. **A <= B** – množina A je podmnožinou (částí) množiny B
   b. **A >= B** – množina B obsahuje množinu B (B je podmnožinou množiny A)
   c. **A = B** – množina A je totožná s množinou B
   d. **A <> B** – množina A není totožná s množinou B
   e. pro **hodnotu X** ordinálního typu a **množinu A** je definová relace **X in A**, tj. X je prvkem množiny A

Příklad:

```
{urceni seznamu prvocisel bez pouziti nasobeni a deleni}
program Eratostenovo_sito;
uses crt;
const n=255;
var sito, prvocisla: set of 1..n;
    i,dalsi, nasobek: word;
begin
     clrscr;
     prvocisla:=[];  {inicializace promennych}
     sito:=[2..n];
     dalsi:=2;
     repeat
           while not (dalsi in sito) do dalsi:=succ(dalsi);
                            {moznost pridat dalsi prvocislo}
           prvocisla:=prvocisla+[dalsi]; {pridani prvocisla}
           nasobek:=dalsi;
           while nasobek <= n do   {eliminace nasobku prvocisla}
               begin
```

```
                    sito:=sito - [nasobek];
                    nasobek:=nasobek + dalsi;
                end;
    until sito=[]; {opakovani dokud nebude sito prazdne}
    write('Vypis prvocisel: ');
    for i:=1 to n do
        if i in prvocisla then write(i,' ');
    repeat until keypressed;
end.
```

Záznamy

Soubory

Procedury a funkce

Podprogram je uzavřený programový celek, předem popsaný a pojmenovaný. Původní význam spočíval v tom, že programátor nemusel opisovat stejné nebo podobné posloupnosti příkazů, které se vyskytovali v různých částech programu, ale mohl posloupnost popsat pouze jednou, pojmenovat ji identifikátorem a pak v případě potřeby použít identifikátor jako příkaz ke spuštění. Tomuto způsobu říkáme volání podprogramu.

Využitím podprogramů také dosáhneme logického členění (strukturování) celého řešení. Velký problém rozčleníme na menší a pro ně napíše podprogram, celek tvoří program (řešení problému).

Nejdůležitější skupinu tvoří podprogramy, jejich algoritmus je popsán zcela obecně, a to pomocí identifikátorů (formálních parametrů), které jsou v případě volání nahrazeny jinými identifikátory – skutečnými parametry.

V jazyce TP použíme 2 typy podprogramů – **funkce** a **procedury**. Procedury jsou obecnější, funkce můžeme považovat za speciální případ procedur. Výsledkem funkce je jediná hodnota (podobně jako funkce v matematice). Výhodou je možnost využívat funkcí jako operand ve výrazech nebo jako parametr jiných funkcí.

Procedury

Syntaxe:

```
procedure Jméno_procedury (seznam formálních parametrů);
var k:integer; {seznam lokalnich parametru}
begin
      posloupnost prikazu;
end;
```

Procedury bez parametrů

Procedura může být i bez parametrů. Potom se ale v těle procedury mohou používat jen takové proměnné, které jsou deklarovány vně procedury – jsou deklarovány v nadřazeném bloku (jsou pro proceduru globální). Jen pomocí nic může procedura komunikovat s okolím.

Lokální proměnné jsou deklarovány v proceduře, jejich platnost je omezena pouze pro proceduru. Volání procedury bez parametrů se provádí příkazem, který obsahuje pouze jméno procedury.

Příklad:

```
program volani {volani procedury, ktera vymeni hodnoty dvou promennych}
var   x,y: real;  {promenne x a y jsou globalni v celem programu}
procedure VYMENA;
var   pom:real;   {promenna pom je lokalni – mozno vyuzit pouze v teto, }
{popr. podrizne casti procedury }
begin
      pom:=x;
      x:=y;
      y:=pom;
end;
begin
      x:=2;
      y:=3;
      VYMENA;
      writeln(x,y);     {Vytiskne 3, 2}
      VYMENA;
      writeln(x,y);     {Vytiskne 2, 3}
end.
```

Výše uvedené procedury mají pouze omezené použití, protože komunikují s okolím pouze pomocí globálních proměnných. Vyměnit obsah proměnných A a B pomocí této procedury nejsme schopni.