# PASCAL V KOCKE

*pre základnú úroveň maturitnej skúšky - B*

Pascal sa považuje za základný jazyk informatiky. Programom zapísaným v Pascale musí rozumieť každý.

Každý program sa skladá z údajovej a príkazovej časti. Údajovú časť tvoria deklarácie premenných. Príkazovú časť príkazy programu, ktoré procesor postupne vykonáva. Príkazová časť sa začína príkazom: **`begin`** a končí príkazom: **`end.`** (aj s bodkou)!

Deklarácia premenných pozostáva z vymenovania identifikátorov (mien) premenných a určenia ich typu. Deklaráciou sa nájde v operačnej pamäti miesto pre premennú a toto miesto sa po dobu vykonávania programu bude volať identifikátorom premennej.

Údajové typy:
- ⇨ jednoduché – jedna položka
- ⇨ zložené – viacero položiek

Príkazy:
- ⇨ jednoduché
- ⇨ riadiace

## 1. Jednoduché údajové typy

Pre znak:
- ⇨ **`char`** – rozsah: #0 .. #255 (binárne kódy znakov), veľkosť 1B

Pre celé čísla:
- ⇨ **`byte`** – rozsah: 0 .. 255, veľkosť 1B
- ⇨ **`word`** – rozsah: 0 .. 65 535, veľkosť 2B
- ⇨ **`shortint`** – rozsah: -128 .. 127, veľkosť 1B
- ⇨ **`integer`** – rozsah: -32 768 .. 32 767, veľkosť 2B
- ⇨ **`longint`** – rozsah: -2 147 483 647 .. 2 147 483 647, veľkosť 4B

Pre racionálne čísla:
- ⇨ **`real`** – rozsah: $2{,}9.10^{-9}$ .. $1{,}7.10^{38}$, počet číslic: 11-12, veľkosť 6B
- ⇨ **`single`** – rozsah: $1{,}5.10^{-45}$ .. $3{,}4.10^{38}$, počet číslic: 7-8, veľkosť 4B
- ⇨ **`double`** – rozsah: $5{,}0.10^{-324}$ .. $1{,}7.10^{308}$, počet číslic: 15-16, veľkosť 8B
- ⇨ **`extended`** – rozsah: $3{,}4.10^{-4932}$ .. $1{,}1.10^{4932}$, počet číslic: 19-20, veľkosť 10B
- ⇨ **`comp`** (zobrazenie čísel bez desatinnej časti) – rozsah: $-2^{63}+1$ .. $2^{63}-1$, počet číslic: 19-20, veľkosť 8B

Pre logické hodnoty:
- ⇨ **`boolean`** – hodnoty: false, true; veľkosť 1B

Údajové typy pre celé čísla, pre znak a logické hodnoty sa nazývajú ***ordinálne***.

### 1.1. Operácie nad ordinálnymi údajovými typmi

- ⇨ predchodca: ***`pred(p)`***
- ⇨ následník: ***`succ(p)`***
- ⇨ ordinálna hodnota: ***`ord(p)`***
- ⇨ zvýšenie ordinálnej hodnoty o 1: ***`inc(p)`***
- ⇨ zníženie ordinálnej hodnoty o 1: ***`dec(p)`***

### 1.2. Operácie nad logickým údajovým typom

- logická negácia: **`not`** – počet operandov: 1
- logický súčin: **`and`** – počet operandov: 2
- logický súčet: **`or`** – počet operandov: 2
- logický výhradný súčet (nonekvivalencia): **`xor`** – počet operandov: 2

| *Hodnoty operandov* | | *Výsledky operácií* | | | |
|---|---|---|---|---|---|
| *p* | *q* | *p* **and** *q* | *p* **or** *q* | *p* **xor** *q* | **not** *p* |
| false | false | false | false | false | true |
| false | true | false | true | true | true |
| true | false | false | true | true | false |
| true | true | true | true | false | false |

### 1.3. Relačné operácie a relačné operátory

- menší: **`<`**
- menší alebo rovný: **`<=`**
- rovný: **`=`**
- nerovný, je rôzny: **`<>`**
- väčší alebo rovný: **`>=`**
- väčší: **`>`**

Do každej relačnej operácie vstupujú dva operandy (konštanta, premenná, výraz, funkcia). Výsledkom každej relačnej operácie je logická hodnota (true, false).

### 1.4. Operácie nad celočíselnými údajovými typmi

- sčítanie: **`+`**
- odčítanie: **`-`**
- násobenie: **`*`**
- celočíselné delenie: **`div`**
- zvyšok po delení: **`mod`**
- absolútna hodnota: ***`abs(p)`***
- druhá mocnina: ***`sqr(p)`***
- určenie nepárnosti: ***`odd (p)`*** – výsledok je true, ak je číslo nepárne, alebo false, ak je párne

### 1.5. Operácie nad racionálnymi údajovými typmi

- sčítanie: **`+`**
- odčítanie: **`-`**
- násobenie: **`*`**
- delenie: **`/`**
- absolútna hodnota: ***`abs(p)`***
- druhá mocnina: ***`sqr(p)`***
- sínus: ***`sin(p)`***

- ⇨ kosínus: ***cos(p)***
- ⇨ arkus tangens: ***arctan(p)***
- ⇨ prirodzený logaritmus: ***ln(p)***
- ⇨ exponenciálna funkcia: ***exp(p)***
- ⇨ druhá odmocnina: ***sqrt(p)***
- ⇨ celá časť racionálneho čísla: ***trunc(p)*** – výsledkom je celé číslo!
- ⇨ zaokrúhlenie racionálneho čísla na celé číslo: ***round(p)*** – výsledkom je celé číslo!
- ⇨ desatinná časť (iba Turbo Pascal): ***frac(p)***
- ⇨ celá časť (iba Turbo Pascal): ***int(p)***
- ⇨ Ludolfovo číslo (iba Turbo Pascal): ***pi***

Ak jeden z operandov vstupujúcich do operácie sčítania alebo odčítania alebo násobenia je typu racionálne číslo a druhý je typu celé číslo, potom výsledok operácie je racionálne číslo. Ak do operácie delenia (/) vstupujú celé čísla, výsledok je racionálne číslo.

### 1.6. Priorita operátorov

1. not
2. *, /, div, mod, and
3. +, −, or, xor
4. <, <=, =, <>, >=, >

### 1.7. Deklarácia premenných jednoduchých údajových typov

Začína kľúčovým slovíčkom **var**. Potom nasleduje

*(zoznam premenných rovnakého typu, identifikátory premenných sú od seba oddelené čiarkami)* **: typ premennej** ***;***

```
napr.: var a,b : integer;
           c : char;
           d : real;
```

Ak deklarujeme premenné pred deklaráciou podprogramov, budú prístupné aj podprogramom (podprogram môže zmeniť ich hodnotu), hovoríme, že sme ich deklarovali globálne. Ak premenné zadeklarujeme za deklaráciami podprogramov, nebudú im prístupné (podprogram nemôže zmeniť ich hodnotu), hovoríme, že sme ich deklarovali lokálne. Všetky premenné zadeklarované za názvom podprogramu, sú prístupné iba tomuto podprogramu, sú teda lokálne.

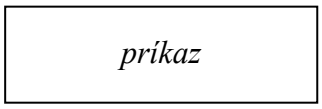
### 1.8. Deklarácia konštánt

**const** ***názov_konštanty = hodnota;*** napr.: const MaxCislo = 100;

V programe namiesto čísla 100 použijeme konštantu MaxCislo. Výhoda je taká, že pri zmene hodnoty konštanty, stačí urobiť zmenu iba v deklarácii a nie na všetkých miestach programu, kde je táto konštanta použitá.

## 2. Jednoduché príkazy

Značka v štrukturogramoch:




- priradenie: ***premenná := výraz;*** napr.: `a := b + 25;`
- volanie podprogramu: ***názov_podprogramu;***

Podprogramom môže byť funkcia (vracia do volajúceho prostredia jednu hodnotu, ktorá je vo funkcii priradená premennej s identifikátorom zhodným s názvom funkcie) alebo procedúra (nič nevracia).

Deklarácia funkcie bez parametrov:

```
function názov_funkcie;
deklarácia lokálnych premenných;
begin
   príkazy funkcie a medzi nimi aj nasledujúce priradenie:
   názov_funkcie :=výraz;
end;
```

Deklarácia procedúry bez parametrov:

```
procedure názov_procedúry;
deklarácia lokálnych premenných;
begin
   príkazy procedúry;
end;
```

Podprogramy deklarujeme vždy pred hlavným programom a tiež pred ich prvým použitím (volaním).

Pre riešenie všeobecných úloh podprogramami, môžeme do podprogramu poslať buď hodnoty lokálnych premenných alebo adresy na tieto premenné. Pri volaní podprogramu s parametrami za názvom podprogramu uvádzame v zátvorke zoznam premenných, ktoré podprogramu posielame.

Ak posielame do podprogramu hodnotu premennej, deklarujeme ho takto:
***typ_podprogramu názov_podprogramu(parameter:typ, ...);***
Vtedy sa hodnota posielanej premennej počas behu podprogramu nezmení, lebo podprogram si tuto hodnotu uloží do svojej lokálnej premennej s rovnakým identifikátorom.

Ak posielame do podprogramu adresu premennej, deklarujeme ho takto:
***typ_podprogramu názov_podprogramu(var parameter:typ, ...);***
Vtedy podprogram môže zmeniť hodnotu posielanej premennej počas svojho behu, lebo má k tejto premennej priamy prístup.

### 2.1. Komunikácia programu s užívateľom

- vstup údajov: **read(***premenné oddelené čiarkou***);** – prečíta každý údaj zo vstupu, prekonvertuje ho do typu príslušnej premennej a uloží ho do tejto premennej. Znak klávesy Enter, ktorou boli vstupné údaje programu

poslané, neprečíta. Tento znak zostáva nespracovaný, bude následne čítaný!

**readln(***premenné oddelené čiarkou***);** – prečíta každý údaj zo vstupu, prekonvertuje ho do typu príslušnej premennej a uloží ho do tejto premennej. Znak klávesy Enter, ktorou boli vstupné údaje programu poslané, prečíta tiež.

- ⇨ výstup údajov: **write(***premenné a reťazce znakov oddelené čiarkou***);** – vypíše obsahy premenných v textovom tvare a uvedené reťazce znakov. Po výpise zostáva kurzor na mieste za posledným vypísaným znakom.
  **writeln(***premenné a reťazce znakov oddelené čiarkou***);** – vypíše obsahy premenných v textovom tvare a uvedené reťazce znakov. Po výpise sa kurzor presunie na začiatok nového riadka.

Výpis premenných na pevný počet miest:

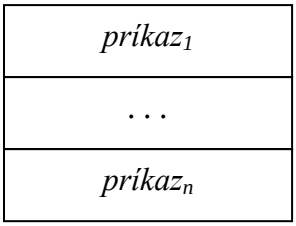
napr.: write(a:5, b:7:2); {premenná a je typu integer a vypíšeme ju na 5 miest zarovnanú doprava, premenná b je typu real a vypíšeme ju na 7 miest, pričom desatinná bodka bude na 5-tej pozícii a desatinná časť bude mať dĺžku 2 znaky}

## 3. Riadiace príkazy

- ⇨ sekvencia – blok príkazov
- ⇨ príkaz vetvenia
- ⇨ príkaz cyklu

### 3.1. Sekvencia príkazov

Značka v štrukturogramoch:




Zápis v Pascale:

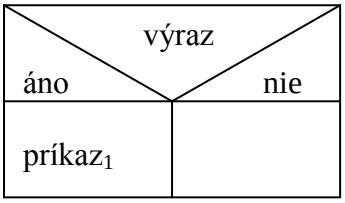
```
begin
      príkaz1;
      . . .
      príkazn;
end;
```

Fungovanie: V sekvencii príkazov sa vykonáva postupne jeden príkaz za druhým tak, ako nasledujú príkazy za sebou.

### 3.2. Príkaz vetvenia

#### 3.2.1. Príkaz if – neúlpný tvar

Značka v štrukturogramoch:



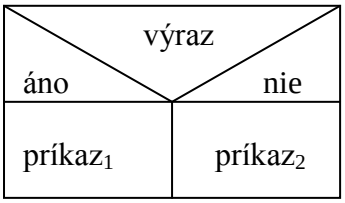

Zápis v Pascale:

```
if výraz then príkaz1;
```

Fungovanie: Najprv sa vyhodnotí logický výraz. Ak je pravdivý, vykoná sa príkaz$_1$ a potom sa pokračuje za príkazom vetvenia. Ak je nepravdivý, príkaz$_1$ sa preskočí a pokračuje sa za príkazom vetvenia.

### 3.2.2. Príkaz if – úlpný tvar

Značka v štrukturogramoch:




Zápis v Pascale:

```
if výraz then príkaz1
else príkaz2;
```

Fungovanie: Najprv sa vyhodnotí logický výraz. Ak je pravdivý, vykoná sa príkaz$_1$, príkaz$_2$ sa preskočí a pokračuje sa za príkazom vetvenia. Ak je nepravdivý, príkaz$_1$ sa preskočí, vykoná sa príkaz$_2$ a pokračuje sa za príkazom vetvenia.

### 3.2.3. Príkaz case

Zápis v Pascale:

```
case premenná of
     hodnota1 : príkaz1;
     hodnota2 : príkaz2;
     . . .
     hodnotan : príkazn;
     else príkazn+1;
end;
```

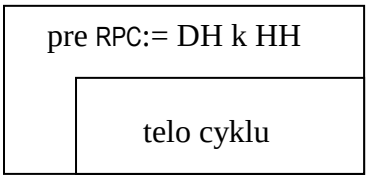
Hodnotou môže byť konštanta alebo interval hodnôt.

Fungovanie: Podľa toho, akú hodnotu má premenná, vykoná sa príslušný príkaz a pokračuje sa za príkazom case. V prípade, že hodnota premennej sa nerovná hodnote$_1$ až hodnote$_n$, vykoná sa príkaz za kľúčovým slovom else.

## 3.3. Príkaz cyklu

### 3.3.1. Cyklus s pevným počtom opakovaní

Značka v štrukturogramoch:




Zápis v Pascale:

```
for RPC:= DH to HH do
     telo_cyklu;
```

Fungovanie: RPC je riadiaca premenná cyklu, je vždy typu integer. DH je dolná hranica, celé číslo. HH je horná hranica, celé číslo. Telo cyklu je jeden príkaz alebo blok príkazov. Na začiatku sa do RPC priradí hodnota DH a testuje sa podmienka cyklu, teda či **RPC<=HH.** Ak je tento výraz pravdivý, vykoná sa telo cyklu, **zväčší** sa hodnota RPC **o 1** a opäť sa testuje podmienka cyklu. Ak je výsledok opäť pravdivý, pokračuje sa v cykle, ak nie je, pokračuje sa za telom cyklu. Minimálny počet opakovaní cyklu je 0.

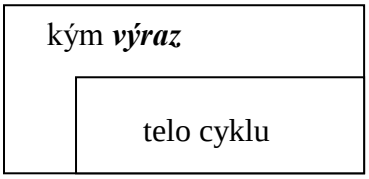
Tento cyklus môže mať aj takýto tvar:

```
for RPC:= HH downto DH do
     telo_cyklu;
```

Fungovanie: Na začiatku sa do RPC priradí hodnota HH a testuje sa podmienka cyklu, teda či **RPC>=DH**. Ak je tento výraz pravdivý, vykoná sa telo cyklu, hodnota RPC sa **zmenší o 1** a opäť sa testuje podmienka cyklu. Ak je výsledok opäť pravdivý, pokračuje sa v cykle, ak nie je, pokračuje sa za telom cyklu. Minimálny počet opakovaní cyklu je 0.

### 3.3.2. Podmienený cyklus s podmienkou ukončenia na začiatku

Značka v štrukturogramoch:



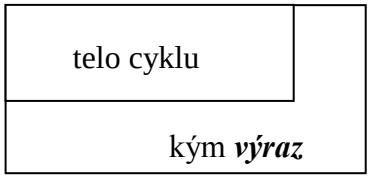

Zápis v Pascale:

```
while výraz do
      telo_cyklu;
```

Fungovanie: Najprv sa vyhodnotí pravdivosť logického výrazu. Ak je tento výraz pravdivý, vykoná sa telo cyklu (jeden príkaz alebo blok príkazov), a opäť sa testuje jeho pravdivosť. Ak je opäť výsledok pravdivý, pokračuje sa v cykle, ak nie je, pokračuje sa za telom cyklu. Minimálny počet opakovaní cyklu je 0.

### 3.3.3. Podmienený cyklus s podmienkou ukončenia na konci

Značka v štrukturogramoch:




Zápis v Pascale:

```
repeat
      telo_cyklu
until výraz;
```

Fungovanie: Najprv sa vykoná telo cyklu (jeden príkaz alebo blok príkazov) a potom sa vyhodnotí pravdivosť logického výrazu. Ak je tento výraz nepravdivý, pokračuje sa v cykle, ak je pravdivý, pokračuje sa za telom cyklu. Minimálny počet opakovaní cyklu je 1.

## 4. Zložené údajové typy

- pole
- reťazec
- textový súbor

### 4.1. Jednorozmerné pole

Vlastnosti:

- ⇨ pevne daný počet prvkov poľa
- ⇨ všetky prvky poľa sú rovnakého údajového typu
- ⇨ jednotlivé prvky poľa majú svoje jednoznačné označenie, tzv. **index**. Ordinálna hodnota indexu nasledujúceho prvku je vždy o 1 vyššia než ordinálna hodnota indexu prvku predchadzajúceho.

Deklarácia:
*(zoznam premenných, premenné sú oddelené čiarkou)* **: array[**z..k**] of** *údajový_typ* **;**

z je počiatočná hodnota indexu, k je posledná (koncová) hodnota indexu.

napr.: pole : array[1..7] of integer;

Práca s poľom: príklad:

```
var     i : integer;
        cisla : array[1..10] of real;
begin
        for i:=1 to 10 do readln(cisla[i]);
        writeln('Siedme nacitane cislo je ',cisla[7]);
end.
```

### 4.2. Reťazec znakov

Vlastnosti:

- ⇨ správa sa ako pole znakov
- ⇨ sústavne udržuje počet znakov vložených do reťazca, tzv. aktuálnu veľkosť reťazca
- ⇨ je deklarované množstvo funkcií a procedúr pre prácu s reťazcami
- ⇨ procedúry read, readln, write a writeln umožňujú priamo vstupné a výstupné operácie

Deklarácia:
*(zoznam premenných)* **: string;**

napr.: ret,text1,názov : string;

Práca s reťazcom:

```
ret:='Turbo';
text1:='';   {do premennej text1 priradzujeme prázdny reťazec}
nazov:=ret + text1;    {spojenie reťazcov do jedného reťazca}
```

Operácie pre prácu s reťazcami:

- ⇨ funkcia **Concat(*reťazec1, reťazec2, …, reťazecN*)** – spojí reťazce do jedného celku, výsledkom je hodnota typu string. Ak je súčet dĺžok všetkých reťazcov väčší ako deklarovaná dĺžka premennej, do ktorej výsledok ukladáme, reťazec sa automaticky skráti na túto dĺžku.

- ⇨ funkcia **Length(*reťazec*)** – zistí aktuálnu dĺžku reťazca, výsledkom je hodnota typu byte.
- ⇨ funkcia **Copy(*reťazec, od_pozicie, pocet_znakov*)** – vyberie z *reťazca* časť dĺžky *pocet_znakov* od stanovenej pozície. Výsledkom je hodnota typu string.
- ⇨ funkcia **Pos(*hľadaný_podreťazec, reťazec*)** – vráti pozíciu, od ktorej sa v *reťazci* nachádza *hľadaný_podreťazec*. Táto hodnota je typu byte. Ak sa hľadaný podreťazec v reťazci nenachádza, funkcia vráti 0.
- ⇨ procedúra **Insert(*vkladaný_reťazec, reťazec, kam*)** – vloží *vkladaný_reťazec* do druhého *reťazca* od pozície *kam*. Výsledná hodnota je uložená v premennej *reťazec*. Ak je v parametri *kam* hodnota väčšia ako je aktuálna dĺžka *reťazca* vloží sa *vkladaný_reťazec* za *reťazec*.
- ⇨ procedúra **Delete(*reťazec, odkiaľ, koľko_znakov*)** – odstráni časť reťazca od pozície *odkiaľ* o dĺžke *koľko_znakov*. Výsledná hodnota je uložená v premennej *reťazec*.
- ⇨ procedúra **Str(*číselný_výraz, reťazec*)** – prevedie výraz ľubovoľného číselného údajového typu na reťazec. Požadovanú dĺžku reťazca a počet desatinných miest určíme podobným spôsobom ako u procedúr write a writeln.
- ⇨ procedúra **Val(*reťazec, číslo, pozícia_chyby*)** – z reťazca vytvorí číselnú hodnotu. Rovnaký proces prevodu vykonávajú procedúry read a readln. Medzery pred začiatkom zápisu čísla nevadia, za zápisom čísla nesmú byť žiadne ďalšie znaky. Ak *reťazec I*neodpovedá zápisu čísla, resp. prevedená číselná hodnota nie je kompatibilná s údajovým typom skutočnej premnnej *číslo*, prevod sa neuskutoční a do parametra *pozícia_chyby* sa vloží číslo, ktoré odpovedá prvej pozícii reťazca, kde sa nachádza neprípustný znak v číselnom zápise. Ak je hodnota parametra *pozícia_chyby* 0, prevod prebehol bezchybne.

### 4.3. Textový súbor

Pojem súbor označuje časť diskového priestoru, ktorá obsahuje určité údaje. Každý súbor je označený názvom a prístupovou cestou. Textové súbory chápeme ako súbory znakov, ktoré sú vnútorne organizované do riadkov. Koniec riadku je vyznačený dohodnutými riadiacimi znakmi (v OS DOS je to #13#10).

Pred prácou so súborom musíme súbor najprv otvoriť – sprístupniť obsah súboru programu. Uzatvorením súboru oznamujeme OS, že súbor už nebude používaný.

Deklarácia textového súboru:

```
premenná : text;          {cez túto premennú sa sprístupní programu súbor}
```

napr.: subor : text;

Operácie pre prácu so súbormi:

- ⇨ procedure **assign(*premenná, názov_súboru*);** – prepojenie skutočného súboru s premennou pre sprístupnenie súboru. Táto procedúra musí byť použitá vždy pred otvorením súboru.
- ⇨ procedure **reset(*premenná*);** – otvorenie súboru pre čítanie.
- ⇨ procedure **rewrite(*premenná*);** – otvorenie súboru pre zápis. Ak takýto súbor existuje, pôvodný obsah sa vymaže.
- ⇨ procedure **append(*premenná*);** – otvorenie súboru pre zápis na koniec.
- ⇨ procedure **close(*premenná*);** – zatvorenie súboru.

- ⇨ **eof(*premenná*);** – vracia hodnotu true, ak sme sa dostali na koniec súboru, inak vráti hodnotu false.
- ⇨ **eoln(*premenná*);** – vracia hodnotu true, ak sme sa dostali na koniec riadka, inak vráti hodnotu false.
- ⇨ **seekeof(*premenná*);** – vracia hodnotu true, ak sme sa dostali na koniec súboru, inak vráti hodnotu false. Používa sa iba pre otvorené textové súbory.
- ⇨ **seekeoln(*premenná*);** – vracia hodnotu true, ak sme sa dostali na koniec riadka, inak vráti hodnotu false. Používa sa iba pre otvorené textové súbory.

Čítanie a zápis textových súborov:

Pre tieto operácie používame read, readln, write a writeln, pričom prvým parametrom v zátvorke je vždy premenná pre sprístupnenie súboru.