Moderné vzdelávanie pre vedomostnú spoločnosť / Projekt je spolufinancovaný zo zdrojov EÚ

# ZBIERKA

## RIEŠENÝCH A NERIEŠENÝCH

# ÚLOH V DELPHI

## II. DIEL

Materiál vznikol

na **Gymnáziu, Párovská 1, Nitra**

v rámci projektu **Implementácia kľúčových kompetencií v školskom vzdelávacom programe a modernizácia vyučovania prírodovedných predmetov**

aktivity **2.2 Inovácia vzdelávacieho programu voliteľných hodín predmetu informatika**

s cieľom aktivity **Zmodernizovať vyučovanie informatiky, pripraviť a zrealizovať školský vzdelávací program gymnázia pre vyučovanie voliteľných hodín predmetu informatika**

v období **od augusta 2009 do júla 2011**

autor **Jozef Piroško**

Na úvod si zhrnieme poznatky, ktoré by sme už mali vedieť, niektoré spresníme, niektoré zovšeobecníme a niektoré doplníme.

# Algoritmus, algoritmické konštrukcie, riadiace príkazy

## Algoritmus

**Algoritmus** – konečná postupnosť elementárnych príkazov, vykonanie ktorých, pre prípustné vstupné údaje, spĺňajúce vstupné podmienky, po konečnom počte krokov vedie mechanicky k výstupným údajom spĺňajúcim výstupné podmienky.
**Proces** – dej, ktorý prebieha počas vykonávania algoritmu
**Procesor** – vykonávateľ algoritmu (človek, počítač)

**Vstupná podmienka** – podmienka, ktorú musia spĺňať údaje na vstupe algoritmu, pre ktoré, pri správnom algoritme, dostaneme správny výsledok.
Napr.: A, B sú celé čísla;
**Výstupná podmienka** – podmienka, ktorú musia spĺňať všetky údaje na výstupe algoritmu, ak použité vstupné údaje spĺňajú vstupnú podmienku a algoritmus je správny.
Napr.: NSD je celé číslo, pre ktoré platí: NSD|A a NSD|B a ∄ NSD1>NSD: NSD1|A a NSD1|B
**Špecifikácia algoritmickej úlohy** – určenie vstupnej a výstupnej podmienky.
Údaje, ktoré spĺňajú vstupnú podmienku nazývame **vstupné údaje** (**vstupné hodnoty**).
Údaje, ktoré spĺňajú výstupnú podmienku nazývame **výstupné údaje** (**výstupné hodnoty**).

**Vlastnosti algoritmu:**

- hromadnosť – algoritmus slúži na riešenie celej skupiny úloh rovnakého typu. Algoritmus sa „dozvie“ až po zadaní konkrétnych vstupných údajov, ktorú „konkrétnu“ úlohu rieši.
- jednoznačnosť (deterministickosť) – po každom kroku (akcii) je jednoznačne určený ďalší krok. Táto vlastnosť zabezpečuje, že pre tie isté vstupné údaje dostaneme vždy rovnaké výsledky.

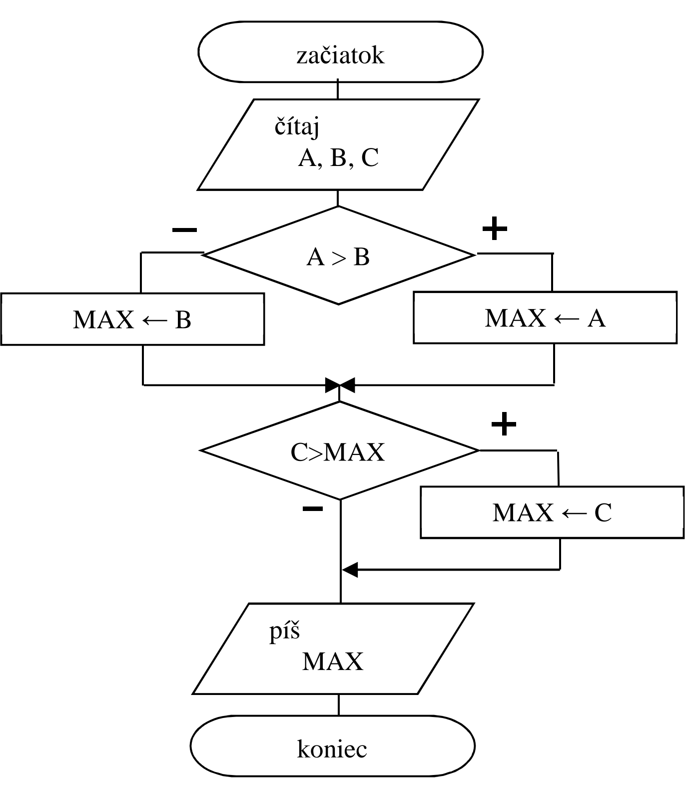
Ďalšie vlastnosti sú konečnosť, rezultatívnosť, elementárnosť atď.

**Formy zápisu algoritmu:**

- **zápis v jazyku vývojových diagramov** (JVD) používa štátnou normou stanovené značky, prehľadne (graficky) vyjadruje tok riadenia a dát.

  Príklad pravo: Algoritmus vo forme vývojového diagramu na nájdenie najväčšieho z troch čísel.

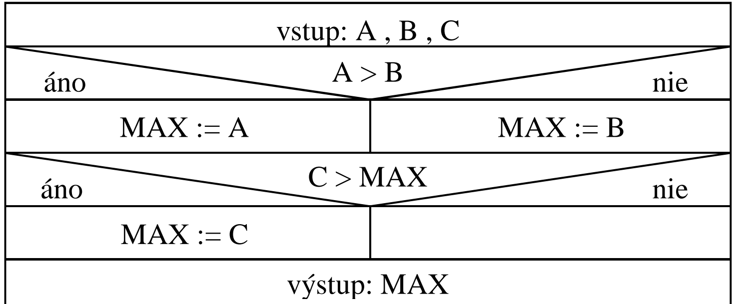
  Algoritmus sa vykonáva zhora nadol. Vetvou „+“ sa pôjde, keď bude podmienka splnená a vetvou „ -“, keď podmienka nebude splnená. Symbol „←“ čítame „priraď“.



- **slovný zápis** používa tzv. vyhradené slová, napr. ak – tak – inak, čítaj, začiatok a pod.
  Príklad: Slovný zápis algoritmu na nájdenie najväčšieho z troch čísel.

```
začiatok
čítaj ( A , B , C );
ak A > B
    tak   MAX ← A
    inak MAX ← B;
ak C > MAX
    tak   MAX ←C;
píš ( MAX );
koniec
```

- **zápis štruktúrogramom** – novšia grafická forma zápisu algoritmu.
  Príklad: Algoritmus na nájdenie najväčšieho z troch čísel zapísaný N-S diagramom.




**Algoritmické konštrukcie**
Ak je úloha algoritmicky riešiteľná, jej algoritmus možno vytvoriť kombináciou len troch algoritmických konštrukcií. Algoritmické konštrukcie sú: **sekvencia, vetvenie a cyklus**.

**Program** – algoritmus úlohy zapísaný v jazyku, ktorému „rozumie“ počítač, t.j. v programovacom jazyku; postupnosť príkazov programovacieho jazyka umožňujúca riešenie problému počítačom.
**Programovanie** – tvorba programov (pozri nižšie životný cyklus programu - etapy tvorby).
**Syntax** programovacieho jazyka – presný opis, ktoré symboly vytvárajú abecedu programovacieho jazyka (písmená anglickej abecedy, číslice 0 až 9, vyhradené slová begin, end, if, while, const, var, array, procedure,...) a ktoré reťazce reprezentujú správne zapísané konštanty, premenné, výrazy, príkazy, deklarácie či programy (napr. identifikátor, definovanie konštanty, tvar príkazu, unitu, procedúry,...). Rôzne programovacie jazyky majú rôznu syntax.
**Sémantika** – presné určenie významu vyššie uvedených prvkov.

## Testovanie a ladenie programu

Po napísaní programu musíme vhodnými prostriedkami overiť jeho funkčnosť a správnosť. Táto etapa sa nazýva testovanie a ladenie. **Testovaním** zisťujeme, či nie sú v danom programe chyby. **Ladením** ich odstraňujeme, t.j. predovšetkým lokalizujeme (určíme miesto chyby) a špecifikujeme (určíme chybu).

**Chyby** môžu byť:

- **syntaktické** (syntax errors), *zistené pri kompilácii* (preklade)
  napr.: preklep v príkaze, nedeklarovaná premenná, chybný počet parametrov a pod.
- **logické** (chyby v algoritme), *zistené počas alebo po skončení behu programu*
  - vedúce k predčasnému zastaveniu programu s chybovou správou (run time errors)
    napr.: delenie nulou, súbor na otvorenie nenájdený a pod.
  - vedúce k chybným výsledkom (prekročením rozsahu údajového typu – pozri str.37), k zacykleniu programu (zastavenie vykonávania programu: Run – Program Reset) a pod.

**Zásady testovania**:

- poznať správny výsledok
- pri konečnej množine vstupných údajov úplné otestovanie programu
- testovanie všetkých ciest (vetiev)
- testovanie hraničných a „problémových“ hodnôt

## Testovanie a ladenie programu v Delphi:

**Spustenie programu**
Run, F9

**Ukončenie behu programu**
Run – Program Reset, Ctrl+F2

**Zastavenie behu programu**
Run – Program Pause

**Krokovanie (trasovanie, tracing) programu**
Run – Trace Info, F7

**Krokovanie programu bez krokovania v podprogramoch**
Run – Step Over, F8

**Zastavenie na riadku s kurzorom**
Run – Run to Cursor, F4

**Zastavenie na určenom mieste (Breakpoints)**
Run – Add Breakpoint – Source Breakpoint... (Condition – zastavovacia podmienka, hodnota premennej rovná... a pod.; Pass Count – počet priechodov bez zastavenia)

**Sledovanie obsahu premenných (watching)**
Run – Add Watch..., Ctrl+F5 (pozri kontextovú ponuku vo Watch List)

**Zmena obsahu premenných**
Run – Evaluate/Modify..., Ctrl+F7 (pozri nástroje Evaluate, Modify, Watch)

## Životný cyklus programu (etapy tvorby programu):

(rovnaké kroky treba vykonať pri každej výraznejšej zmene programu)

1. **rozbor problému**
   sformulujeme zadanie problému a požiadavky na vznikajúci program; zodpovieme na otázku **čo** treba robiť
2. **návrh riešenia**
   hľadáme riešenie zvážením poznatkov z danej oblasti (napr. výberom najvhodnejšieho algoritmu), zvážením prostriedkov na riešenie a navrhnutím vhodného spôsobu organizácie údajov - výsledkom je algoritmus; zodpovieme na otázku **ako** sa dá daný problém riešiť
3. **realizácia**
   prepíšeme navrhnutý algoritmus do vhodného programovacieho jazyka; sem patrí aj príprava obrázkov, zvukových efektov, hudby do pozadia a pod.
4. **údržba**
   používanie softvéru a s ním súvisiace odhaľovanie a oprava skrytých chýb, prispôsobovanie softvéru meniacim sa požiadavkám používateľov, vývoj novších verzií atď.

## Programová dokumentácia

Dokumentácia k programu vysvetľuje, aký problém program rieši; aké vstupné údaje a akým spôsobom treba zadať, aké výstupné údaje možno očakávať, ako program funguje a ako ho používať. Slúži ako pomôcka pri odstraňovaní chýb, pri zmene či rozširovaní programu,...

Dokumentácia sa obyčajne robí dvoma spôsobmi:

- **komentáre** v programe
- **písomná dokumentácia**

Komentáre v programe sú vysvetľujúce poznámky priamo v programe a prekladač ich ignoruje (uvádzajú sa v množinových zátvorkách { } alebo jednoriadkové za „dvojlomítkami“ // ).

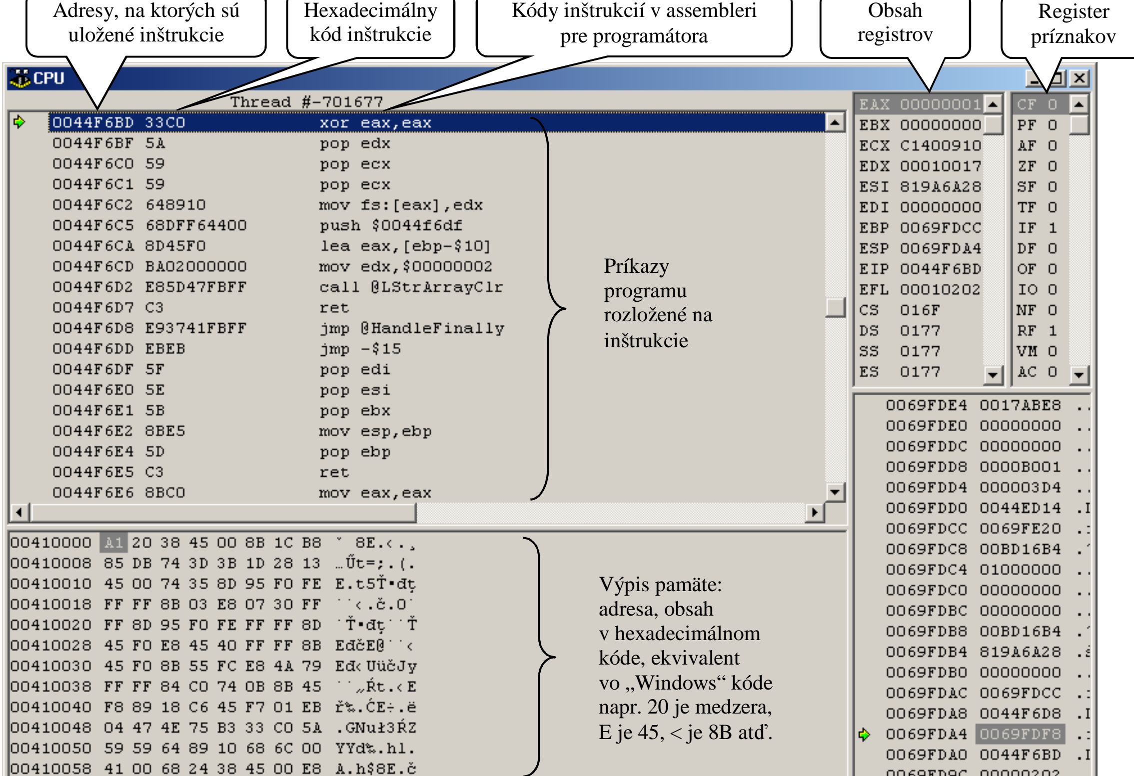
Písomná dokumentácia sa nachádza mimo programu a môže byť v papierovej alebo elektronickej podobe. Môže to byť **manuál** alebo **používateľská príručka**. Manuál obsahuje informácie pre programátorov, príručka pre používateľov softvéru.

## Spôsoby vykonávania programu

Po vytvorení programu (zdrojového kódu), ktorý je „obyčajným“ textom,

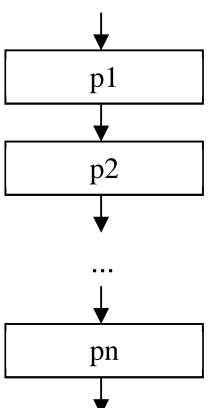
- **prekladač** (**kompilátor**) preloží tento text do objektového súboru (objektového kódu) – tento proces sa nazýva preklad alebo kompilácia. Objektový kód je v podstate už strojovým kódom procesora, obsahuje však ešte „relatívne“ informácie, napríklad relatívne adresy, odkazy na externé premenné a externý kód v knižniciach a pod. Program tiež často tvorí niekoľko objektových súborov, preto v druhom kroku **zostavovací program** (**linker**) spojí objektové súbory do výsledného strojového (binárneho) kódu a spraví ho „absolútnym“, napríklad sa nahradia relatívne adresy absolútnymi. Program sa stal postupnosťou čísel – strojových inštrukcií procesora, ktoré vie procesor veľmi rýchlo vykonať (pozri v Delphi po spustení programu a vyvolaní Run - Program Pause). Vznikla plnohodnotná aplikácia. Takto pracujú prekladače Pascalu, Delphi, C++ Builder,...
- **interpretér** „číta“ program a v texte rozpoznáva jednotlivé príkazy, ktoré hneď vykonáva (interpretuje). Vykonávanie programu interpretáciou je pomalšie (musia sa neustále rozpoznávať príkazy v programe). Interpretáciu využívajú napríklad programy zapísané v Basicu, Comenius Logo a Imagine. V počítači musí byť prítomný interpetér jazyka.
- **v Jave** kompilátor vytvorí namiesto objektového kódu **bajtkód**. Bajtkód je interpretovaný prostredníctvom **JVM** (Java Virtual Machine) a pomocou Java **API** (Application Programming Interface) v rôznych OS. Tri rôzne verzie Javy sú určené pre servre, stolné PC a PDA/mobily.

Príloha: Po zastavení behu programu Run – Program Pause sa zobrazí okno zobraz. obsah procesora:



## Sekvencia

Sekvencia je najjednoduchšou algoritmickou konštrukciou. Použijeme ju, ak sa majú príkazy vykonať za sebou, v poradí, ako sú zapísané.
Má tvar:




v slovnom zápise a v pascale:

**p1;**
**p2;**
**...**
**pn;**

kde p1, p2, ... , pn sú príkazy

Príkazy od seba oddeľujeme bodkočiarkou!

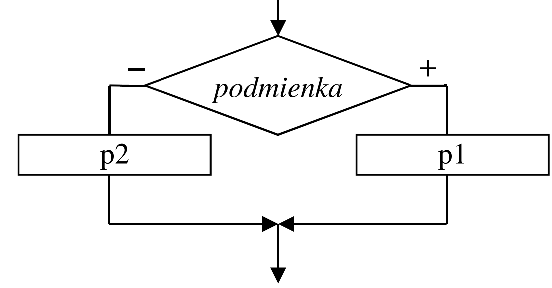
Vykonanie sekvencie: Príkazy p1, p2 až pn sa vykonajú za sebou v poradí, ako sú zapísané (ak neobsahujú riadiaci príkaz).
**Riadiace príkazy** môžu zmeniť poradie vykonávania príkazov, môžu zabezpečiť vykonanie skupiny príkazov, len ak je splnené určitá podmienka (vetvenie) alebo môžu zabezpečiť opakované vykonávanie skupiny príkazov (cyklus).

## Vetvenie

Vetvenie použijeme, ak vykonanie príkazu (alebo skupiny príkazov) je podmienené splnením určitej podmienky. Nesplnenie danej podmienky môže viesť k vykonaniu inej skupiny príkazov.
Vetvenie poznáme **binárne** (dve možnosti) a **n-árne** (n možností, n≥2).

### Úplné binárne vetvenie, podmienený príkaz if

Má tvar:




**ak *podmienka***
  **tak p1**
  **inak p2**
kde p1 a p2 sú príkazy

Vykonanie: Ak je podmienka splnená, vykoná sa príkaz p1, ak nie je splnená, vykoná sa príkaz p2.

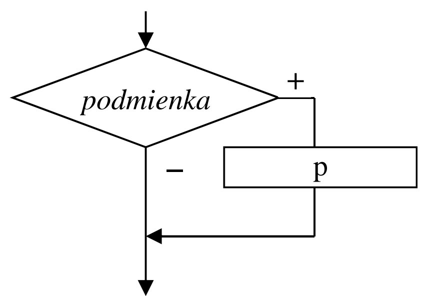
V pascale úplnému binárnemu vetveniu zodpovedá úplný príkaz if, ktorý má tvar:

**if** b **then** p1
**else** p2 kde b je výraz typu boolean, p1 a p2 sú príkazy

Vykonanie úplného príkazu if: Ak výraz b nadobudne hodnotu true, vykoná sa príkaz p1, ak nadobudne hodnotu false, vykoná sa príkaz p2.

## Neúplné binárne vetvenie, neúplný príkaz if

Má tvar:




**ak** ***podmienka***
  **tak p**
kde p je príkaz

Vykonanie: Ak je podmienka splnená, vykoná sa príkaz p, inak je vetvenie bez účinku.

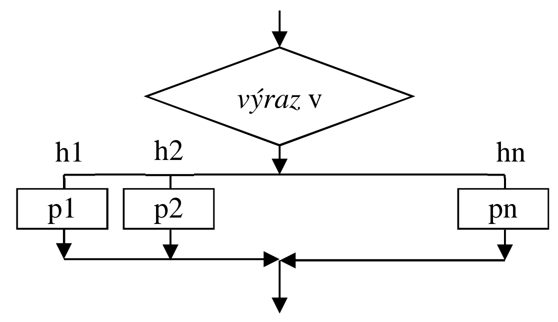
V pascale neúplnému binárnemu vetveniu zodpovedá neúplný podmienený príkaz if, ktorý má tvar:

**if** b **then** p    kde b je výraz typu boolean a p je príkaz

Vykonanie neúplného príkazu if: Ak výraz b nadobudne hodnotu true, vykoná sa príkaz p, ak nadobudne hodnotu false, príkaz if je bez účinku.

## N-árne vetvenie, podmienený príkaz case

Má tvar:




kde p1 až pn sú príkazy
h1, h2 až hn sú hodnoty, ktoré môže nadobudnúť výraz v

v slovnom zápise nemá ekvivalent, realizuje sa zápisom pomocou binárneho vetvenia:

ak *podm1* tak p1
inak ak *podm2* tak p2
inak ak *podm3* tak p3
inak ...
inak ak *podmn* tak pn

V pascale sa n-árne vetvenie realizuje podmieneným príkazom case.

Má tvar:

```
case v of
  h1 : p1;
  h2 : p2;
   ...
  hn : pn
[ else p ]
end
```

kde v je tzv. výberový výraz ordinálneho typu,
h1, h2 až hn sú hodnoty rovnakého typu ako výberový výraz
a p, p1, p2 až pn sú príkazy

do hranatých zátvoriek sa umiestňuje nepovinná časť
príkaz case končí vyhradeným slovom end

Vykonanie: Vyhodnotí sa výberový výraz a vykoná príkaz, predznačený hodnotou, ktorú nadobudol výberový výraz. Ak výraz v nenadobudne ani jednu z hodnôt h1 až hn a príkaz case obsahuje časť else, vykoná sa príkaz p; ak príkaz case neobsahuje časť else, príkaz case je bez účinku.

Napríklad

```
case Cislo of
     1 : mena:= ‘euro‘;
   2..4: mena:= ‘eurá‘;
   else mena:= ‘eur‘;
end;
```

```
case Znak of
‘A‘..‘Z‘,‘a‘..‘z‘: Label1.Caption:= ‘písmeno angl. abecedy‘;
        ‘0‘..‘9‘: Label1.Caption:= ‘číslica‘;
           else Label1.Caption:= ‘iný znak‘
end;
```

```
Priestupny:= (Rok mod 100 <> 0) and (Rok mod 4 = 0) or (Rok mod 400 = 0);
case Mesiac of
        1,3,5,7,8,10,12: Dni:= 31;
                        2: if Priestupny then Dni:= 29
                           else Dni:= 28;
              4,6,9,11: Dni:= 30
        else ShowMessage ('Zle zadaný mesiac!')
end;
```

### Príklad 5.1

Vytvorte program, ktorý po spustení vypíše aktuálny dátum (napr.1.7.2010) v tvare Dnes je štvrtok 1. júl 2010.

*Analýza*

Po algoritmických konštrukciách si zopakujeme a doplníme údajové typy. Tam by ste našli aj funkcie na prácu s dátumom a časom.

„Dnes je “ je obyčajný reťazec, s tým by sme mohli začať a postupne k nemu pripájať ďalšie reťazce, napríklad „štvrtok“ - deň v týždni. Funkcia **DayOfWeek** vráti poradové číslo dňa v týždni (1 je nedeľa!), potrebuje však aktuálny dátum. Funkcia **Date** vráti aktuálny dátum (typ TDateTime). S použitím príkazu case už ľahko vyriešime vypísanie názvu dňa. Keby sme ďalej vystačili len s číselným dátumom, t.j. 1. 7. 2010, mohli by sme použiť konverznú funkciu DateToStr (Date). My však chceme názov mesiaca slovne. Príkaz **DeCodeDate** prekonvertuje rok, mesiac a deň z typu TDateTime na celé čísla typu word (0 až 65 535). Teraz by už nižšie uvedené riešenie malo byť zrejmé až na jednu „maličkosť“. Doteraz sme na vykonanie príkazov používali udalosť kliknutie na tlačidlo. Úloha znie: Vytvorte program, ktorý po spustení... t.j. hneď po vytvorení formulára, vypíše... Formulár (Form1) má v zozname udalostí (Events) aj udalosť OnCreate. Po dvojkliku vpravo do bieleho poľa od OnCreate sa vytvorí procedúra TForm1.FormCreate, do ktorej stačí napísať príkazy, ktoré sa majú vykonať hneď pri vzniku formulára.

Riešenie

```
procedure TForm1.FormCreate(Sender: TObject);
var Oznam: string;
    Den, Mesiac, Rok: word;
begin
Oznam := 'Dnes je ';
case DayOfWeek (Date) of
        1: Oznam := Oznam + 'nedeľa ';
        2: Oznam := Oznam + 'pondelok ';
        3: Oznam := Oznam + 'utorok ';
        4: Oznam := Oznam + 'streda ';
        5: Oznam := Oznam + 'štvrtok ';
        6: Oznam := Oznam + 'piatok ';
        7: Oznam := Oznam + 'sobota ';
end;
DeCodeDate (Date , Rok , Mesiac , Den);
Oznam := Oznam + IntToStr (Den) + '. ';
case Mesiac of
        1: Oznam := Oznam + 'januára ';
        2: Oznam := Oznam + 'februára ';
        3: Oznam := Oznam + 'marca ';
        4: Oznam := Oznam + 'apríla ';
        5: Oznam := Oznam + 'mája ';
```

```
        6: Oznam := Oznam + 'júna ';
        7: Oznam := Oznam + 'júla ';
        8: Oznam := Oznam + 'augusta ';
        9: Oznam := Oznam + 'septembra ';
      10: Oznam := Oznam + 'októbra ';
      11: Oznam := Oznam + 'novembra ';
      12: Oznam := Oznam + 'decembra ';
end;
Label1.Caption := Oznam + IntToStr (Rok) + '.';
end;
```

Riešenie bez premenných a príkazu case (mesiac neskloňuje)

```
begin
ShortDateFormat := 'dddd d. mmmmm. yyyy.';
Label1.Caption := 'Dnes je ' + DateToStr (Date);
end;
```

✍💻 Vytvorte program, ktorý pre aktuálny dátum vypíše: Do konca mesiaca zostáva ...dní, do konca roka zostáva ...dní (problém riešime aj v časti konštantné pole).

## Príklad 5.2

Vytvorte program, ktorý zašifruje zadaný text (reťazec) posunutím písmen anglickej abecedy a číslice o jeden znak doprava, Z na A, z na a, 9 na 0, ostatné znaky nezmení.

*Analýza*

S podobným problémom sme sa už zaoberali v príklade 3.4.12 (I. diel). Keďže pri šifrovaní môže nastať viacej ako dve alternatívy (písmená A až Y sa šifrujú ináč ako Z a ináč ako číslice 0 až 8, ináč číslica 9,...), ide o viacnásobné vetvenie a je vhodné použiť príkaz case.

Keďže pravdepodobne hneď budeme chcieť aj dešifrovať, voľbu Šifrovať / Dešifrovať sme umiestnili do komponentu RadioGroup (popísaný v I. dieli príklad 2.1).

*Riešenie*

```
procedure TForm1.PracujClick(Sender: TObject);
var Ret, novyRet: string;
    i: integer;
begin
Ret:=Edit1.Text;
// do premennej Ret uloží reťazec z Edit1
setlength (novyRet , length(Ret));            // pre novyRet vyhradí pamäť dĺžky Ret
if RadioGroup1.ItemIndex = 0                   // index prvej položky (šifrovať) je nula, ďalšej jedna
then begin
        for i:= 1 to length(Ret) do            // zober prvý, potom druhý,... až posledný znak z Ret
          case Ret[i] of
             'A'..'Y','a'..'y', '0'..'8': novyRet[i]:= succ (Ret[i]);      // do novyRet ulož nasledovníka
                                    'Z': novyRet[i]:= 'A';                 // do novyRet ulož znak A
                                    'z': novyRet[i]:= 'a';
                                    '9': novyRet[i]:= '0'
                            else novyRet[i]:= Ret[i]                       // iný znak nezmeň
          end;
        Edit1.Text:= novyRet                                               // zobraz v Edit1 zašifrované
     end
else begin                                                                 // dešifruj
        for i:=1 to length (Ret) do
```

```
            case Ret[i] of
              'B'..'Z','b'..'z','1'..'9': novyRet[i]:= pred (Ret[i]);
                                'A': novyRet[i]:= 'Z';
                                'a': novyRet[i]:= 'z';
                                '0': novyRet[i]:= '9'
                           else novyRet[i]:= Ret[i]
            end;
      Edit1.Text:=novyRet
      end;
end;
```

### Príklad 5.3

Vytvorte program, ktorý prevedie nezáporné celé číslo zadané v šestnástkovej pozičnej sústave (ako reťazec) do desiatkovej.

*Analýza*

🕮 Potrebujeme si zopakovať alebo poznať niekoľko poznatkov z matematiky.

Použitie pozičnej sústavy znamená, že napríklad v čísle **123** je 1 na mieste stoviek, 2 na mieste desiatok, 3 na mieste jednotiek, čo možno zapísať $\mathbf{1}.10^2 + \mathbf{2}.10^1 + \mathbf{3}.10^0$ pri desiatkovej číselnej sústave.

Z matematiky vieme, že výraz $\mathbf{1}.10^2 + \mathbf{2}.10^1 + \mathbf{3}.10^0$ možno zapísať aj ako $((\mathbf{1}).10+\mathbf{2}).10+\mathbf{3} = (12).10+3 = 120+3 = 123$ a teda univerzálnejšie $a_2.10^2 + a_1.10^1 + a_0 = ((a_2).10 + a_1).10 + a_0$.

Všeobecne hodnotu výrazu $a_n.10^n + a_{n-1}.10^{n-1} + a_{n-2}.10^{n-2} +...+a_1.10^1 + a_0$ pre konkrétne číslice $a_i$ (koeficienty) a prirodzené n (i = 0,1,...,n) môžeme vypočítať aj pomocou výrazu $(...((a_n).10 + a_{n-1}).10 + a_{n-2}).10 +...+ a_1).10 + a_0$ resp. pomocou výrazu $(...(((\mathbf{0}).\mathbf{10}+a_n).10 + a_{n-1}).10 + a_{n-2}).10 +...+a_1).10 + a_0$, pričom pridanú nulu použijeme ako počiatočnú hodnotu výpočtu.

Výhodou takéhoto výpočtu je ušetrenie násobenia – nemusíme počítať $10^n$, $10^{n-1}$,..., $10^2$, lebo je „skryté“ vo vnorených zátvorkách. Kto chce vedieť viac, nech si vyhľadá na internete Hornerovu schému na výpočet hodnoty polynómu.

Schéma nás vedie k opakovaniu násobenia „čohosi“ desiatimi a k pripočítavaniu číslic. To „čosi“ je najprv 0, ktorú vynásobíme 10 a pridáme prvú číslicu čísla, opäť vynásobíme desiatimi (posunieme o rád doľava) a pripočítame druhú číslicu čísla atď. až kým nepripočítame poslednú číslicu. Čiže opakujeme: nová hodnota ← predchádzajúca hodnota.10 + ďalšia číslica.

Šestnástková číselná sústava používa šestnásť znakov 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A (10), B (11), C (12), D (13), E (14) a F (15). Napríklad číslo $1C_{16}$ je $28_{10}$ pretože $1C_{16} = 1.16^1 + 13.16^0 = 28$. Číslo $123_{16}$ by bolo $1.16^2 + 2.16^1 + 3.16^0 = 1.256 + 2.16 + 3.1 = 256 + 32 + 3 = 291$.

Základ je teda 16 a nie 10 a „cifry“ A, B, C, D, E a F musíme pri výpočte nahradiť číslami 10, 11, 12, 13, 14 alebo 15. Preto treba opakovať výpočet:

nová hodnota ← predchádzajúca hodnota.16 + ďalší šestnástk. znak zmenený na desiatkové číslo!

*Riešenie*

```
procedure TForm1.Prevod16Click(Sender: TObject);
var Cislo16Str: string;
    Cislo10: longint;                    // longint má väčší rozsah ako typ integer
    i: integer;
begin
Cislo16Str:= Edit1.Text;                 // do Cislo16Str uloží „šestnástkový“ reťazec
Cislo10:= 0;                             // počiatočná hodnota výsledku
for i:= 1 to length (Cislo16Str) do      // zober 1., 2.,..., posledný znak reťazca Cislo16Str
        case UpCase (Cislo16Str[i]) of   // UpCase zmení malé písmeno na veľké
```

```
        '0'..'9': Cislo10:= Cislo10*16 + ord(Cislo16Str[i]) - ord('0');   //zmeň na desiatkovú číslicu
        'A': Cislo10:= Cislo10*16 + 10;                                    // znak A nahraď číslom 10
        'B': Cislo10:= Cislo10*16 + 11;                                    // znak B nahraď číslom 11
        'C': Cislo10:= Cislo10*16 + 12;
        'D': Cislo10:= Cislo10*16 + 13;
        'E': Cislo10:= Cislo10*16 + 14;
        'F': Cislo10:= Cislo10*16 + 15;
      end;
Label2.Caption:= IntToStr(Cislo10);              // zobrazenie výsledku
end;
```

🕮 Vyššie rozobratý problém prevodu šestnástkového čísla na desiatkové sa dá zrejme zovšeobecniť. Ak z je základ číselnej sústavy ($z \geq 2$), podstatou prevodu v cykle čísla so základom z na desiatkové číslo je príkaz Cislo10 := Cislo10 * z + desiatkový ekvivalent spracovávaného znaku; pričom postupne berieme 1., 2.,..., length(Cisloz). znak so zadaného reťazca (čísla so základom z). Pokiaľ je základ väčší ako 10, je výhodné desiatkový ekvivalent spracovávaného znaku získať cez príkaz case.

💻 Ukážka podprogramu, ktorý prevedie číselný reťazec čísla so základom Z ($2 \leq Z \leq 9$, v konštante nastavené Z=8) na desiatkové číslo.

```
procedure TForm1.PrevodZClick(Sender: TObject);
const Z = 8;                              // takto sa definuje konštanta
      ZChar = chr (ord ('0') + Z-1);      // aj toto je definícia konštanty, zrejme ZChar = '7' pre Z = 8
var   CisloZStr: string;
      Cislo10, i: longint;
      Chyba: boolean;
begin
CisloZStr:= Edit1.Text;
Label2.Caption:= '';                      // zmaže predchádzajúci výstup
Chyba:= False;                            // predpokladáme, že všetky znaky v CisloZStr sú dovolené
Cislo10:= 0;
for i:= 1 to length(CisloZStr) do
      if CisloZStr[i] in ['0'..ZChar]
      then Cislo10:= Cislo10*Z + ord(CisloZStr[i]) - ord('0')
      else begin Chyba:= True; ShowMessage ('Nedovolený znak ' + CisloZStr[i]); end;
if not Chyba then Label2.Caption:= IntToStr (Cislo10);
end;
```

💻 Odlaďte aj pre Z = 2, napríklad $10000000000_2 = 1024 = 2^{10}$. Skúste aj Z = 10 ☺.

💻 Program upravte tak, aby sa opýtal na základ Z (celé číslo) a skontroloval aj jeho správnosť z intervalu <2,9>.

## Cyklus

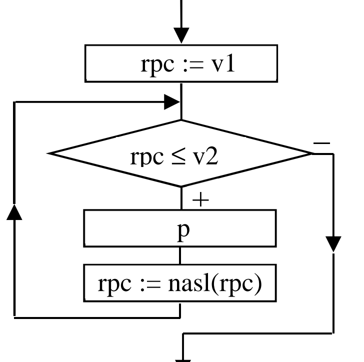
Použijeme, ak nejaký príkaz alebo skupina príkazov sa má opakovane vykonávať, pokiaľ je splnená (prípadne nesplnená) určitá podmienka.

Cykly rozdeľujeme:

1. podľa umiestnenia podmienky ukončenia na:
   a) cyklus s podmienkou ukončenia na začiatku
   b) cyklus s podmienkou ukončenia na konci
   c) úplný cyklus (s podmienkou ukončenia v strede – v pascale nemá samostatný príkaz)
2. podľa toho, či je známy alebo neznámy počet opakovaní príkazov v cykle na:
   a) cyklus s explicitne (zvonka) daným počtom opakovaní – s pevným počtom opakovaní
   b) cyklus s implicitne (zvnútra) daným počtom opakovaní – cyklus s podmienkou ukončenia

## Cyklus s pevným počtom opakovaní, príkaz for

Cyklus s pevným počtom opakovaní použijeme pri známom počte opakovaní príkazov v cykle.
V algoritmizácii nemá tento cyklus jednoznačnú formu zobrazenia, my sme sa rozhodli pre tvar:




v slovnom zápise:

**pre** *rpc* **:=** *v1* **až po (naspäť po)** *v2* **opakuj** *p*

kde rpc je tzv. riadiaca premenná cyklu (riadi počet prechodov cyklom), v1 a v2 sú výrazy a p je príkaz
nasl(rpc) znamená nasledovník rpc

V pascale cyklu s pevným počtom opakovaní zodpovedá príkaz for.

Má tvar: **for** rpc **:=** v1 **to** (**downto**) v2 **do** p

kde rpc je tzv. riadiaca premenná cyklu ordinálneho typu, v1 a v2 sú výrazy rovnakého typu ako rpc a p je príkaz; príkaz for má dva varianty, buď s „to“ alebo s „downto“

Vykonanie príkazu for (v zátvorkách pre downto):

1. vyhodnotia sa výrazy v1 a v2; hodnota výrazu v1 sa priradí ako začiatočná hodnota rpc, hodnota výrazu v2 ako koncová hodnota rpc,
2. pokiaľ je hodnota rpc menšia (väčšia) alebo rovná koncovej hodnote, opakovane sa vykonáva príkaz p a rpc nadobúda hodnoty nasledovníka rpc (predchodcu rpc).

Napríklad

```
for Znak:= 'A' to 'Z' do Memo1.Lines.Add ( IntToStr (ord(Znak)) + '          ' + Znak);
Faktorial:= 1; for Cislo:= N downto 1 do Faktorial:= Faktorial * Cislo;
Ret:= ''; for Znak:= 'a' to ZadanyZnak do Ret:= Ret + Znak + Ret;
```

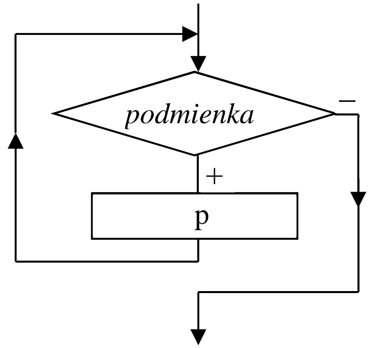
Viete, čo bude výsledkom každého z vyššie napísaných cyklov?

💣 Ešte stále platí, že v tele for-cyklu nemeníme riadiacu premennú cyklu! Teda nie je dobrým programátorským štýlom použiť napríklad for i:= v1 to v2 do begin príkazy; i:= i + 2 end;

## Cyklus s podmienkou ukončenia na začiatku, príkaz while

O typickom použití cyklu s podmienkou ukončenia na začiatku si povieme v časti „Kedy ktorý cyklus“.

Má tvar:




v slovnom zápise:

**pokiaľ** *podmienka* **opakuj** *p*

kde p je príkaz

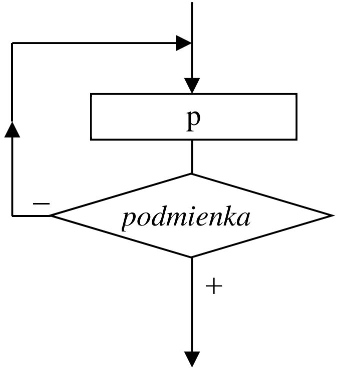
V pascale má tvar: **while** b **do** p

kde b je výraz typu boolean a p príkaz

Vykonanie: Pokiaľ výraz b nadobúda hodnotu true, opakovane sa vykonáva príkaz p, ak výraz b nadobudne hodnotu false, cyklus sa ukončí.

## Cyklus s podmienkou ukončenia na konci, príkaz repeat

Má tvar:




v slovnom zápise:

**opakuj**
*p1;*
*p2;*
...
*pn*
**pokiaľ nebude** *podmienka* (splnená)

kde p1, p2 až pn sú príkazy

V pascale má tvar:

```
repeat
  p1;
  p2;
  ...
  pn
until b
```

kde b je výraz typu boolean a p1 až pn sú príkazy

Vykonanie: Vykonajú sa príkazy p1, p2 až pn a opakovane sa budú vykonávať, pokiaľ výraz b bude nadobúdať hodnotu false. Ak výraz b nadobudne hodnotu true, cyklus sa ukončí.

## Kedy ktorý cyklus

Keď už máme základnú predstavu o fungovaní jednotlivých cyklov a im zodpovedajúcich príkazov, môžeme si poznatky trochu zosystematizovať:

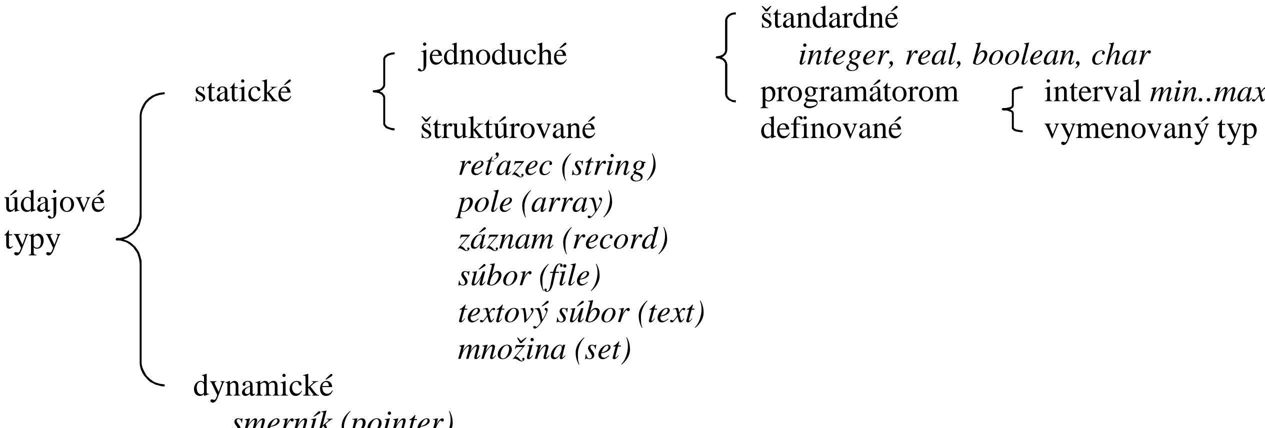
- ak vieme počet opakovaní príkazov v cykle a premenná, ktorá riadi počet prechodov cyklom, sa môže meniť na nasledovníka alebo predchodcu (krok +/- 1 alebo postupnosť za sebou idúcich znakov prípadne vymenovaných hodnôt), použijeme príkaz for
- ak nevieme počet opakovaní príkazov v cykle alebo premenná, ktorá riadi počet prechodov cyklom, nenadobúda „pekné“ hodnoty (hodnoty nasledovníkov alebo predchodcov), použijeme príkazy while alebo repeat, pričom:
  - príkaz while, t.j. s podmienkou na začiatku použijeme, ak môže nastať situácia, že príkaz v cykle sa nemá vykonať ani raz
  - príkaz repeat, t.j. s podmienkou na konci použijeme, ak sa príkazy v cykle majú vykonať aspoň raz.

Uvedomte si, že v každom cykle musí byť premenná, ktorá riadi počet prechodov daným cyklom! Kontrolujte, či sa jej hodnoty zväčšujú alebo zmenšujú a či sú ohraničené podmienkou ukončenia cyklu, ináč cyklus nikdy neskončí.

*Poznámky*:

# Údajové typy

Aj keď väčšinu údajových typov ešte nepoznáte, rozdelenie zvládnete.




**Ordinálne** sú typy integer, boolean, char a programátorom definované typy. Platia pre ne operácie succ, pred a ord. Sú to typy, pre ktoré jednoznačne existuje nasledovník a predchodca; pri reálnom čísle nevieme povedať jeho nasledovníka ani predchodcu, keďže medzi ľubovoľnými dvoma reálnymi číslami existuje nekonečne veľa reálnych čísel ☺.

## Štandardné údajové typy

Meno typu: **Integer** (ordinálny typ)
Množina hodnôt: celé čísla z intervalu $< MaxInt - 1 , MaxInt >$, pričom konštanta MaxInt má v Delphi hodnotu $2^{31} - 1$, čo je 2 147 483 647.
Dovolené operácie: + (sčítanie), - (odčítanie), * (násobenie), div (celočíselné delenie), mod (zvyšok po celočíselnom delení)
Funkcie: abs(x), sqr(x), odd(x), trunc(x), round(x), succ(x), pred(x), ord(x)
Relačné operátory: <, <=, =, <>, >=, >
Vysvetlivky:
Ak nám uvedený rozsah nevyhovuje, môžeme použiť ďalšie celočíselné typy, ktorých rozsahy si môžete pozrieť stlačením F1 s kurzorom v slove integer. Napríklad typ int64 umožňuje použiť celé číslo z intervalu $< -2^{63} , 2^{63} - 1>$, pričom $2^{63}$ je 9 223 372 036 854 775 808.
U celočíselných typov nie je dovolené klasické delenie, lebo jeho výsledkom nemusí byť celé číslo. Dovolené je tzv. celočíselné delenie (celočíselný podiel) div a zvyšok po celočíselnom delení mod. Tieto funkcie si treba ozrejmiť, lebo sa v numerických algoritmoch a programoch často využívajú. Napríklad 15 div 6 = 2 a 15 mod 6 = 3 lebo 15:6 = 2 zvyšok 3, alebo 7 div 10 = 0 a 7 mod 10 = 7 lebo 7:10 = 0 zvyšok 7, alebo 21 div 3 = 7 a 21 mod 3 = 0 lebo 21:3 = 7 zvyšok 0. Všeobecne pre kladné celé čísla A a B platí: Nech A div B = D a A mod B = M potom B.D + M = A a 0 <= M < B. Pre záporné celé čísla funkcie div a mod v Pascale pracujú ináč ako v matematike!
Výsledkom nasledujúcich funkcií je typ integer a to:
**abs**(x) je absolútna hodnota čísla x, napr. abs(8) = 8, abs(-5) = 5, abs(0) = 0
**sqr**(x) je druhá mocnina čísla x, napr. sqt(8) = 64, sqr(-5) = 25, sqr(0) = 0
**odd**(x) je true (pravda), ak x je nepárne číslo, inak false (nepravda), napr. odd(0) = false
**trunc**(x) je celá časť reálneho čísla x, napr. trunc(8.9) = 8, trunc(-5.6) = -5, trunc(0.5) = 0
**round**(x) je „bankársky“ zaokrúhlené reálne číslo x, napr. round(1.5) = 2, round(2.5) = 2 (!)
**succ**(x) je nasledovník x, napr. succ(5) = 6, succ(-1) = 0, succ(-5) = -4
**pred**(x) je predchodca x, napr. pred(5) = 4, pred(-1) = -2, pred(1) = 0
**ord**(x) je poradové číslo x, napr. ord(5) = 5, ord (0) = 0, ord (-12) = -12

Meno typu: **Real** (nie je ordinálny typ!)
Množina hodnôt: niektoré reálne čísla z intervalu v absolútnej hodnote približne $5{,}0.10^{-324}$ po $1{,}7.10^{308}$ s presnosťou na 15-16 platných cifier.
Dovolené operácie: +, -, *, / (delenie)
Funkcie: abs(x), sqr(x), sqrt(x), trunc(x), round(x), int(x), frac(x), sin(x), ln(x), exp(x), arctan(x)
Relačné operátory: <, <=, =, <>, >=, >
Vysvetlivky:
Keďže reálnych čísel v matematike je nekonečne veľa (aj medzi dvoma ľubovoľnými reálnymi číslami) a pamäť počítača je konečná, počítač si dokáže zapamätať len niektoré reálne čísla.
Niektoré funkcie sú popísané vyššie, u typu integer, a málo používané funkcie nepopíšeme.
Výsledkom nasledujúcich funkcií je typ real a to:
**sqrt**(x) je druhá odmocnina z nezáporného čísla x, napr. sqrt(9) = 3, sqrt(2) = 1,4142135624
**int**(x) je celá časť (cifry pred desatinnou bodkou) z reálneho čísla x, napr. int(123.456) = 123.0
**frac**(x) je desatinná časť z reálneho čísla x, napr. frac(123.456) = 0.456
exp(x) je $e^x$
ln(x) je prirodzený logaritmus kladného čísla x[(1)]

Meno typu: **Boolean** (ordinálny typ)
Množina hodnôt: {False, True}
Dovolené operácie: not (negácia), and (logický súčin), or (logický súčet), xor (logický xor)
Funkcie: succ(x), pred(x), ord(x)
Relačné operátory: <, <=, =, <>, >=, >, in (je prvkom)
Vysvetlivky:
Ide o logické hodnoty False – nepravda a True – pravda; false < true.
pred(true) = false, pred(false) – nedefinovaný, succ(false) = true; ord(false) = 0, ord(true) = 1

Meno typu: **Char** (ordinálny typ)
Množina hodnôt: znaky použitej kódovacej tabuľky
Funkcie: upcase(z), succ(z), pred(z), ord(z), chr(x)
Relačné operátory: <, <=, =, <>, >=, >, in
Vysvetlivky:
Prvých 32 znakov (0-31) je riadiacich. Znak s poradovým číslom 32 je medzera atď.
Výsledkom funkcie **chr**(x) je znak zodpovedajúci poradovému číslu x.
Napríklad succ('A') = B, pred('A') = @, ord('A') = 65 a chr(65) = A, chr(ord(z)) = z, ord(chr(x)) = x.
Všimnite si, že ak sa jedná o konkrétny znak, t.j. konštantu, musí byť v apostrofoch, podobne ako reťazcová konštanta. Výraz Znak in [‘0‘..‘9‘] je ekvivalentný matematicky Znak ∈ {‘0‘,‘1‘,...,‘9‘}.
Funkcia **UpCase**(z) zmení malé písmeno na veľké, na iné znaky nemá vplyv, napr. upcase ('b') = B.

Údajové typy, pre ktoré sú definované funkcie succ (successor - nasledovník), pred (predecessor - predchodca) a ord (ordinal - poradové číslo) nazývame **ordinálne** (každá hodnota má presne určené miesto). Sú to typy integer, boolean a char. V Pascale sú pre ne zavedené ešte dva praktické príkazy inc (increase - zväčšiť) a dec (decrease - zmenšiť). Príkaz **inc**(p) zväčší hodnotu premennej p o jednu pozíciu vpravo, napr. inc(5) = 6, inc('A') = B, inc(false) = true a opačne **dec**(p) zmenší hodnotu premennej p o jednu pozíciu vľavo, napr. dec(5) = 4, dec('B') = A, dec(true) = false. Tieto príkazy pracujú podobne ako funkcie succ a pred avšak môžu mať aj tvar **inc**(p,n) resp. **dec**(p,n), kde n je celé číslo (môže byť aj záporné) udávajúce, o koľko treba posunúť hodnotu premennej p. Napr. inc(5,10) = 15, inc('a',10) = k, dec(5,10) = -5, dec('a',5) = W.

---

[1] Posledné dve funkcie možno použiť na výpočet $x^y$ pomocou výrazu exp(y*ln(x)), x kladné reálne a y reálne číslo.

## Štruktúrované údajové typy

Meno typu: **string**
Množina hodnôt: reťazce zo znakov použitej kódovacej tabuľky
Maximálny počet znakov reťazca môžeme ohraničiť zápisom **string[n]** kde n je celé číslo udávajúce maximálnu dĺžku reťazca
Dovolené operácie: + (spája reťazce)
Funkcie: length, uppercase, lowercase, copy, delete, insert, str, val, pos, concat
Relačné operátory: <, <=, =, <>, >=, >
Vysvetlivky:
Napríklad string[5] vyhradí v pamäti miesto pre päťznakový reťazec (pre najviac 5 znakov, ostatné nebudú zapamätané).
Výsledkom funkcie **Length** (reťazec) je celé číslo - dĺžka reťazca. Prázdny reťazec '' má dĺžku 0. Príkazom **SetLength** (reťazec, dĺžka) možno vyhradiť pamäťové miesto pre reťazec. Funkcia **UpperCase**(reťazec) zmení všetky malé písmená angl. abecedy v reťazci na veľké.
Funkcia **LowerCase**(reťazec) zmení všetky veľké písmená angl. abecedy v reťazci na malé.
Funkcia **Copy**(reťazec, od, počet) vytvorí podreťazec z reťazca od pozície „od“ dĺžky „počet“.
Procedúra **Delete**(reťazec, od, počet); odstráni z reťazca „počet“ znakov od pozície „od“.
Procedúra **Insert** (zdroj, reťazec, od); vloží reťazec „zdroj“ do reťazca od pozície „od“.
Procedúra **Str** (číslo [: dĺžka [: des.m.] ]; reťazec); premení „číslo“ na reťazec.
Procedúra **Val** (reťazec, premenná, pozícia); konvertuje reťazec do „premennej“, ktorá je numerického typu, „pozícia“ obsahuje nulovú hodnotu, ak operácia bola úspešná, ináč obsahuje poradové číslo prvého chybného znaku z reťazca.
Funkcia **Pos**(podreťazec, reťazec) vráti prvý výskyt (index) hľadaného podreťazca v reťazci, ak ho nenájde, vráti nulu.
Funkcia **Concat**(r1, r2,..., rN) spojí reťazce r1, r2 až rN.

Ďalšie funkcie na prácu s reťazcami sú súčasťou modulu StrUtils, preto musíme dopísať StrUtils, pri ich použití, do uses unitu. Napríklad:
Funkcie **LeftStr**(reťazec, koľko) a **RightStr**(reťazec, koľko) – vráti podreťazec dĺžky „koľko“ buď zo začiatku alebo z konca reťazca.
Funkcia **ReverseString**(r) otočí reťazec r (vymení prvý znak s posledným atď.).
Funkcie **Trim**(r), **TrimLeft**(r) a **TrimRight**(r) odseknú prázdne znaky (medzery, tabulátory a nové riadky); Trim -na začiatku a konci reťazca, TrimRight – iba na konci, TrimLeft – iba na začiatku.
Funkcia **StringOfChar**(znak, počet) vráti reťazec z daného početkrát opakovaného znaku.

Pri relačných operáciách sa najprv porovnajú prvé znaky v porovnávaných reťazcoch, ak sú zhodné, druhé atď. Napríklad JAN < JANA < JANO < Jana < Jano < Ján < ja

## Typ interval

určuje neprázdnu súvislú podmnožinu hodnôt nejakého ordinálneho typu.
Definícia typu interval má tvar: type *mt* = *min..max*;
kde *mt* je meno typu – identifikátor a *min* a *max* sú dolná a horná hranica hodnôt ordinálneho typu.
Napríklad:

```
type tCislice = '0'..'9';      tPocet = 1..100;      tKladne = 1..MaxInt;
```

## Vymenovaný typ

je typ definovaný vymenovaním hodnôt, preto jeho definícia má tvar:
type *mt* = ( *hodnota1*, *hodnota2*, ..., *hodnotaN* ); kde *mt* je meno typu – identifikátor

Napríklad:
```
type    tDni = (Pondelok, Utorok, Streda, Stvrtok, Piatok, Sobota, Nedela);
        tFarba = (Cervena, Oranzova, Zelena);
```
Každý vymenovaný typ je ordinálny a preto pre neho platia operácie succ, pred a ord.
Napríklad: ord (Pondelok) = 0, succ (Oranzova) = Zelena, Stvrtok < Piatok, succ (Nedela) nedefin.
Nevýhodou vymenovaného typu je, že jeho hodnoty nemožno priamo načítať z klávesnice ani zobraziť na monitore (výstup sa realizuje najčastejšie cez príkaz case).

## Údajový typ množina

použijeme, ak potrebujeme pracovať s podmnožinami nejakého ordinálneho typu.
Definícia typu množina má tvar: type *mt* = set of *bázový typ*;
kde *mt* je meno typu – identifikátor a *bázový typ* určuje typ prvkov množiny – ordinálny typ.
Napríklad:
```
type tFarba = (Cervena, Oranzova, Zelena, Modra, Zlta);
     tOdtien = set of  tFarba;
var   Odtien: tOdtien;
```
hodnoty: [Cervena, Modra], [Oranzova..Zlta] alebo []
Znak in [‘0’..’9‘]
Pismena := [’A’..’Z’]

Operácie: + zjednotenie, * prienik, - rozdiel, = rovnosť, <> nerovnosť, <= „je obsiahnutá v“, >= „obsahuje“

Relácia: *prvok* **in** *množina* – „je prvkom množiny“ - nadobúda hodnoty true alebo false

Príklad použitia:
```
if Retazec[i] in [‘A’..’Z’, ‘a’..’z’] then... // písmeno
```
V procedúre (udalosti) Edit1KeyPress:
```
if not ( Key in [‘0’..’9’, #8] ) then Key := #0;     //ak nebola stlačená číslica alebo BS, nereaguje.
```

## Priorita operácií

Priorita operácií pri vyhodnocovaní výrazov je nasledovná
1. sa vykoná negácia not a zmena znamienka
2. násobenie, delenie, div, mod, and
3. sčítanie, odčítanie, or
4. relačné operátory <, <=, =, <>, >=, >

pri rovnosti operátorov sa výraz vyhodnocuje zľava doprava, zmeniť poradie vykonania operácií možno zátvorkami, pri použití viacerých zátvoriek sa výraz vyhodnocuje od vnútorných zátvoriek k vonkajším.
Napríklad
výraz $\frac{-b+\sqrt{D}}{2a}$ sa zapíše (-b + sqrt(D))/(2*a) alebo (-b + sqrt(D))/2/a
$x \in \langle -1, 0) \vee (0, 1\rangle$ sa zapíš (x >= -1) and (x < 0) or (x > 0) and (x <= 1)

# Procedúry, funkcie a unity

Podprogram je relatívne samostatná programová jednotka (výstavbou podobná programu) riešiaca čiastkový problém. Podprogramy používame najmä:

- ak chceme sprehľadniť program - uvedením riešení jeho čiastkových problémov v podprogramoch alebo
- ak potrebujeme vykonať rovnakú postupnosť príkazov viackrát s rôznymi vstupnými hodnotami (na rôznych miestach programu).

Programovací jazyk Pascal má dva druhy podprogramov, procedúry a funkcie.
Procedúry môžu byť:

- bez lokálnych objektov a bez parametrov,
- s lokálnymi objektmi a bez parametrov a
- s parametrami.

## Procedúry bez lokálnych objektov a bez parametrov

**Deklarácia procedúry bez lokálnych objektov a bez parametrov má tvar:**

```
procedure mp;   { hlavička procedúry }
begin    )
p1;      |
p2;      |
...      }  príkazová časť
pn       |
end;     )
```

kde *mp* je meno procedúry – identifikátor a *p1*, *p2* až *pn* sú príkazy.

Napríklad:

```
procedure VSTUP;
begin
        A := StrToInt ( InputBox (‘Vstup’,‘Zadaj prvé číslo: ‘,‘1‘));
        B := StrToInt ( InputBox (‘Vstup’,‘Zadaj druhé číslo: ‘,‘2‘));
end;

procedure VYMENA;
begin
        POM := A; A := B; B := POM
end;

procedure VYSTUP;
begin
        Label1.Caption := IntToStr (A)  +  ’   ’ + IntToStr (B)
end;
```

Procedúra bez lokálnych objektov a bez parametrov používa len globálne objekty (zavedené v nadradenej časti), hovoríme aj, že komunikuje s okolím len pomocou globálnych premenných. Na mieste, kde chceme, aby došlo k vykonaniu príkazov uvedených v procedúre, stačí uviesť meno procedúry - hovoríme o tzv. volaní procedúry. Napríklad v príkazovej časti ďalšej procedúry:

```
begin          // časť programu využívajúca podpr. VSTUP, VYMENA a VYSTUP
VSTUP;
VYMENA;
VYSTUP;
end;
```

Vidíme, že volanie procedúry je na úrovni príkazu (mená procedúr sme použili ako nami definované príkazy). Všetky použité premenné musia byť deklarované v úseku definícií a deklarácií

nadradeného bloku. Ako v celom Pascale (až na málo výnimiek), aj pri podprogramoch platí, že každý objekt musí byť najprv definovaný alebo deklarovaný a až potom ho môžeme použiť.
Podstatná časť programu v Delphi

```
...
implementation
{$R *.dfm}

var A, B, POM: integer;          // deklarácia globálnych premenných (platia vo všetkých procedúrach)

procedure VSTUP;                 // napísané bez použitia komponentov
begin
        A := StrToInt ( InputBox ('Vstup' , 'Zadaj prvé číslo: ' , '1'));
        B := StrToInt ( InputBox ('Vstup' , 'Zadaj druhé číslo: ' , '2'));
end;

procedure VYMENA;    // napísané bez použitia komponentov
begin
        POM := A; A := B; B := POM
end;

procedure VYSTUP;                // napísané bez použitia komponentov
begin                            // do formulára Form1 nezabudnite vložiť komponent Label
        Label1.Caption := IntToStr (A)  +  '   ' + IntToStr (B)          // *
end;

procedure TForm1.PouziProceduryClick(Sender: TObject);
begin                            // v tejto procedúre voláme všetky tri podprogramy
VSTUP;
VYMENA;
VYSTUP;
end;
end.    // koniec unitu
```

Ak ste dodržali všetko, čo je vyššie uvedené, napriek tomu vám prekladač zahlási v riadku s hviezdičkou chybu [Error] Unit1.pas(42): Undeclared identifier: 'Label1'! Nepozná Label1. Je to spôsobené tým, že doteraz sme procedúry vkladali len dvojklikom na zvolený komponent a Delphi za slovo procedure, pred názov komponentu, automaticky vložilo aj TForm1 a zároveň hlavičku novej procedúry vložilo aj do definície typu TForm1 (pozri v programe v hornej časti unitu za slovom type). Jednoducho povedané, procedúra vie, že ak je v nej použitý nejaký komponent, má ho hľadať vo formulári Form1.
V procedúrach VSTUP a VYMENA sme nepoužili žiadne komponenty vložené do formulára, preto tam problém nenastáva. V procedúre VYSTUP však chceme použiť komponent Label1 a prekladač nevie, „čo je zač“. Musíme mu povedať, že tento objekt nájde vo formulári Form1. Stačí zmeniť Label1.Caption :=... na **Form1**.Label1.Caption :=... a všetko funguje.

🕮 Ak v hlavičke procedúry nie je uvedené, ku ktorému formuláru patrí (chýba TForm1.) a využíva komponenty z formulára, musíme v procedúre pred názvy komponentov dopísať názov formulára, v našom prípade Form1. Existuje aj iná možnosť riešenia tohto problému, tú však považujeme za zložitejšiu.

💣 Nezabudnite, že najprv musíte všetky procedúry deklarovať (napísať) a až potom ich môžete volať (použiť) v procedúre napísanej **pod nimi** v programe!

## Procedúry s lokálnymi objektmi a bez parametrov

Keď sa pozrieme na procedúru VYMENA, ľahko zistíme, že premennú POM potrebujeme len počas vykonávania tejto procedúry. Takýchto objektov môže byť viacej a nie je dôvod zaťažiť pamäť počítača počas behu celého programu vyhradením pamäťových miest objektom, ktoré používame len lokálne. Preto takéto objekty stačí definovať a deklarovať len v danej procedúre, hovoríme, že sú lokálne, čo znamená, že sú použiteľné len v danej procedúre (alebo v podriadených procedúrach).

**Deklarácia procedúry s lokálnymi objektmi má tvar:**

procedure mp;
*úsek definícií a deklarácií* } blok
*príkazová časť*

kde *mp* je meno procedúry.

Pamäťovo efektívnejší zápis procedúry VYMENA:

```
procedure VYMENA;
var POM : integer;                    // lokálna premenná POM
begin
        POM := A; A := B; B := POM
end;
```

Volanie procedúry s lokálnymi objektmi je rovnaké ako procedúry bez lokálnych objektov. Procedúra naďalej komunikuje s okolím len cez globálne premenné.

💣 Ak použijeme rovnaké pomenovanie pre lokálnu aj globálnu premennú, dôjde k tzv. zatieneniu globálnej premennej lokálnou premennou v danej procedúre, čo znamená, že daná globálna premenná je nepoužiteľná v danej procedúre.

Doteraz sme vo všetkých našich procedúrach vytvorených dvojklikom na komponent Button používali len lokálne premenné. Ak by premenná mala byť lokálnou, prekladač nás na to upozorní hlásením: [Warning] Unit1.pas(33): For loop control variable must be simple local variable (stačí, ak deklaráciu var... presuniete nad procedúru). Program napriek tomu bude pracovať správne.

## Procedúry s parametrami

Ak by sme procedúru VYMENA chceli použiť viackrát na výmenu hodnôt rôzne označených premenných (nie len A a B), najvýhodnejšie by bolo použiť parametre. Parametre umožňujú efektívne komunikovať procedúre so svojim okolím, umožňujú hodnoty do procedúry dovážať prípadne aj vyvážať. Pri písaní (deklarovaní) procedúry nemusíme poznať hodnoty, ktoré budú do procedúry napr. dovezené, dokonca ani len označenie premenných, ktoré sa použije pri volaní procedúry. Musíme však poznať počet parametrov a ich typ. Preto deklaráciu procedúry píšeme s tzv. formálnymi parametrami.

**Deklarácia procedúry s parametrami má tvar:**

procedure mp (šfp1; šfp2; ... ; šfpn);
blok;

kde *mp* je meno procedúry a *šfp1* až *šfpn* sú špecifikácie formálnych parametrov.

Napríklad:

```
procedure Vymena ( var X , Y : integer );
procedure Rovnica ( A , B , C : real; var x1, x2 : real );
procedure Zasifruj ( Retazec : string; Posun : integer; var Sifra : string );
```

Ak potrebujeme hodnoty do procedúry len doviesť, použijeme parametre nahradzované hodnotou.
**Špecifikácia parametrov nahradzovaných hodnotou má tvar:**

*fp1, fp2, ... , fpn* : *tfp*

kde *fp1* až *fpn* sú identifikátory formálnych parametrov a *tfp* je identifikátor ich typu.
V príklade hlavičiek procedúr vyššie sú parametre nahradzované hodnotou: A, B, C, Retazec a Posun.
Ak potrebujeme hodnoty z procedúry aj vyviesť, použijeme parametre nahradzované referenciou (odkazom).

**Špecifikácia parametrov nahradzovaných referenciou má tvar:**

var *fp1, fp2, ... , fpn* : *tfp*

kde *fp1* až *fpn* sú identifikátory formálnych parametrov a *tfp* je identifikátor ich typu.
V príklade hlavičiek procedúr vyššie sú parametre nahradzované odkazom: X, Y, x1, x2 a Sifra.

Pri volaní procedúry s parametrami za formálne parametre dosadzujeme skutočné parametre. Ich počet, poradie a typy musia súhlasiť s formálnymi parametrami.
Napríklad:

```
Vymena ( A , B );                              Vymena ( X , Y );
Rovnica ( 5 , -3 , 7 , Koren1, Koren2 );       Rovnica (  a , b , c , k1, k2 );
Zasifruj ( ´CEZAR´, 1 , Vysledok );            Zasifruj ( Sprava , N , Sifra );
```

Pri volaní procedúry môžu byť skutočné parametre nahradzované hodnotou konkrétne hodnoty (tieto hodnoty chceme do procedúry len „doviesť“); pri náhrade referenciou to musia byť premenné, lebo len tak sa môžu získané hodnoty „vyviesť“. Pri náhrade hodnotou sa hodnoty skutočných parametrov pri volaní procedúry len odovzdajú formálnym parametrom. Pri náhrade referenciou sa skutočné parametre, počas vykonávania príkazov v procedúre, stotožnia s formálnymi parametrami – všetky zmeny vykonané s formálnymi parametrami sa vykonajú aj na skutočných parametroch – preto sa hodnoty z procedúry aj „vyvezú“. Pre skutočné parametre môžeme, ale nemusíme, použiť rovnaké označenie ako pre formálne parametre.

## Príklad 7.1

Vytvorte program, ktorý vypíše koľko číslic, veľkých a malých písmen anglickej abecedy a iných znakov je v zadanom reťazci. Vytvorte procedúru s parametrami na zistenie počtu číslic, veľkých a malých písmen anglickej abecedy a iných znakov.

*Analýza*

Formulár môže vyzerať napríklad takto (obrázok vpravo). Aby sme docielili zarovnanie čísel pod seba, nastavili sme v Memo1 – Font písmo Courier o ktorom vieme, že má všetky znaky rovnakej šírky.

Form1
Napíš text 1.7.2010
Počet malých písmen angl. abecedy.......6
Počet veľkých písmen angl. abecedy......1
Počet číslic............................6
Počet iných znakov......................6
Zisti

Popis objektov v procedúre PocetVyskytov:
i lokálna premenná, platí iba v procedúre PocetVyskytov
Ret formálny parameter nahradzovaný hodnotou (reťazec chceme do procedúry len doviesť)
PocetMale, PocetVelke, PocetCislic, PocetIne formálne parametre nahradzované odkazom (ich hodnoty chceme vyviezť z procedúry PocetVyskytov do procedúry TForm1.ZistiClick)

```
procedure PocetVyskytov (Ret: string; var PocetMale, PocetVelke, PocetCislic, PocetIne: integer);
var i: integer;              // lokálna premenná
begin
// nastavenie počiatočných hodnôt parametrov PocetMale, PocetVelke, PocetCislic, PocetIne
// v procedúre s nimi narábame ako s premennými
PocetMale:=0;
PocetVelke:= 0;
```

```
PocetCislic:= 0;
PocetIne:= 0;
// výpočet
for i:= 1 to length(Ret) do                  // zober postupne 1. znak, potom 2.,..., posledný
        case Ret[i] of                       // aj je
                'a'..'z': inc (PocetMale);   // malé písmeno angl. abecedy, ich počet zvýš o 1
                'A'..'Z': inc (PocetVelke);  // veľké písmeno angl. abecedy, ich počet zvýš o 1
                '0'..'9': inc (PocetCislic); // číslica, ich počet zvýš o 1
                else inc (PocetIne)          // inak zvýš počet PocetIne o 1
        end;
end;
```

V procedúre TForm1.ZistiClick sú všetky premenné lokálne (Sender si nevšímame) a v príkaze – volaní procedúry PocetVyskytov (Retazec, Male, Velke, Cifry, Ine); sú aj vo funkcii skutočných parametrov.

```
procedure TForm1.ZistiClick(Sender: TObject);
var Retazec: string;
    Male, Velke, Cifry, Ine: integer;
begin
Memo1.Clear;
Retazec:= Edit1.Text;
PocetVyskytov (Retazec, Male, Velke, Cifry, Ine);        // volanie procedúry s parametrami
// výstup
Memo1.Lines.Add('Počet malých písmen angl. abecedy.......' + IntToStr(Male));
Memo1.Lines.Add('Počet veľkých písmen angl. abecedy......' + IntToStr(Velke));
Memo1.Lines.Add('Počet číslic...........................' + IntToStr(Cifry));
Memo1.Lines.Add('Počet iných znakov.....................' + IntToStr(Ine));
end;
```

## Príklad 7.2

Vytvorte podprogram simulujúci príkaz Val pre typ integer.

*Analýza*

Príkaz Val (reťazec, číslo, kód) konvertuje reťazec na číslo (pozri koniec I. dielu zbierky). Ak nenastala chyba, kód nadobudne hodnotu nula, ak nastala, kód obsahuje index prvého nedovoleného znaku.

Procedúru nazvime ValInt. Doviesť do procedúry treba reťazec – formálny parameter nahradzovaný hodnotou typu string a vyviesť celé číslo a kód – dva formálne parametre nahradzované odkazom typu integer.

Keďže typ integer sú aj záporné celé čísla, môžu nastať dva prípady. Ak prvý znak v reťazci je „-“, musíme ešte výsledok zmeniť na opačné číslo (vynásobiť -1), inak to je nezáporné celé číslo. Najprv si však vyriešte problém len pre nezáporné celé čísla, potom „dorobte“ aj záporné.

Naše riešenie

```
procedure ValInt (Ret: string; var Cislo, Pozicia: integer);
var i: integer;              // Ret -form.par.nahr. hodn., Cislo a Pozicia -form.par.nahr.odkazom
    Chyba: boolean;          // i, Chyba -lokálne premenné
begin
Cislo:= 0;                   // počiatočná hodnota pre číslo (reťazec je určite neprázdny)
Chyba:= False;               // predpokladáme, že chyba nenastane
```

```
if Ret[1] = '-' then i:= 1                 // ak je prvý znak reťazca „-“, začni až nasledujúcim znakom
else i:= 0;                                // začni od začiatku reťazca
repeat                                     // opakuj
  inc(i);                                  // totožné s i:= i + 1;
  if Ret[i] in ['0'..'9']
  then Cislo:= Cislo*10 + ord(Ret[i]) - ord('0')      // pozri príklad 3.4.14
  else Chyba:= True                        // Ret[i] nie je jeden zo znakov „0“ až „9“
until Chyba or (i=length(Ret));            // pokiaľ nenastalo: chyba alebo koniec reťazca
if Chyba then Pozicia:= i                  // ak chyba, treba vyviesť i – index chybného znaku
else begin                                 // nenastala chyba
      Pozicia:= 0;                         // treba vyviesť nulu
      if Ret[1] = '-' then Cislo:= -Cislo;  // reťazec začínal znakom „-“, číslo má byť záporné
      end;
end;
```

Nižšie procedúra, v ktorej voláme procedúru ValInt. Skutočné parametre Cislo a Pozicia (lokálne premenné) sme nazvali rovnako ako vo volanej procedúre (môžete premenovať).
Len pre zaujímavosť sme použili nový komponent LabeledEdit (záložka Additional), ktorý spája komponenty Label a Edit do jedného (používa sa zriedka).

```
procedure TForm1.KonvertujClick(Sender: TObject);
var CisloStr: string;
     Cislo, Pozicia: integer;
begin
if LabeledEdit1.Text = '' then ShowMessage ('Zadaný prázdny reťazec!')
else begin
        CisloStr:= LabeledEdit1.Text;                  // do CisloStr sa uloží zadaný reťazec
        ValInt (CisloStr, Cislo, Pozicia);             // volanie procedúry ValInt so skut.par.
        if Pozicia > 0 then ShowMessage ('Chyba v ' + IntToStr (Pozicia) + '. znaku')
        else Label1.Caption:= IntToStr (Cislo);        // zobrazený správny výsledok
      end;
end;
```

## Príklad 7.3

Vytvorte procedúru so vstupnými parametrami Zaklad a Vstup a výstupnými parametrami Vystup a Chyba, ktorá vykoná prevod čísla (zadané ako string) s celočíselným základom z intervalu <2,9> na desiatkové číslo.
Návod
Pozri príklad 5.1.3.
Vpravo možný vizuálny návrh.
Riešenie

```
procedure Prevod (Zaklad: integer; Vstup: string; var Vystup: integer;
var Chyba: boolean);
var   i: integer;
      Znak: char;
begin
Znak := chr ( ord('0')+Zaklad-1 );
Chyba:= False;
Vystup:= 0;
i:= 0;
```

```
repeat
        i:= i + 1;
        if Vstup[i] in ['0'..Znak] then Vystup:= Vystup*Zaklad + ord(Vstup[i]) - ord('0')
        else Chyba:= True;
until ( i = length(Vstup) ) or Chyba;
end;

procedure TForm1.btPrevodClick(Sender: TObject);
var   CisloZStr: string;
      Zaklad, Cislo10, i: integer;
      Chyba: boolean;
begin
Zaklad:= StrToInt (Edit1.Text);
CisloZStr:= Edit2.Text;
Prevod (Zaklad,CisloZStr,Cislo10,Chyba);
if not Chyba
then Label3.Caption:= IntToStr (Cislo10)
else Label3.Caption:= 'Chybný znak'
end;
```

💻 Program upravte tak, aby, ak nastala chyba, sa cez parameter Vystup vyviezol index prvého chybného znaku a aj vypísal v chybovom hlásení.

## Funkcie

Funkcia je špeciálnym prípadom procedúry. Procedúru môžeme zapísať ako funkciu, keď jej výsledkom je hodnota jednoduchého typu (integer, real, boolean, char) alebo typu string.

**Deklarácia funkcie má tvar:**

```
function mf ( šfp1; sfp2; ... ; šfpn ) : tvf;
   blok;
```

kde *mf* je meno funkcie – identifikátor, šfp1 až šfpn sú špecifikácie formálnych parametrov a tvf je typ výsledku funkcie, ktorý musí byť jednoduchý typ alebo typ string.

Napríklad:

```
function Maximum ( A , B , C : integer ) : integer;
function Nachadza_sa : boolean;
function Mocnina ( X : real; N : integer ) : real;
```

Výsledok sa z funkcie vyváža cez meno funkcie, preto identifikátor mena funkcie sa musí aspoň raz vyskytnúť na ľavej strane príkazu priradenia v tele danej funkcie. Aktivizácia funkcie (vykonanie príkazov v tele funkcie) sa deje uvedením tzv. zápisu funkcie (mena funkcie a skutočných parametrov) a nie je na úrovni príkazu. Prakticky to znamená, že zápis funkcie sa uvedie na mieste, kde chceme dosadiť výsledok funkcie.

Napríklad

```
Max := Maximum ( 5 , -3, 2 );   alebo Max:= Maximum (Cislo1, Cislo2, Cislo3);
if Nachadza_sa then...
Label1.Caption := Format (´%f  na  %d  =  %f ´, [X, N, Mocnina (Zaklad, Exponent)] );
```

### Príklad 7.4

Vytvorte funkciu, ktorá vráti väčšie z dvoch reálnych čísel.

*Analýza a algoritmus*

Výsledkom podprogramu je jedna jednoduchá hodnota – reálne číslo, preto môžeme použiť funkciu a algoritmus je veľmi jednoduchý: ak Cislo1 > Cislo2 tak „vyvez“ Cislo1 inak „vyvez“ Cislo2.
Do funkcie treba len doviesť dve reálne čísla – dva formálne parametre nahradzované hodnotou.

Riešenie

```
function Maximum (Cislo1, Cislo2: real): real;
begin
if Cislo1 > Cislo2 then Maximum:= Cislo1 else Maximum:= Cislo2
end;
```

Použitie napríklad

```
var X,Y: real;
begin
X:= StrToFloat (Edit1.Text);
Y:= StrToFloat (Edit2.Text);
Label3.Caption:= ‘Väčšie má hodnotu ‘ + FloatToStr (Maximum (X, Y));
end;
```

💣 Pri deklarovaní funkcie programátorom sa automaticky vytvorí aj premenná **Result** typu výsledku funkcie, a programátor ju môže použiť pri výpočte vo funkcii, aj na vyvezenie výsledku funkcie (nemusí použiť meno funkcie).

Takže funkcia Maximum môže mať aj tvar

```
function Maximum (Cislo1, Cislo2: real): real;
begin
if Cislo1 > Cislo2 then Result:= Cislo1 else Result:= Cislo2
end;
```

Zaujímavé je aj riešenie

```
function Maximum (Cislo1, Cislo2: real): real;
begin
Result:= Cislo1;
if Cislo2 > Result then Result:= Cislo2;
end;
```

POZOR! Nemôžeme použiť zápis

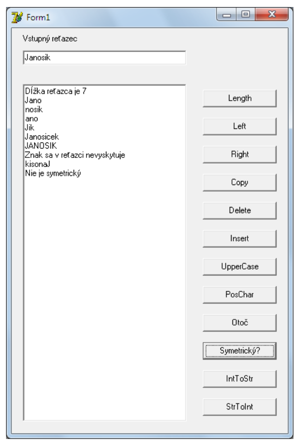
```
function Maximum (Cislo1, Cislo2: real): real;
begin
Maximum:= Cislo1;
if Cislo2 > Maximum then Maximum:= Cislo2;
end;
```

Meno funkcie sa v tele funkcie môže vyskytnúť len na ľavej strane príkazu priradenia, inak si „počítač myslí“, že voláme funkciu Maximum a chýbajú mu skutočné parametre, ale to je už iná „pesnička“ (programovacia technika).
Podstatný rozdiel v použití mena funkcie a premennej Result v tele funkcie je práve v tom, že Result bude vždy pochopená len ako premenná (meno funkcie môže byť pochopené aj ako zápis funkcie, t.j. očakáva sa jej vykonanie na danom mieste).

✍ 💻 Vytvorte a použite funkciu, ktorá vráti najväčšie z troch reálnych čísel.

✍ 💻 Vytvorte a použite funkcie na výpočet súčtu, rozdielu, súčinu a podielu dvoch celých (reálnych) čísel.

Príklady z kapitoly Pohrajme sa s reťazcami z prvého dielu zbierky sú ideálne na precvičenie tvorby funkcií. Keďže sme ich rozbor spravili už v prvom dieli zbierky, poslúžia skôr ako ukážky. Aby sa pôvodné názvy funkcií „nebili“ s našim pomenovaním, pred oficiálny názov funkcie budeme vkladať písmeno f, napríklad fLength. Všetky funkcie môžete opäť odladiť v jednom formulári s jedným vstupným komponentom Edit a výstupným komponentom Memo (obrázok vpravo).




## Príklad 7.5

Naprogramujte funkciu Length, resp. vytvorte a použite funkciu na zistenie dĺžky reťazca.
Pozri príklad 3.4.1

```
function fLength (Retazec: string): integer;
var Dlzka: integer;
begin
Retazec := Retazec + #0;          // vloženie nárazníka
Dlzka:= 0;
while Retazec[Dlzka+1] <> #0 do Dlzka:= Dlzka + 1;
Result:= Dlzka;
end;
```

Ďalšia alternatíva funkcie fLength:

```
...
begin
Dlzka:= 0;
if Retazec <> ''
then begin
        Retazec:= Retazec + #0;
        repeat inc(Dlzka)
        until  Retazec[Dlzka+1] = #0;
    end;
Result:= Dlzka;
end;
```

Použitie funkcie fLength

```
procedure TForm1.BtLengthClick(Sender: TObject);
var Ret: string;
begin
Ret:= Edit1.Text;
Memo1.Lines.Add ( 'Dĺžka reťazca je ' + IntToStr ( fLength(Ret) ) );
end;
```

💻 Dokážete predchádzajúcu procedúru upraviť tak, aby ste nepoužili lokálnu premennú Ret?

## Príklad 7.6

Naprogramujte funkciu LeftStr, resp. vytvorte funkciu, ktorá z dovezeného reťazca vráti zadaný počet znakov zľava.

Pozri príklad 3.4.2.
Napríklad LeftStr ( 'Janosik' , 4 ) vráti reťazec Jano. Všeobecne LeftStr (Retazec, Kolko) vytvorí nový reťazec, do ktorého prenesie z Retazec prvých Kolko znakov.

```
function fLeft (Retazec: string; Kolko: integer): string;
var i: integer;
begin
setlength(Result, Kolko);
for i:=1 to Kolko do Result[i]:= Retazec[i];
end;
```

Použitie funkcie fLeft

```
procedure TForm1.LeftClick(Sender: TObject);
var Ret: string;
    Kolko: integer;
begin
Ret:= Edit1.Text;
Kolko:= StrToInt (InputBox ('Funkcia Left' , 'Koľko znakov' , ''));
Memo1.Lines.Add ( fLeft (Ret, Kolko) );
end;
```

💻 Funkciu fLeft vytvorte aj bez použitia príkazu setlength.

## Príklad 7.7

Naprogramujte funkciu RightStr, resp. vytvorte funkciu, ktorá z dovezeného reťazca vráti zadaný počet znakov sprava.
Pozri príklad 3.4.3.
Napríklad RightStr ( 'Janosik' , 3 ) vráti reťazec sik. Všeobecne RightStr (Retazec, Kolko) vytvorí nový reťazec, do ktorého prenesie z Retazec posledných Kolko znakov. Použili sme algoritmus, ktorý prenesie posledný znak z reťazca, pred neho dá predposledný znak atď.

```
function fRight (Retazec: string; Kolko: integer): string;
var Dlzka, i: integer;
begin
Dlzka:= length (Retazec) + 1;          // +1 aby hodnoty indexov vo for-cykle boli krajšie ☺
Result:= '';
```

```
for i:=1 to Kolko do Result:= Retazec[Dlzka - i] + Result;
end;
```

Použitie funkcie fRight

```
procedure TForm1.RightClick(Sender: TObject);
var Ret: string;
    Kolko: integer;
begin
Ret:= Edit1.Text;
Kolko:= StrToInt (InputBox ('Funkcia Right' , 'Koľko znakov' , ''));
Memo1.Lines.Add ( fRight (Ret, Kolko) );
end;
```

### Príklad 7.8

Naprogramujte funkciu Copy, resp. vytvorte funkciu, ktorá z dovezeného reťazca vráti od zadaného čísla znaku zadaný počet znakov.
Pozri príklad 3.4.4.
Napríklad Copy (‘Janosik‘ , 2 , 3 ) vráti ano; Copy (‘Janosik‘ , 3 , 3 ) vráti nos.

```
function fCopy (Retazec: string; Od, Kolko: integer): string;
var i: integer;
begin
Result:= '';
for i:= Od to Od+Kolko-1 do Result:= Result + Retazec[i];
end;
```

Použitie funkcie fCopy

```
procedure TForm1.BtCopyClick(Sender: TObject);
var Ret: string;
    Od, Kolko: integer;
begin
Ret:= Edit1.Text;
Od:= StrToInt (InputBox ('Funkcia Copy' , 'Od koľkého znaku' , ''));
Kolko:= StrToInt (InputBox ('Funkcia Copy' , 'Koľko znakov' , ''));
Memo1.Lines.Add ( fCopy ( Ret, Od, Kolko ) );
end;
```

### Príklad 7.9

Naprogramujte funkciu Delete, resp. vytvorte funkciu, ktorá odstráni z dovezeného reťazca, od zadaného čísla znaku zadaný počet znakov. Napríklad Delete (‘Janosik‘ , 4 , 2) vráti Janik.

```
function fDelete(Retazec: string; Od, Kolko: integer):string;
var i: integer;
begin
Result:= '';
for i:=1 to length(Retazec) do if (i<Od) or (i>Od+Kolko-1) then Result:= Result + Retazec[i];
end;
```

Použitie funkcie fDelete

```
procedure TForm1.BtDeleteClick(Sender: TObject);
var Ret: string;
    Od, Kolko: integer;
begin
Ret:= Edit1.Text;
Od:= StrToInt (InputBox ('Funkcia Delete' , 'Od koľkého znaku' , ''));
Kolko:= StrToInt (InputBox ('Funkcia Delete' , 'Koľko znakov' , ''));
Memo1.Lines.Add ( fDelete ( Ret, Od, Kolko ) );
end;
```

### Príklad 7.10

Naprogramujte funkciu Insert, resp. vytvorte funkciu, ktorá vloží zadaný reťazec do zadaného reťazca, od zadaného čísla znaku. Napríklad Insert (‘ce‘ , ‘Janosik‘ , 7) vráti Janosicek.
Použite funkciu fCopy prípadne aj fLength.

```
function fInsert (VlozitRetazec, DoRetazec: string; Od: integer): string;
begin
fInsert:= fCopy(DoRetazec, 1, Od-1)+VlozitRetazec+fCopy (DoRetazec, Od, fLength (DoRetazec)-Od+1);
end;
```

Použitie funkcie fInsert

```
procedure TForm1.BtInsertClick(Sender: TObject);
var DoRet, VlozitRet: string;
   Od: integer;
begin
DoRet:= Edit1.Text;
VlozitRet:= InputBox ('Funkcia Insert' , 'Vložiť reťazec' , '');
Od:= StrToInt (InputBox ('Funkcia Insert' , 'Od koľkého znaku' , ''));
Memo1.Lines.Add ( fInsert ( VlozitRet, DoRet, Od ) );
end;
```

### Príklad 7.11

Naprogramujte funkciu UpperCase, resp. vytvorte funkciu, ktorá zmení v dovezenom reťazci malé písmená anglickej abecedy na veľké, ostatné znaky nemení.
Pozri príklad 3.4.6.

```
function fUpperCase (Retazec: string): string;
const POSUN = ord('a') - ord('A');
var    i: integer;
begin
setlength(Result,length(Retazec));
for i:=1 to length (Retazec) do
        if Retazec[i] in ['a'..'z']
        then Result[i]:= chr (ord (Retazec[i]) - POSUN)
        else Result[i]:= Retazec[i];
end;
```

Použitie funkcie fUpperCase

```
procedure TForm1.BtUpperCaseClick(Sender: TObject);
var Ret: string;
begin
Ret:= Edit1.Text;
Memo1.Lines.Add ( fUpperCase (Ret) );
end;
```

✍ 💻 Funkcia LowerCase zmení v zadanom reťazci veľké písmená anglickej abecedy na malé, ostatné znaky nemení. Naprogramujte ju.

## Príklad 7.12

Naprogramujte funkciu, ktorá zistí miesto prvého výskytu zadaného znaku v reťazci. Ak sa znak v reťazci nenachádza, vráti nulu.
Pozri príklad 3.4.7.

```
function fPosChar (Retazec: string; Znak:char): integer;
var i: integer;
begin
Retazec:= Retazec + Znak;
i:= 0;
repeat
        i:= i + 1
until Retazec[i] = Znak;
if i = length(Retazec) then Result:= 0
else Result:= i
end;
```

Použitie funkcie fPosChar

```
procedure TForm1.PosCharClick(Sender: TObject);
var Ret: string;
    Znak: char;
    Poz: integer;
begin
Ret:= Edit1.Text;
Znak:= InputBox ( 'PosChar' , 'Hľadať znak' , '')[1];
Poz:= fPosChar (Ret, Znak);                          // uloženie hodnoty funkcie do premennej
if Poz = 0
then Memo1.Lines.Add ('Znak sa v reťazci nevyskytuje')
else Memo1.Lines.Add ('Prvý výskyt znaku na indexe ' + IntToStr (Poz))
end;
```

🕮 Ak výsledok funkcie potrebujeme v procedúre použiť viackrát, je efektívnejšie ho uložiť do premennej a používať jej hodnotu, ako opakovane používať funkčný výraz.

## Príklad 7.13

Vytvorte funkciu, ktorá obráti zadaný reťazec, t.j. vymení prvý znak s posledným, druhý s predposledným atď.
Pozri príklad 3.4.9.

```
function Otoc (Retazec: string): string;
var Dlzka, i: integer;
begin
Dlzka:= length (Retazec);
setlength (Result, Dlzka);
for i:= 1 to Dlzka do Result[i]:= Retazec[Dlzka -i + 1];
end;
```

Použitie funkcie Otoc

```
procedure TForm1.BtOtocClick(Sender: TObject);
var Ret: string;
begin
Ret:= Edit1.Text;
Memo1.Lines.Add (Otoc ( Ret ) );
end;
```

## Príklad 7.14

Vytvorte funkciu, ktorá zistí, či zadaný reťazec je symetrický.
Pozri príklad 3.4.10.

```
function Symetricky (Retazec: string): boolean;
var Dlzka, i: integer;
begin
Dlzka:= length (Retazec);
i:= 1;
while (Retazec[i] = Retazec[Dlzka-i+1]) and (i < Dlzka div 2) do i:= i + 1;
if Retazec[i] = Retazec[Dlzka-i+1] then Result:= True
else  Result:= False
end;
```

Použitie funkcie Symetricky

```
procedure TForm1.BtSymetrickyClick(Sender: TObject);
var Ret: string;
begin
Ret:= Edit1.Text;
if Symetricky (Ret) then Memo1.Lines.Add ('Je symetrický')
else Memo1.Lines.Add ('Nie je symetrický')
end;
```

Funkciu Symetricky upravte tak, aby po dovezení prázdneho reťazca nedošlo k chybovému hláseniu ale sa vyviezla hodnota True.

## Príklad 7.15

Naprogramujte funkciu IntToStr, t.j. funkciu, ktorá prekonvertuje celé číslo na reťazec.
Pozri príklad 3.4.14.

```
function fIntToStr (Cislo: integer): string;
var Cifra: integer;
    Zaporne: boolean;
begin
if Cislo < 0 then begin Zaporne:= True; Cislo:= -Cislo end
else Zaporne:= False;
Result:= '';
```

```
repeat
        Cifra:= Cislo mod 10;
        Result:= chr(ord('0') + Cifra) + Result;
        Cislo:= Cislo div 10;
until Cislo = 0;
if Zaporne then Result:= '-' + Result;
end;
```

Použitie funkcie fIntToStr

```
procedure TForm1.BtIntToStrClick(Sender: TObject);
var Cislo: integer;
begin
Cislo:= StrToInt (Edit1.Text);
Memo1.Lines.Add ( fIntToStr ( Cislo ) );
end;
```

### Príklad 7.16

Naprogramujte funkciu StrToInt, t.j. funkciu, ktorá prekonvertuje reťazec na celé číslo.
Pozri príklad 3.4.15.

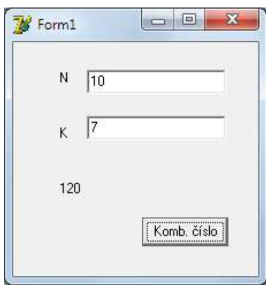
```
function fStrToInt (Retazec: string): integer;
var i: integer;
begin
Result:= 0;
for i:= 1 to length(Retazec) do Result:= Result*10 + ord( Retazec[i] ) - ord( '0' );
end;
```

Použitie funkcie fStrToInt

```
procedure TForm1.BtStrToIntClick(Sender: TObject);
var CisloStr: string;
begin
CisloStr:= Edit1.Text;
Memo1.Lines.Add ('Dvojnásobok je ' + IntToStr ( 2 * fStrToInt ( CisloStr ) ) );
end;
```

## Kedy použiť lokálne objekty, kedy parametre?

Kedy je výhodnejšie použiť lokálne objekty a kedy ich nemôžeme použiť? Odpoveď znie: všetky premenné (objekty), ktorých hodnoty nepotrebujeme mimo procedúru, t.j. slúžia len na výpočty vo vnútri procedúry, je dobré (nie nevyhnutné) deklarovať ako lokálne. Čiže napríklad premennú cyklu „i" síce použijeme vo viacerých cykloch v rôznych procedúrach, ale použijeme ju len na riadenie toho ktorého cyklu a mimo procedúr jej hodnoty nepotrebujeme, preto je lepšie, ak je lokálnou premennou v každej použitej procedúre! Lokálne objekty šetria pamäť. Zamedzujú, aby ich hodnota, pokiaľ by boli globálne, nechtiac ovplyvnila výpočet mimo procedúru. Čiže lokálnu premennú zmeníme na globálnu, len ak chceme, aby sa jej hodnota „vyviezla" aj mimo procedúru. Na to však môžeme použiť aj parametre nahradzované odkazom.




Ako to je s parametrami, si vysvetlíme na takomto príklade. Program počíta kombinačné čísla C (n, k) = $\frac{n!}{(n-k)!.k!}$ , kde n a k sú prirodzení čísla, symbol ! znamená výpočet faktoriálu. Napríklad 5! = 5.4.3.2.1 = 120 (0! je definovaný ako 1). Preto napríklad C (10,7) = $\frac{10!}{(10-7)!.7!}$ = 120.

**Použijeme parameter X**

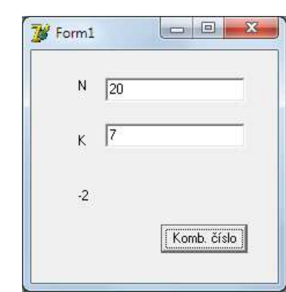
```
function Fakt (X: integer):integer;
var i: integer;
begin
Result:= 1;
for i:= X downto 1 do Result:= Result * i;
end;

procedure TForm1.KombinacieClick(Sender: TObject);
var N, K: integer;
begin
N:= StrToInt (Edit1.Text);
K:= StrToInt (Edit2.Text);
Label3.Caption:= IntToStr ( Fakt(N) div ( Fakt(N-K) * Fakt(K) ) );
end;
```

**Bez parametra X**

```
var X: integer;                    // X musí byť globálna premenná, dovezie sa ňou do funkcie
                                   // Fakt aktuálna hodnota, pre ktorú chcem vypočítať faktoriál
function Fakt:integer;             // vypočítaná hodnota sa vyvezie cez meno funkcie
var i: integer;
begin
Result:= 1;
for i:= X downto 1 do Result:= Result * i;
end;

procedure TForm1.KombinacieClick(Sender: TObject);
var N, K, Nf, NKf, Kf: integer;    // museli sme pridať nové premenné
begin
N:= StrToInt(Edit1.Text);
K:= StrToInt(Edit2.Text);
X:= N;                             // faktoriál chcem vypočítať pre N
Nf:= Fakt;                         // výsledok musíme odložiť (do Nf)
X:= N-K;                           // faktoriál chceme vypočítať pre N-K
NKf:= Fakt;                        // výsledok musíme odložiť (do NKf)
X:= K;                             // faktoriál chceme vypočítať pre K
Kf:= Fakt;
Label3.Caption:= IntToStr ( Nf div ( NKf * Kf ) );          // výpočet kombinačného čísla
end;
```

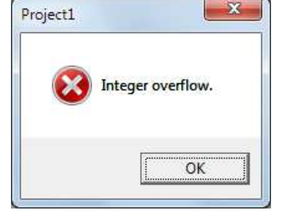
V takomto jednoduchom príklade sme museli na každé volanie procedúry s jedným parametrom pridať ďalšie dva riadky (spolu 6) len preto, že sme nepoužili parameter a naviac sme museli deklarovať ďalšie tri premenné (Nf, NKf, Kf)!

📖 Parametre „musíme“ (rozumej „je efektívne“) použiť vždy vtedy, keď chceme tú istú procedúru použiť viackrát v programe s rôznymi vstupnými hodnotami. Ak danú procedúru použijeme (voláme) v programe len raz, resp. stále pracujeme len s tým istým objektom (napríklad stále vypisujeme hodnotu X, trebárs aj zmenenú) je jednoduchšie daný objekt deklarovať ako globálny a parameter nepoužiť. Ukážeme si to pri údajovom type pole.




💣 Pri výpočte n! už pre malé hodnoty n je výsledkom veľké číslo, ktoré sa veľmi ľahko dostane mimo výpočtový rozsah programu (MaxInt = 2 147 483 647 a už 13! = 6 227 020 800; pri použití údajového typu int64 prvý chybný výsledok dostávame pre 21!). O chybnom výsledku sa nemusíme ani dozvedieť, veď keby sme poznali výsledok, tak ho nedáme počítať počítaču ☺. Upozorniť nás môže neočakávaná hodnota, ako napríklad pri N = 20 a K = 7 program vypíše -2, čo je zrejme neprípustný výsledok (na to sa však nespoliehajte, lebo už od N = 13 určite nie je správny!).

Zapnutím Overflow checking (kontrola pretečenia) v paneli Project – Options... – Compiler (obrázok vpravo) síce nezískate vyriešenie problému, ale vás aspoň počítač počas behu programu upozorní na pretečenia nad maximálnu hodnotu hlásením Integer overflow (obrázok vpravo). Aby sa kontrola pri konkrétnom programe prejavila, musí dôjsť ku novej kompilácii, preto musíte v programe spraviť nejakú zmenu a opätovne ho spustiť. V zobrazenom paneli môžete zapnúť aj kontrolu Range checking.




Ujasnite si pravdivé tvrdenie: ak je celočíselný výsledok výpočtu v dovolenom rozsahu (pre integer od -MaxInt-1 po MaxInt), je absolútne presný. Preto, pokiaľ to je možné, sa snažíme počítať s celočíselným údajovým typom.

Horšie to je s reálnymi číslami, pri ktorých vieme, že napríklad medzi dvoma reálnymi číslami je nekonečne veľa reálnych čísel alebo že výsledok výpočtu môže mať nekonečne veľa cifier. Obe tvrdenia pri aplikácii v počítači narážajú na konečnú pamäť počítačov a jej dôsledkom je skutočnosť, že väčšina reálnych čísel nie je v počítači zobraziteľná.

Nižšie uvedená procedúra nájde číslo eps, ktoré spĺňa tvrdenie, že je približne najväčším číslom, pre ktoré ešte platí 1 + eps = 1.

```
procedure TForm1.btPresnostClick(Sender: TObject);
var eps, Vysledok: real;
begin
eps:= 1;
repeat
        eps:= 0.5*eps;
        Vysledok:= 1 + eps;
until Vysledok = 1;                                         //my sme dostali výsledok
Memo1.Lines.Add('Presnosť výpočtu = ' + FloatToStr(eps));   //1,11022302462516E-16
end;
```

**Neverte bezhranične počítaču!**

## Unity

Kým procedúry sú základom štruktúrovaného programovania, unity – programové jednotky, moduly sú základom modulárneho programovania. Unity sú samostatne kompilovateľné súbory (procedúry nie). Štruktúra unitu je podobná štruktúre programu s jedným podstatným rozdielom, modul má štyri základné sekcie: interface, implementation, initialization a finalization.

Časť **interface** (rozhranie, komunikácia s okolím)

Začína vyhradeným slovom inteface a patrí do nej všetko až po začiatok ďalšej sekcie uvedenej slovom implementation. Obsahuje deklarácie konštánt, typov, premenných, procedúr a funkcií, ktoré majú byť verejné – prístupné pre ostatné moduly a hlavný program. Pri procedúrach a funkciách v časti interface uvádzame len ich hlavičky, telá píšeme až v implementačnej časti.

Časť **implementation** (aplikovanie, použitie)

Začína vyhradeným slovom implementation a pokračuje až po začiatok inicializačnej časti alebo po koniec modulu (ak inicializačná časť chýba). Obsahuje úplné deklarácie procedúr a funkcií uvedených v interfejsovej časti. Obsahuje tiež deklarácie konštánt, typov, premenných, procedúr a funkcií, ktoré nechceme, aby boli mimo unit prístupné – tzv. súkromné (neverejné) objekty. Tieto prvky môžu využívať len podprogramy daného unitu.

Pokiaľ nerobíme programy, v ktorých sú údaje prenášané medzi unitmi, „vystačíme" s implementačnou časťou. To znamená, že aj globálne premenné stačí deklarovať v tejto časti, t.j. pod príkazom {$R *.dfm} (príkaz pre prekladač)

```
implementation
{$R *.dfm}
// naše globálne definície a deklarácie
```

Časť **initialization** (inicializácia, nastavenie počiatočných hodnôt)

Je nepovinná. Uvádzame v nej príkazy, ktoré sa majú vykonať hneď po spustení programu, ktorý v časti uses obsahuje meno „nášho" unitu.

Časť **finalization** (ukončenie, uzavretie)

Je nepovinná a nesmie byť použitá bez časti initialization. Obsahuje príkazy, ktoré sa majú vykonať v okamžiku ukončenia hlavného programu.

Vytvorený modul je možné použiť v ľubovoľnom programe, názov modulu sa však musí uviesť v sekcii uses daného programu.

Jednoduchá ukážka – nad end s bodkou dopíšte dva riadky a spustite.

```
...
implementation
{$R *.dfm}

initialization
ShowMessage ( 'Vitaj!' );
end.
```

💣 Do inicializačnej časti nesmieme napísať príkazy, ktoré využívajú formulár, napr. výstup cez Label alebo Memo, lebo formulár sa vytvorí až po vykonaní príkazov z časti initialization. Došlo by ku chybe počas behu programu (Runtime error...). Ani do časti finalization nemôžeme napísať „hocijaké" príkazy.

# Rekurzia

je programovacia technika, ktorá umožňuje elegantne riešiť niektoré úlohy. Elegancia spočíva najmä v jednoduchosti zápisu algoritmu. Rekurziu možno použiť, ak riešenie nejakého problému vedie k riešeniu analogických problémov smerujúcich až k triviálnemu problému.
Napríklad:
Rekurentná definícia výpočtu n-faktoriálu pre $n \geq 0$ hovorí:

1. 0! = 1 {výpočet 0! - triviálny problém}
2. n! = n * ( n- 1)! pre n > 0 {výpočet n! vedie k analogickému problému, výpočtu (n-1)!}

V zmysle rekurentnej definície možno napr. 5! vypočítať ako 5*4! = 5*(4*3!) = 5*(4*(3*2!)) = 5*(4*(3*(2*1!))) = 5*(4*(3*(2*(1*0!)))) = 5*(4*(3*(2*(1*1)))) = 120 (zátvorky naznačujú, ako sa uskutoční výpočet)

```
function NFaktorial (N: integer): integer;
begin
if N = 0 then NFaktorial := 1                // nerekurzívna vetva
else NFaktorial := N * NFaktorial (N-1)      // rekurzívna vetva, volanie funkcie NFaktorial
end;
```

Rekurentná definícia výpočtu mocniny $x^n$ pre $x \in R-\{0\}$ a $n \in N_0$:

1. $x^0 = 1$ {výpočet $x^0$ - triviálny problém}
2. $x^n = x * x^{n-1}$ pre n > 0 {výpočet $x^n$ vedie k analogickému problému, výpočtu $x^{n-1}$}

```
function Mocnina (N: integer; X: real): real;
begin
if N = 0 then Mocnina := 1                   // nerekurzívna vetva
else Mocnina := X * Mocnina (N-1, X)         // rekurzívna vetva, volanie funkcie Mocnina
end;
```

Pripomíname iteračné (iterácia – opakovanie) riešenia:

```
function NFaktorial (N: integer): integer;
var Fakt, i : integer;
begin
Fakt := 1;
for i :=1 to N do Fakt := Fakt * i;
NFaktorial := Fakt
end;
```

```
funkction Mocnina (N: integer; X: real): real;
var Moc: real; i: integer;
begin
Moc := 1;
for i := 1 to N do Moc := Moc * X;
Mocnina := Moc
end;
```

Každá rekurzia obsahuje rekurzívnu vetvu, v ktorej je vlastne schovaný cyklus – opakované volanie procedúry, a nerekurzívnu vetvu, ktorá zabezpečuje ukončenie „cyklu“. Aktuálne hodnoty premenných pred vnorením do ďalšej procedúry sa odkladajú do zásobníka. Rekurzívne riešenie úlohy, v porovnaní s iteračným, je časovo aj pamäťovo náročnejšie!
V uvedených príkladoch sa jednalo o **priamu rekurziu**, keďže procedúra vo svojom tele volá samu seba. Ďalšou možnosťou je **vzájomná rekurzia**, keď procedúra A vo svojom tele obsahuje volanie procedúry B a tá obsahuje vo svojom tele volanie procedúry A. Vzájomná rekurzia si vyžaduje použitie direktívy forward. Vzájomnou rekurziou sa nebudeme zaoberať. Ak rekurzívna funkcia volá samu seba ako posledný príkaz (rekurzívny výpočet faktoriálu aj mocniny), hovorí sa aj **o chvostovej rekurzii**.

## Príklad R1

Napíšte rekurzívnu procedúru na nájdenie najväčšieho spoločného deliteľa dvoch prirodzených čísel, ak viete, že:

1. NSD (a, 0) = a resp. NSD (0, b) = b
2. pre $a \neq 0$ a zároveň $b \neq 0$ NSD (a,b) = $\begin{cases} \text{NSD (a mod b, b)} & \text{ak } a > b \\ \text{NSD (a, b mod a)} & \text{ak } b > a \end{cases}$

*Riešenie*

Naprogramujte si algoritmus tak, ako je uvedený v zadaní. Nižšie ďalšia alternatíva.

```
function NSD(a,b: integer):integer;
begin
if b=0
then NSD:=a
else NSD:= NSD(b, a mod b);
end;

procedure TForm1.btNSDClick(Sender: TObject);
var A,B: integer;
BEGIN
A:= StrToInt(InputBox( 'NSD' , 'A' , '12') );
B:= StrToInt(InputBox( 'NSD' , 'B' , '15') );
Memo1.Lines.Add(Format( 'NSD(%d,%d) = %d' , [ A , B , NSD(A,B) ] ));
END;
```

Ďalšou možnosťou je využitie algoritmu

1. ak a = b NSD (a, b) = a resp. b
2. pre $a \neq b$ NSD (a,b) = $\begin{cases} \text{NSD (a - b, b)} & \text{ak } a > b \\ \text{NSD (a, b - a)} & \text{ak } b > a \end{cases}$

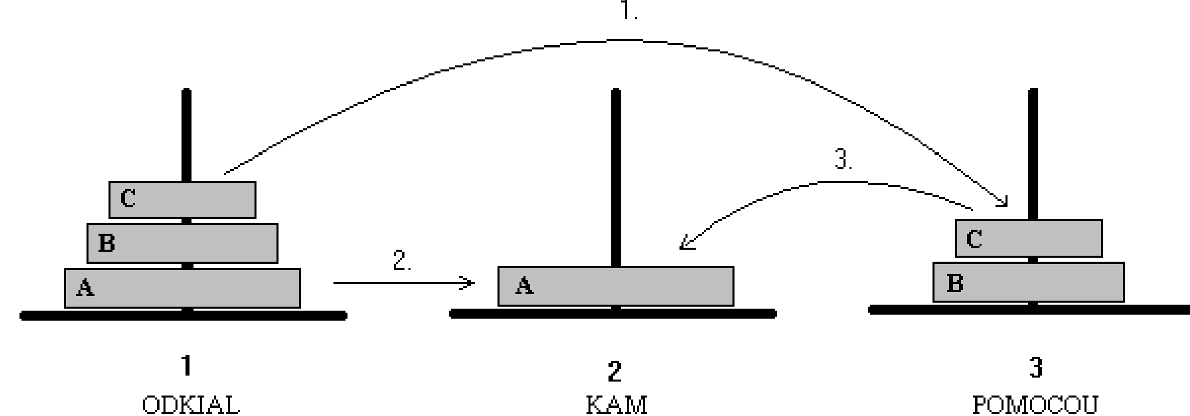
## Príklad R2

Pravdepodobne najznámejším problémom riešeným štandardne rekurziou sú tzv. Hanojské veže. Legenda hovorí, že v starobylom kláštore ukrytom v ďalekom kúte Ázie stoja tri zlaté kolíky. Na prvom z nich je nasunutých 64 diskov rôznych veľkostí, a to tak, že najväčší leží dole, nad ním leží o niečo menší, nad ním ešte menší atď. Úlohou mníchov je premiestniť túto vežu na druhý kolík, pričom musia dodržať tieto pravidlá:

- naraz sa môže premiestniť iba jeden disk
- väčší disk sa nikdy nesmie položiť na menší
- ktorýkoľvek z troch kolíkov možno použiť ako „odkladací" na dočasné umiestnenie disku

Podľa legendy svet zanikne vo chvíli, keď mnísi svoju úlohu splnia.

Obrázok sme nakreslili pre tri disky (A, B, C). Stojany sú označené 1, 2 a 3. Pre označenie kolíkov sme zvolili premenné ODKIAL, KAM a POMOCOU.




Myšlienkový postup pre N diskov je nasledovný:
Aby sme najväčší disk (v obrázku označený A) mohli presunúť z kolíka 1 na kolík 2, musíme:

1. najprv presunúť nad ním ležiace disky, t.j. vežu zloženú z N-1 diskov, na kolík 3 pomocou kolíka 2 – analogický problém, ako presunúť N diskov
2. potom môžeme disk A premiestniť z kolíka 1 na kolík 2 a
3. nakoniec presunúť vežu zo zvyšných N-1 diskov z kolíka 3 na kolík 2 pomocou kolíka 1.

```
procedure HANOJ;
var Pocet: integer;

procedure PrenesVezu( N , ODKIAL , KAM , POMOCOU: integer );

    procedure PrenesDisk( Z , Na: integer );
    begin
    Memo1.Lines.Add( IntToStr(Z) + ' -> ' + IntToStr(Na) )
    end;

begin { PrenesVezu }
if N > 0
then begin
    PrenesVezu( N-1 , ODKIAL , POMOCOU , KAM );
    PrenesDisk( ODKIAL , KAM);
    PrenesVezu( N-1 , POMOCOU , KAM , ODKIAL );
  end;
end;

BEGIN { HANOJ }
Pocet := StrToInt( InputBox( 'Vstup' , 'Počet diskov:' , '3' ) );
PrenesVezu(Pocet , 1 , 2 , 3);
END;
```

Jednoduchosť procedúry PrenesVezu prekvapí asi každého. Úspešnosť rekurzie spočíva v tom, že ak počítač nevie niečo vyhodnotiť, odloží to do zásobníka. Tak počítač odkladá PrenesVezu pre čoraz menej diskov, aktualizuje ODKIAL, KAM, POMOCOU – uvedomte si, ktoré parametre sú formálne a ktoré skutočné(!), až niet čo preniesť a pokračuje ďalším príkazom PrenesDisk atď.
Určte rekurzívnu a nerekurzívnu vetvu procedúry PrenesVezu.

### Príklad R3

Zamyslite sa nad rekurzívnym výpočtom n-faktoriálu uvedeným nižšie. Údajový typ int64 má rozsah $-2^{63}$ až $2^{63}-1$ čo je číslo 9 223 372 036 854 775 807.

```
procedure TForm1.btFaktorialClick(Sender: TObject);
var N: integer;

     function fakt(n: integer; tmp:int64):int64;
     begin
     if n = 0
     then Result:= tmp
     else Result:= fakt(n-1, n*tmp)
     end;

BEGIN
N:= StrToInt(InputBox('Rekurzia','N','5'));
if N > 20 then ShowMessage( 'Ľutujem, mimo môj rozsah!' )
else Memo1.Lines.Add(Format( '%d! = %d' , [ N , fakt(N,1) ] ));
END;
```

### Príklad R4*

K tomuto príkladu sa vráťte až po prebratí dynamického poľa a vyhľadávacích algoritmov. Jeho úlohou je rekurzívne zistiť, či sa zadaná hodnota nachádza v poli.

*Riešenie*

```
procedure TForm1.btVyskytClick(Sender: TObject);
var Hladat, Vysledok: integer;

     procedure Hladaj(Lavy, Pravy: integer; var Vysledok: integer);
     begin
     if Lavy > Pravy then Vysledok:= -1
     else if Pole[Lavy] = Hladat
        then Vysledok:= Lavy
        else Hladaj(Lavy+1, Pravy, Vysledok)
     end;

BEGIN
Hladat:= StrToInt(InputBox( 'Rekurzia' , 'Hľadať hodnotu' , '' ));
Hladaj(0, High(Pole), Vysledok);
if Vysledok = -1
then Memo1.Lines.Add( 'Nie je.' )
else Memo1.Lines.Add(Format( 'Na %d. mieste.' , [ Vysledok+1 ] ));
END;
```

### Príklad R5*

K tomuto príkladu sa vráťte až po prebratí dynamického poľa. Demonštruje obrátenie poľa, t.j. výmenu prvého prvku s posledným, druhého s predposledným,...

```
procedure TForm1.btObratClick(Sender: TObject);

        procedure Vymen(var x,y: integer);
        var Pom: integer;
        begin
        Pom:= x; x:= y; y:= Pom
        end;

        procedure Obrat(Lavy, Pravy: integer);
        begin
        if Lavy < Pravy
        then begin
                Vymen(Pole[Lavy] , Pole[Pravy]);
                Obrat(Lavy+1, Pravy-1)
             end;
        end;

BEGIN
Obrat(0, High(Pole));
END;
```

### Príklad R6

Vytvorte rekurzívnu funkciu na prevod prirodzeného čísla z desiatkovej do dvojkovej sústavy. Výsledkom nech je reťazec obsahujúci znaky čísla dvojkovej sústavy.

*Riešenie*

```
procedure TForm1.btPrevodClick(Sender: TObject);
var Cislo10: integer;

        function Prevod(Cislo10: integer): string;
        begin
        if cislo10 = 0 then Prevod:=''
        else Prevod:= Prevod(cislo10 div 2) + IntToStr(cislo10 mod 2);
        end;

BEGIN
Cislo10:= StrToInt(InputBox( 'Prevod' , 'Prirodzené číslo v 10-súst.' , '' ));
Memo1.Lines.Add(Prevod(Cislo10));
END;
```

### Príklad R7

Vytvorte rekurzívnu funkciu na výpočet kombinačného čísla $\binom{n}{k}$ ($n \geq 0$, $k \geq 0$, $n \geq k$, n, k prirodzené čísla) – koeficientu Pascalovho trojuholníka.

*Analýza*

Z matematiky vieme, že

1. pre $k = 0$ a $k = n$ sa $\binom{n}{k} = 1$ inak
2. $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$

*Riešenie*

```
procedure TForm1.btNnadKClick(Sender: TObject);
var N,K: integer;

        function NnadK(n,k:integer):integer;
        begin
        if (k = 0) or (k = n)
        then NnadK:= 1
        else NnadK:= NnadK( n-1 , k-1 ) + NnadK( n-1 , k )
        end;

BEGIN
N:= StrToInt(InputBox( 'Pascal' , 'N' , '' ));
K:= StrToInt(InputBox( 'Pascal' , 'K' , '' ));
Memo1.Lines.Add(Format( '( %d , %d ) = %d' , [ N , K , NnadK(N,K) ] ));
END;
```

### Príklad R8*

Vytvorte rekurzívnu procedúru na rozklad prirodzeného čísla na súčin prvočiniteľov.
Cislo = $c_1.c_2. \ldots .c_n$, kde $c_i$ je prvočíslo; napr. 84 = 2 . 2 . 3 . 7
*Riešenie*

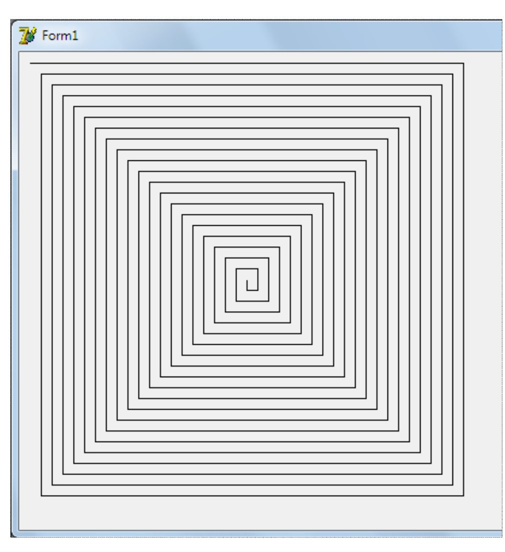
```
procedure TForm1.PocitajClick(Sender: TObject);
var Vysledok: string;

        procedure Rozklad(k: integer);
        var i: integer;
        begin
        i := 1;
        if k > 1 then repeat i := i + 1 until (k mod i) = 0;
        if k = i  then Vysledok := Vysledok + IntToStr(k)
        else begin
                Vysledok := Vysledok + IntToStr(i) + ' * ';
                Rozklad (k div i)
            end
        end;

BEGIN
Vysledok := Edit1.Text + ' = ';
Rozklad (StrToInt (Edit1.Text));
Label1.Caption := Vysledok
END;
```

Na záver dva príklady na rekurzia aj z grafiky.

### Príklad R9*

Vytvorte program, ktorý rekurzívne nakreslí, po zadaní medzery, špirálu podľa obrázka.
*Analýza*
Špirála sa skladá „zo štvorcov s posunutými stenami smerom dovnútra". Každý nový „štvorec" začína v bode, kde skončil predchádzajúci „štvorec" (pozri aj stred špirály).

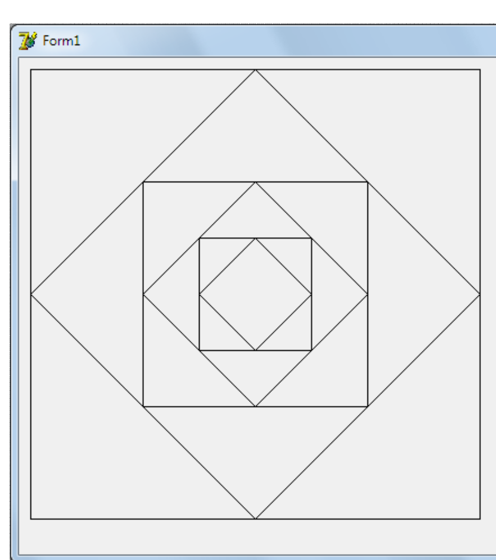
*Riešenie*




```
procedure TForm1.btSpiralaClick(Sender: TObject);
var ZacDlzka, Medzera, ZaciatokX, ZaciatokY: integer;

procedure Spirala(dlzka, x, y: integer);
begin
if dlzka>0
then begin
        with Form1.Canvas do
        begin
            LineTo(x + dlzka, y);
            LineTo(x + dlzka, y + dlzka);
            LineTo(x + medzera, y + dlzka);
            LineTo(x + medzera, y + medzera);
        end;
        Spirala(dlzka-2*medzera, x+medzera, y+medzera);
        end;
end;

BEGIN
//Zmaz Canvas;
Canvas.Brush.Color:= Form1.Color;
Canvas.FillRect(ClientRect);

Medzera:= StrToInt(InputBox( 'Špirála' , 'Medzera v špirále' , '10' ));
ZacDlzka:= 400; ZaciatokX:= 10; ZaciatokY:= 10;
Canvas.MoveTo(ZaciatokX , ZaciatokY);
Spirala(ZacDlzka , ZaciatokX , ZaciatokY);
END;
```

## Príklad R10*

Vytvorte program, ktorý rekurzívne nakreslí, po zadaní počtu „dvojštvorcov“, obrazec podľa obrázka (tu tri „dvojštvorce“).




*Riešenie*

```
procedure TForm1.btStvorceClick(Sender: TObject);
var N, ZacDlzka, ZaciatokX, ZaciatokY: integer;

            procedure Stvorce(pocet,dlzka,x,y:integer);
            begin
            if pocet>0
            then begin
                    with Canvas do
                    begin
                        LineTo(x+dlzka,y);
                        LineTo(x+dlzka,y+dlzka);
                        LineTo(x,y+dlzka);
                        LineTo(x,y+dlzka div 2);
                        LineTo(x+dlzka div 2,y+dlzka);
                        LineTo(x+dlzka,y+dlzka div 2);
```

```
                LineTo(x+dlzka div 2,y);
                LineTo(x+dlzka div 4,y+dlzka div 4);
                Stvorce(pocet-1,dlzka div 2,x+dlzka div 4,y+dlzka div 4);
                LineTo(x,y+dlzka div 2);
                LineTo(x,y);
            end;
            end;
        end;

BEGIN
//Zmaz Canvas;
Canvas.Brush.Color:= Form1.Color;
Canvas.FillRect(ClientRect);

N:= StrToInt(InputBox( 'Štvorce' , 'Počet dvojštvorcov' , '3'));
ZacDlzka:= 400; ZaciatokX:= 10; ZaciatokY:= 10;
Canvas.MoveTo(ZaciatokX , ZaciatokY);
Stvorce(N, ZacDlzka , ZaciatokX , ZaciatokY);
END;
```

Úlohy

Napíšte rekurzívnu a nerekurzívnu procedúru na výpočet Fibonacciho čísel a diskutujte o ich výpočtových zložitostiach, ak viete, že:

1. $F_0 = 0$ a $F_1 = 1$
2. $F_n = F_{n-1} + F_{n-2}$   pre $n > 1$

Čísla Fib. postupnosti: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,...

* Je daných n závaží s hmotnosťami 1kg, 3kg, 9kg,...,$3^{n-1}$ kg a predmet , ktorý máme odvážiť na dvojramenných váhach. Napíšte program, ktorý zistí, ako treba rozložiť závažia a predmet na obe ramená váh. Napríklad pri hmotnosti predmetu 29kg treba na ľavú misku položiť predmet + 1kg a vpravo 27kg + 3kg.

* Napíšte program, ktorý zistí, či možno vydláždiť trominami námestie v tvare štvorca, ktorého strana je mocninou čísla 2 (2, 4, 8, 16,...), pričom na jednom nevydláždenom štvorčeku je umiestnená socha. Tromino je dlaždica zložená z troch štvorčekov zvierajúcich 90° uhol. Vpravo príklad vydláždeného námestia.

S

## NA ZÁVER

Rekurzia je veľmi elegantnou programovacou technikou, avšak pamäťovo aj časovo omnoho náročnejšou, ako je iteračné riešenie (použitie for-cyklu, while-cyklu) toho istého problému. Vysvetlili sme ju na jednoduchých úlohách, ktoré by ste však, okrem Hanojských veží, nikdy nemali riešiť rekurziou. Rekurzia sa využíva pri riešení zložitejších problémov, ktoré sú rozobraté napríklad v knihe N.Wirtha Algoritmy a štruktúry údajov.

## Jednorozmerné pole

*Motivačný príklad*
Vytvorte program na výpočet aritmetického priemeru zo zadaných reálnych čísel.
Úloha pre nás, v tomto štádiu vedomostí, dúfam, jednoduchá ☺. Ak pre niekoho nie, nech si pozrie príklady 3.1.9 a 3.3.4 v prvom diele zbierky. Prestáva byť však jednoduchou, ak zadanie doplníme o požiadavku - výstup nech obsahuje aj zadané čísla, napríklad Aritmetický priemer z čísel... je...
Túto požiadavku, bez nového údajového typu, ktorý je schopný si zapamätať ľubovoľný konečný počet hodnôt, nevieme splniť. (K riešeniu príkladu sa vrátime neskôr.)
Označme vkladané hodnoty ako $h_1, h_2, ..., h_i, ..., h_n$ zrejme i nadobúda hodnoty 1, 2, ..., n
V matematike by sme hovorili o konečnej postupnosti n hodnôt; v programovaní môžeme tieto hodnoty uložiť do poľa s názvom h (presnejšie do premennej h typu pole) pričom $h_1, h_2, ..., h_n$ sú prvky poľa h. Každý prvok je v poli jednoznačne určený svojim indexom. Prvok s indexom 1 sa zapisuje h[1], s indexom 2 h[2], s indexom i h[i], s indexom n h[n]. Rozlišujte medzi indexom prvku poľa a hodnotou prvku poľa! Index prvku udáva pozíciu prvku - hodnoty v poli a je uvedený v hranatých zátvorkách, hodnota prvku je to, čo sme chceli v poli zapamätať na príslušnom mieste. Napríklad zápis Pole[i] predstavuje hodnotu, ktorá je uložená v poli s názvom Pole na mieste s indexom i. Index nemusí byť len premenná alebo hodnota, môže to byť aj výraz.
Ak index poľa začína 1, je súčasne aj poradovým číslom prvku v poli. Môžeme povedať, že zápis Pole[i] predstavuje i-ty prvok poľa (napríklad Pole[3] znamená tretí prvok poľa). Potom prvok Pole[n-1] je predposledným prvkom poľa (ak n je počet prvkov poľa).
Ak však pole začína indexom 0, zápis Pole[i] predstavuje (i+1). prvok poľa a prvok Pole[n-1] je posledným prvkom poľa! Preto je lepšie hovoriť, že Pole[i] je prvok s indexom i ako že je to i-ty prvok poľa.

Pole (Array) je štruktúrovaný údajový typ, ktorý sa skladá z prvkov rovnakého typu, pričom k prvku pristupujeme cez index – pozíciu daného prvku v poli.
**Pole** môže byť
**dynamické** – počet jeho prvkov (veľkosť poľa), sa môže meniť počas behu programu a nemusí byť známy pri spustení programu; známe musí byť, akého typu budú prvky poľa,
**statické** – jeho veľkosť a typ prvkov sú uvedené v úseku definícií a deklarácií, t.j. sú nastavené pred spustením programu, a príslušná pamäť sa vyhradí hneď po spustení programu,
**konštantné** – jeho veľkosť aj hodnoty sú uvedené v úseku definícií a deklarácií a nemôžu sa meniť počas behu programu.
Najuniverzálnejšie je dynamické pole, preto sa s ním budeme zaoberať najviac, statické a konštantné polia majú špecifické použitie a na tie si uvedieme príklady na záver kapitoly.

## Dynamické pole

Jeho definícia alebo deklarácia má tvar:

... array of *TypPrvkovPola*;

```
Pr.: type
        tPole = array of integer;     // definovaný údajový typ pole s vlastným názvom tPole
     var
        Pole: tPole;                  // deklarovaná premenná Pole typu tPole
        A: array of real;             // deklarovaná premenná A typu pole bez názvu typu poľa
```

Takto definované alebo deklarované pole nemá žiadnu veľkosť a nemá v pamäti vyhradený priestor pre zapamätanie svojich prvkov.
Pred prvým použitím premennej typu dynamické pole je potrebné nastaviť veľkosť poľa príkazom **SetLength**, čím sa alokuje (vyhradí) potrebná veľkosť z voľnej pamäte.

```
Pr.:    SetLength (Pole, 100);        //V pamäti sa vyhradí miesto pre 100 čísel typu integer
```

SetLength (A, Pocet); //V pamäti sa rezervuje miesto pre Pocet čísel typu real

**Indexy dynamických polí začínajú vždy od 0 (nuly)!** Po prekročení rozsahu indexov sa nemusí zobraziť chybové hlásenie! Odporúča sa používať štandardné funkcie **Low, High** a **SizeOf**.

Funkcia **Low** vracia najnižší index poľa, pri dynamických poliach vždy 0.
Funkcia **High** vracia najvyšší index poľa, pri dynamických poliach: veľkosť poľa – 1.
Pr.: for i:= Low(A) to High(A) do A[i]:= i; //analogicky: for i:= 0 to Pocet-1 do A[i]:= i;
Funkcia **SizeOf** vracia hodnotu: veľkosť poľa * veľkosť prvku poľa.
Napríklad v 32-bitových Delphi je hodnota typu integer uložená v 4B, real v 6B.

Zväčšiť veľkosť dynamického poľa funkciou SetLength možno počas behu programu viackrát. „Vyrezať“ časť prvkov poľa do nového poľa možno funkciou **Copy**, ktorá má tvar:
*PremennaTypuPole* := Copy ( *PremennaTypuPole*, *PociatocnyIndex*, *PocetPrvkovPola* )
Pr.: A:= copy(A,0,5); //V poli A zostane 5 prvkov od pozície 0
Dynamické pole sa zruší priradením hodnoty **nil**.
Pr.: A:= nil; //Uvoľní sa alokovaná pamäť

*Pamätaj*!
Po priradení B:= A; kde A, B sú dynamické polia, každá zmena v jednom poli sa prejaví aj v poli druhom! Nevytvorí sa totiž kópia poľa A (ako pri statických poliach), ale do poľa B sa uloží len ukazovateľ na pole A.

## Príklad 9.1

Vytvorte program na zapamätanie vopred neznámeho počtu celých čísel (počet čísel zadá užívateľ po spustení programu) a na vypísanie uložených čísel.
*Analýza*
Po preklade do „programátorského jazyka“ úloha znie: Napíšte podprogram, ktorý umožní užívateľovi, po zadaní počtu prvkov poľa, uložiť celočíselné hodnoty do jednorozmerného poľa. Keďže veľkosť - počet prvkov poľa, bude známy až po spustení programu, použijeme dynamické pole. Aby sme sa mohli presvedčiť o úspešnosti riešenia, dáme hodnoty vypísať. Keďže už poznáme podprogramy, vytvorenie dynamického poľa zrealizujeme v procedúrach.

1. V celom programe použijeme globálnu premennú Pole, deklarovanú v časti implementation zápisom var Pole: array of integer;
2. Všetky vytvorené procedúry „zviažeme“ s formulárom Form1, t.j. ich hlavičky uvedieme aj v definícii typu TForm1 v časti interface
3. Vytvoríme dve procedúry na naplnenie dynamického poľa hodnotami
   a. v procedúre VytvorPoleNahodne sa hodnoty poľa vygenerujú ako náhodné celé čísla od 0 po 99; riešenie:
   ```
   procedure TForm1.VytvorPoleNahodne;
   var i: integer;
   begin
   SetLength( Pole , StrToInt( InputBox( 'Vytvor' , 'Počet prvkov poľa' , '20' )));
   for i:=0 to High(Pole) do Pole[i]:= random(100);
   end;
   ```
   Pred prvým použitím funkcie random treba použiť príkaz randomize, umiestnime ho do inicializačnej časti programu zápisom
   ```
   initialization
   randomize;
   end.
   ```

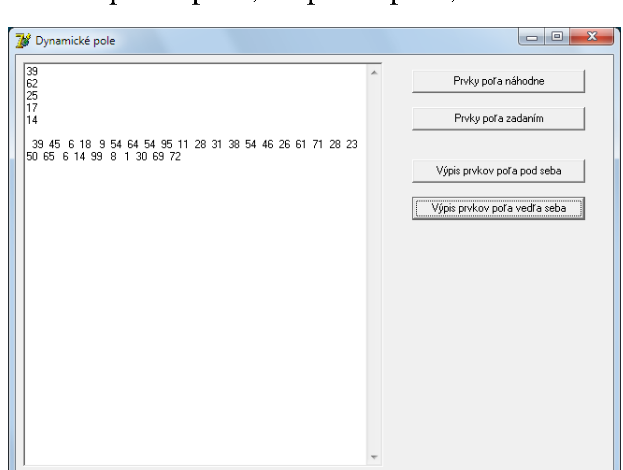
b. v procedúre VytvorPoleZadanim hodnoty poľa zadá užívateľ; riešenie:

```
procedure TForm1.VytvorPoleZadanim;
var i: integer;
begin
SetLength( Pole, StrToInt( InputBox( 'Vytvor' , 'Počet prvkov poľa' , '5' )));
for i:=0 to High(Pole) do Pole[i]:= StrToInt(InputBox( 'Vytvor' , 'Prvok poľa' , '' ));
end;
```

Funkciu InputBox upravte tak, aby sa vypisovalo: 1. prvok poľa, 2. prvok poľa, atď.

Obe procedúry volajte cez vhodne pomenované tlačidlá, napríklad

```
procedure TForm1.btVytvorNahodneClick
(Sender:TObject);
begin
VytvorPoleNahodne;
end;

procedure TForm1.btVytvorZadanimClick
(Sender:TObject);
begin
VytvorPoleZadanim;
end;
```




4. Vytvoríme dve procedúry na vypísanie prvkov dynamického poľa
   a. procedúra VypisPole čo najjednoduchšie, t.j. pod seba vypíše hodnoty poľa

```
procedure TForm1.VypisPole;
var i: integer;
begin
Memo1.Clear;
for i:=0 to High(Pole) do Memo1.Lines.Add( IntToStr( Pole[i] ) );
end;
```

   b. procedúra VypisPoleVRiadku zapíše hodnoty do premennej Riadok typu string a zalomenie textu v riadku zrealizuje komponent Memo podľa jeho šírky

```
procedure TForm1.VypisPoleVRiadku;
var i: integer; Riadok: string;
begin
Memo1.Clear;
Riadok:= '';
for i:=0 to High(Pole) do Riadok:= Riadok + Format( '%4d' , [ Pole[i] ] );
Memo1.Lines.Add(Riadok);
end;
```

## Funkcia Format

Funkcia Format umožňuje formátovať výstupný reťazec. Obsahuje formátovací reťazec (v apostrofoch), ktorý určuje, ako má vyzerať výsledok na obrazovke, a pole hodnôt, ktoré majú byť formátované (v hranatých zátvorkách).
Formátovací reťazec je zložený z formátovacích príkazov a zo sprievodných textov (literálov). Formátovacie príkazy začínajú znakom % a vo výpise sú nahradené príslušnou hodnotou z poľa hodnôt. Formátovacie príkazy určujú, ako budú hodnoty zobrazené. Začínajú znakom % a končia písmenom udávajúcim typ argumentu (napr. d – celé číslo, f – reálne číslo, s – reťazec atď.).

Formátovací príkaz môže obsahovať

- pre typ integer
  - %d zobrazí celé číslo bez úpravy
  - %*pzv*d napr. %10d *pzv* znamená počet znakov výstupu; na výpis hodnoty použije *pzv* znakov, ak celé číslo obsahuje menej znakov, zľava doplní medzerami; ak celé číslo obsahuje viacej znakov, *pzv* ignoruje – použije príkaz %d
  - %-*pzv*d napr. %-10d na výpis hodnoty použije *pzv* znakov, ak celé číslo obsahuje menej znakov, sprava doplní medzerami – prakticky zarovná výstupnú hodnotu vľavo v zadanom počte znakov výstupu; ak celé číslo obsahuje viacej znakov, *pzv* ignoruje – použije príkaz %d
- pre typ real
  - %f zobrazí reálne číslo zaokrúhlené na stotiny
  - %*pzv*f *pzv* znamená počet znakov výstupu; zobrazí reálne číslo zaokrúhlené na stotiny, ak číslo obsahuje menej znakov, zľava doplní medzerami; ak číslo obsahuje viacej znakov, hodnotu *pzv* ignoruje
  - %-*pzv*f zobrazí reálne číslo zaokrúhlené na stotiny, ak číslo obsahuje menej znakov, sprava doplní medzerami – prakticky zarovná výstupnú hodnotu vľavo v zadanom počte znakov výstupu; ak číslo obsahuje viacej znakov, hodnotu *pzv* ignoruje
  - %*pzv.pdm*f *pdm* znamená počet desatinných miest; výstupnú hodnotu zobrazí so zadaným počtom desatinných miest; ak číslo obsahuje viacej desatinných miest, zaokrúhli ho na zadaný počet desatinných miest; ak číslo obsahuje menej desatinných miest, doplní výpis nulami do zadaného počtu desatinných miest
  - %-*pzv.pdm*f ako predchádzajúce, len výstupná hodnota bude zarovnaná vľavo (ak *pzv* je väčšie ako počet znakov výstupnej hodnoty)
- pre typ string
  - %s zobrazí reťazec bez úpravy
  - %*pzv*s *pzv* znamená počet znakov výstupu; na výpis hodnoty použije *pzv* znakov, ak reťazec obsahuje menej znakov, zľava doplní medzerami; ak reťazec obsahuje viacej znakov, *pzv* ignoruje – použije príkaz %s
  - %*pzv.pzr*s *pzr* udáva počet znakov reťazca; ak je hodnota *pzr* menšia ako dĺžka reťazca, reťazec bude sprava skrátený na zadaný počet *pzr* znakov; ak je hodnota *pzr* väčšia ako dĺžka reťazca, *pzr* bude ignorované
  - %- *pzv.pzr*s ako predchádzajúce, len výstupná hodnota bude zarovnaná vľavo (ak *pzv* je väčšie ako počet znakov výstupnej hodnoty)

**Poznámka**
Ostatné tvary funkcie Format si môžete pozrieť v helpe.
Funkciu Format je výhodné používať aj preto, že vracia typ string a teda pre číselné hodnoty nemusíme už použiť konverzné funkcie.
Ak chceme počet znakov (*pzv*, *pdm* alebo *pzr*) zadať cez premennú prípadne výraz, vo formátovacom príkaze použijeme *.

Napríklad
presnost:= StrToInt( InputBox( 'Výsledok' , 'Počet desatinných miest' , '2' ));
Memo1.Lines.Add( Format( '%0.*f ', [ presnost , realne ] ));
alebo Format ('%*.*f' , [20, 3, 123.456789]) je to isté ako Format('%20.3f' , [123.456789])
alebo Format ('%1.0f' , [123.456789]) zobrazí reálne číslo bez desatinnej časti, t.j. ako celé číslo (použije zaokrúhlenie hodnoty).

## Príklad 9.2

Príklad 9.1 doplňte o podprogram na nájdenie najväčšieho prvku poľa.
*Analýza*
Nájsť najväčší prvok poľa (maximum) znamená nájsť prvok, pre ktorý platí, že je väčší alebo rovný ako ktorýkoľvek iný prvok poľa. Na začiatku hľadania je najväčším prvkom poľa prvý prvok. Bolo by chybou za najväčší prvok považovať na začiatku hľadania napríklad nulu (ak sú všetky prvky poľa záporné čísla, dostaneme zlý výsledok). Potom sa pozrieme na druhý prvok a zaujíma nás, či je väčší ako dovtedy nájdený najväčší prvok. Ak áno, máme nové maximum. Postupne prechádzame poľom a porovnávame jeho hodnoty s ostatným maximom. Na konci poľa musí byť v premennej určenej pre maximum najväčšia hodnota poľa.
Keďže výsledkom podprogramu je jedna hodnota jednoduchého typu alebo typu string, môže riešenie zrealizovať funkciou. Kvôli pamäťovej efektívnosti sme použili premennú Result.

```
function Maximum: integer;
var i: integer;
begin
Result:= Pole[0];
for i:= 1 to High(Pole) do if Pole[i] > Result then Result:= Pole[i];
end;
```

## Príklad 9.3

Príklad 9.1 doplňte o podprogram, ktorý nájde najväčší prvok a vymení ho s posledným v poli.
*Analýza*
Na to, aby sme mohli vymeniť najväčší prvok v poli s posledným prvkom, musíme poznať hodnotu maxima aj jeho index v poli. Ak sa zamyslíme, zistíme, že keď poznáme index maxima (označme iMax), k hodnote maxima sa už ľahko dostaneme (Pole[iMax]). Preto stačí predchádzajúcu funkciu upraviť tak, aby vracala index maxima v poli.

```
function IndexMaxima: integer;
var i, iMax: integer;
begin
iMax:= 0;                    // na začiatku je najväčším prvý prvok poľa a ten má index 0
for i:= 1 to High(Pole) do if Pole[i] > Pole[iMax] then iMax:= i;
Result:= iMax;
end;
```

Použitie funkcie IndexMaxima na výmenu s posledným prvkom poľa. Kvôli efektívnosti sme funkčný výraz IndexMaxima použili len raz.

```
procedure TForm1.btMaxNaKonciClick(Sender: TObject);
var Pom, iMax: integer;
begin
iMax:= IndexMaxima;
Pom:= Pole[iMax]; Pole[iMax]:= Pole[High(Pole)]; Pole[High(Pole)]:= Pom;
end;
```

### Príklad 9.4

Príklad 9.1 doplňte o podprogram na zistenie počtu výskytov zadanej hodnoty v poli.
*Analýza*
Na to, aby sme zistili počet výskytov zadanej hodnoty v poli, potrebujeme poznať hľadanú hodnotu. Tú môžeme doviesť ako parameter funkcie. Na začiatku prehľadávania poľa je počet výskytov zadanej hodnoty zrejme nula. Postupne, od prvého po posledný prvok, prechádzame poľom, a vždy, keď narazíme na hľadanú hodnotu, zvýšime počet jej výskytov o jeden.

```
function Pocet( HladHod: integer ): integer;
var i: integer;
begin
Result:= 0;
for i:= 0 to High(Pole) do if Pole[i] = HladHod then inc(Result);
end;
```

Použitie funkcie Pocet

```
procedure TForm1.btPocetClick(Sender: TObject);
var Hladat: integer;
begin
Hladat:= StrToInt( InputBox( 'Počet výskytov' , 'Hľadať hodnotu' , '' ));
Memo1.Lines.Add( 'Hodnota sa v poli vyskytuje ' + IntToStr( Pocet( Hladat ) ) + '-krát' );
end;
```

Zrejme, ak sa hodnota v poli nevyskytuje ani raz, vyvezie sa nula. Zmeňte poslednú procedúru tak, aby, ak sa hodnota v poli nevyskytuje, vypísala „Hodnota sa v poli nevyskytuje“.

### Príklad 9.5

Príklad 9.1 doplňte o podprogram na nájdenie najmenšieho prvku a počtu jeho výskytov v poli.
*Analýza*
Nájsť najmenší prvok v poli by už nemal byť problém, stačí obrátiť znak nerovnosti v príklade 9.2. Po nájdení minima by sme mohli použiť funkciu Pocet z príkladu 9.4. My však chceme efektívnejšie riešenie, vyriešiť príklad jedným prechodom poľom.
Zaujímajú nás teda všetky hodnoty, ktoré sú menšie ale aj rovné ostatnému minimu. Ak je hodnota poľa rovná aktuálnemu minimu, treba počet jeho výskytov v poli zvýšiť o jeden. Ak je hodnota poľa menšia ako ostatné minimum, našli sme nové minimum, treba ho zapamätať a jeho počet nastaviť na 1.

```
procedure MinPocet(var Min, Pocet: integer);
var i: integer;
begin
Min:= Pole[0]; Pocet:= 1;
for i:= 1 to High(Pole) do
  if Pole[i] <= Min then if Pole[i] = Min then inc(Pocet)
                         else begin Min:= Pole[i]; Pocet:= 1; end;
end;
```

Použitie procedúry MinPocet

```
procedure TForm1.btMinPocetClick(Sender: TObject);
var Najmensi, Kolko: integer;
begin
MinPocet( Najmensi , Kolko );
```

```
Memo1.Lines.Add( Format( 'Najmenší prvok má hodnotu %d a vyskytuje sa v poli %d-krát' , [ Najmensi, 
Kolko ] ));
end;
```

### Príklad 9.6

Vytvorte podprogram, ktorý nájde najmenší a najväčší prvok poľa, najmenší vymení s prvým prvkom v poli a najväčší s posledným prvkom.

*Analýza*

Ako vyplýva z príkladu 9.3, stačí nájsť miesto výskytu najmenšieho prvku a najväčšieho prvku v poli, výmena je už snáď klasickým problémom. Počiatočná hodnota iMin je nula a počiatočná hodnota iMax je tiež nula (nie napríklad High(Pole)), je zrejmé prečo?

```
procedure VymenMinMax;
var iMin, iMAX, i, Pom: integer;
begin
iMin:= 0; iMax:= 0;
for i:= 1 to High(Pole) do
begin
        if Pole[i] < Pole[iMin] then iMin:= i;
        if Pole[i] > Pole[iMax] then iMax:= i;
end;
Pom:= Pole[iMin]; Pole[iMin]:= Pole[0]; Pole[0]:= Pom;
if iMax=0 then iMax:= iMin;
Pom:=Pole[iMax]; Pole[iMax]:= Pole[High(Pole)]; Pole[High(Pole)]:= Pom;
end;
```

Požadovaná „dvojvýmena“ môže byť zradná. Vynechaním príkazu if iMax=0 then iMax:= iMin; procedúra obsahuje logickú chybu - ak je v poli maximum na prvom mieste (index nula), po prvej výmene už bude na mieste... (domyslite sami:-).

Predchádzajúce úlohy si vyžadovali pozrieť všetky prvky poľa, čo vlastne znamenalo použiť príkaz for. Mnoho úloh je však formulovaných v praxi tak, že napríklad stačí zistiť, či sa hodnota v poli vyskytuje a teda prehľadávanie poľa môžeme skončiť hneď, ako prvok s požadovanou hodnotou nájdeme. Vyhľadávanie v skupine údajov je veľmi častou činnosťou na počítačoch, pričom sa rozlišuje vyhľadávanie v neutriedenej a vyhľadávanie v utriedenej skupine údajov.

Na záver trochu iný príklad.

### Príklad 9.7

Je vygenerované pole celých čísel. Napíšte procedúru, ktorá zistí najviac koľkokrát sa opakujú za sebou idúce rovnaké čísla v poli.

*Analýza*

Napríklad v poli 5 2 7 7 4 9 9 9 8 1 3
sa najviac trikrát opakujú rovnaké čísla idúce v postupnosti za sebou.

Treba prechádzať poľom a testovať, či Pole[i+1] = Pole[i]. Ak áno, počet opakujúcich sa čísel (v procedúre premenná Dlzka) treba zvýšiť o jeden, ak nie, začína nová skupina čísel a predbežne je počet zopakovaných čísel 1. Ak začína nová skupina čísel, treba zistiť, či ostatný počet opakujúcich sa čísel nie je väčší, ako dovtedy najväčší počet. Pozor, aby tento test bol vykonaný aj po ukončení prehľadávania poľa. Keďže výsledkom procedúry je jedna hodnota typu integer, môžeme problém riešiť funkciou.

```
function MaxDlzka: integer;
var i, Dlzka: integer;
begin
Result:= 0;
Dlzka:= 1;
for i:= 0 to High(Pole)-1 do
if Pole[i] = Pole[i+1]
then begin
        inc(Dlzka);
        if Dlzka > Result then Result:= Dlzka;                  //neustále testuje
        end
else Dlzka:=1;
end;

procedure TForm1.btNajdiClick(Sender: TObject);
begin
VytvorPole;
VypisPole;
Memo1.Lines.Add(Format('V postupnosti sa vyskytujú najviac %d-krát rovnaké čísla za sebou',
[MaxDlzka]));
end;
```

Pozor, ak trochu pozmeníme poradie príkazov vo funkcii, dokonca k logickejšiemu poradiu, nemusí to byť správne:

```
...
for i:= 1 to High(Pole)-1 do
          if Pole[i] = Pole[i+1]
          then inc(Dlzka);
          else begin
                    if Dlzka > Result then Result:= Dlzka;              //testuje, len keď začína nový úsek
                    Dlzka:= 1;
               end;
end;
```

## Lineárne vyhľadávanie

Vyhľadávanie je činnosť s cieľom zistiť, či sa prvok so zadanou vlastnosťou nachádza vo zvolenej skupine údajov.
Sústredíme sa na vyhľadávanie v jednorozmernom poli obsahujúcom celé čísla (typ integer).
Môžeme zisťovať:

- či sa prvok s požadovanou vlastnosťou v poli nachádza
- počet jeho výskytov
- miesta výskytov (prvé, posledné, všetky miesta)
- ak sa nenachádza, miesto, kde ho treba vložiť, aby si pole zachovalo pôvodné vlastnosti
- atď.

**Dva základné spôsoby vyhľadávania:**

- **lineárne** (sekvenčné), keď skúmame prvok za prvkom; používame ho pri vyhľadávaní v neutriedenom poli
- **binárne**, keď podľa hodnoty prvku v strede skúmanej oblasti pokračujeme v prehľadávaní buď dolnej alebo hornej časti; možno použiť len pri utriedenom poli.

**Typické procedúry lineárneho vyhľadávania**

### Príklad 10.1

Vytvorte podprogram na zistenie, či sa zadaná hodnota nachádza v neutriedenom poli.

```
function LinVyskyt (HlHodnota: integer): boolean;
var i: integer;
begin
i := 0;
while (Pole[i] <> HlHodnota) and (i < High(Pole)) do i := i + 1;
LinVyskyt:= Pole[i] = HlHodnota;
end;
```

Ak ste príkaz `LinVyskyt:= Pole[i] = HlHodnota;` napísali pomocou príkazu `if Pole[i] = HlHodnota then LinVyskyt:= True else LinVyskyt:= False;` je to tiež v poriadku. Ak ste však použili `if i < High(Pole) then LinVyskyt:= True else LinVyskyt:= False;` neurobili ste dobre (hľadaná hodnota môže byť aj na poslednom mieste v poli). Preto, pokiaľ to je možné, volíme pre test tú z dvoch podmienok ukončenie cyklu, ktorá sleduje zadanie úlohy, teda nie, či sme alebo nie sme na konci poľa, ale či sa posledná hodnota Pole[i] rovná hľadanej hodnote.

Príklad použitia funkcie LinVyskyt

```
procedure TForm1.btLinVyskytClick(Sender: TObject);
var Hladat: integer;
begin
Hladat:= StrToInt(InputBox('LinVyskyt','Hľadať hodnotu',''));
if LinVyskyt(Hladat)
then Memo1.Lines.Add(IntToStr(Hladat) + ' sa v poli vyskytuje')
else Memo1.Lines.Add(IntToStr(Hladat) + ' sa v poli nevyskytuje');
end;
```

## Príklad 10.2

Vytvorte podprogram na zistene miesta prvého výskytu prvku so zadanou hodnotou v neutriedenom poli.

*Analýza*

Treba si uvedomiť, že v predchádzajúcej funkcii LinVyskyt sa cyklus - prehľadávanie poľa, zastaví na hľadanej hodnote alebo na konci poľa. Ak sa teda hodnota v poli vyskytuje, posledná hodnota i udáva index prvého výskytu hľadanej hodnoty v poli. Preto stačí vyviesť hodnotu i. Na hodnotu i sa však nemá zmysel pýtať, ak sa hľadaný prvok v poli nevyskytuje, preto ponecháme pôvodný tvar funkcie LinVyskyt.

```
function LinMiesto (HlHodnota: integer; var i: integer): boolean;
begin
i := 0;
while (Pole[i] <> HlHodnota) and (i < High(Pole)) do inc(i);
LinMiesto:= Pole[i] = HlHodnota;
end;
```

Príklad použitia funkcie LinMiesto

```
procedure TForm1.btLinMiestoClick(Sender: TObject);
var Hladat, PrvyVyskyt: integer;
begin
Hladat:= StrToInt(InputBox('LinMiesto','Hľadať hodnotu',''));
if LinMiesto(Hladat, PrvyVyskyt)
then Memo1.Lines.Add('Prvý výskyt hladanej hodnoty má index ' + IntToStr(PrvyVyskyt))
else Memo1.Lines.Add(IntToStr(Hladat) + ' sa v poli nevyskytuje');
end;
```

Funkciu LinMiesto pozmeňte tak, aby našla posledný výskyt hľadanej hodnoty v poli.

## Príklad 10.3

Vytvorte podprogram na zistenie, či sa zadaná hodnota nachádza v neutriedenom poli, a ak nie, nech ju zaradí za posledný prvok poľa.

```
procedure LinVlozit(HlHodnota: integer);
var i: integer;
begin
i := 0;
while (Pole[i] <> HlHodnota) and (i < High(Pole)) do inc(i);
if Pole[i] <> HlHodnota
then begin
        SetLength(Pole, Length(Pole)+1);
        Pole[High(Pole)]:= HlHodnota;
     end
end;
```

Príklad použitia procedúry LinVlozit

```
procedure TForm1.btLinVlozitClick(Sender: TObject);
var Hladat: integer;
begin
Hladat:= StrToInt(InputBox('LinVyskyt','Hľadať hodnotu',''));
LinVlozit(Hladat);
end;
```

## Príklad 10.4

Vytvorte podprogram na zistenie, či sa zadaná hodnota nachádza v neutriedenom poli, a ak nie, nech ju vloží na začiatok poľa (pred prvý prvok poľa, na index nula).

*Analýza*

V tomto príklade si precvičíme posúvanie prvkov poľa, čo sa nám neskôr zíde. Ak sa zadaná hodnota v poli nenachádza, aby sme ju mohli vložiť na začiatok poľa, musíme ostatné hodnoty poľa posunúť o jednu pozíciu doprava, t.j. prvok s indexom i na index i+1. Posledná hodnota poľa pôjde zrejme na pozíciu High(Pole)+1. Keďže sa pole zväčší, musíme jeho veľkosť zmeniť príkazom setlength. Posúvať prvky musíme od konca, keďže príkaz Pole[i+1]:= Pole[i] prepíše pôvodnú i plus prvú hodnotu poľa. Počet posunutí je známy, preto použijeme príkaz for.

```
procedure LinVlozitZaciatok(HlHodnota: integer);
var i: integer;
begin
i := 0;
while (Pole[i] <> HlHodnota) and (i < High(Pole)) do inc(i);
if Pole[i] <> HlHodnota
then begin
        SetLength(Pole, Length(Pole)+1);
        for i:= High(Pole)-1 downto 0 do Pole[i+1]:= Pole[i];
        Pole[0]:= HlHodnota;
      end;
end;
```

For-cyklus upravte tak, aby začínal for i:= High(Pole) downto...

Doteraz sme v kapitole Lineárne vyhľadávanie pracovali len s celočíselným poľom. Teraz použijeme pole reálnych čísel a v časti Ďalšie úlohy na dynamické pole budeme pracovať aj s poliami obsahujúcimi hodnoty typu boolean alebo string.

Aby sme s poľom reálnych čísel mohli pracovať, musíme reálne čísla buď zadať do poľa alebo ich nechať vygenerovať počítačom. Prvá možnosť je triviálna, stačí v Príklade 9.1 zmeniť v deklarácii poľa Pole typ prvkov poľa z integer na real (var Pole: array of real;) a v bod 3.b procedúru prerobiť na vstup reálnych čísel, t.j. použiť konverziu StrToFloat namiesto StrToInt. Úloha, aby pole obsahovalo náhodne vygenerované reálne čísla, je už zaujímavejšia a potrebujeme vedieť, že funkcia random vracia náhodné reálne číslo z intervalu <0 , 1). Pripomíname: funkcia random(celé_číslo) vracia náhodné celé číslo z rozsahu <0 , celé_číslo), t.j. z množiny { 0, 1 , 2,..., celé_číslo-1}.

## Príklad 10.5

Vytvorte podprogram, ktorý po zadaní počtu prvkov poľa vytvorí pole obsahujúce náhodne vybrané reálne čísla z intervalu <0, 10).

*Analýza*

Prvý výraz, ktorý nám napadne, na vygenerovanie reálnych čísel z intervalu <0, 10) by mohol byť random(10)+random. Trocha „zložitejší“ je výraz random*10. Takže riešením je napríklad

```
procedure TForm1.VytvorPoleNahodne;
var i: integer;
begin
setlength (Pole , StrToInt ( InputBox ( 'Vytvor' , 'Počet prvkov poľa' , '10' ) ) );
for i:= 0 to High(Pole) do Pole[i]:= random*10;                    //pozri študijný text nižšie
end;
```

**Ak budete aplikovať procedúry lineárneho vyhľadávania na pole reálnych čísel, s najväčšou pravdepodobnosťou hodnotu poľa, ktorú uvidíte pri výpise prvkov poľa, nebude vedieť počítač nájsť!** Toto je bežná „chyba" spracovania reálnych čísel počítačom a myslite na ňu aj v iných aplikáciách (nižšie ukážka z programu Excel).

B2 fx 0,11111111119

| | A | B | C |
|---|---|---|---|
| 1 | Výpočet v stĺpci B | Zobrazí | Skutočná hodnota v pamäti |
| 2 | hodnota | 0,1111111112 | 0,11111111119 |
| 3 | 2*B2 | 0,222222222 | 0,22222222238 |
| 4 | 2*B3 | 0,444444445 | 0,44444444476 |
| 5 | 2*B4 | 0,88888889 | 0,88888888952 |
| 6 | 2*B5 | 1,777777779 | 1,77777777904 |

Preto v ostatnej procedúre odporúčame použiť napríklad výraz trunc(random*1e14)/1e13, ktorého výsledkom je také dlhé náhodné reálne číslo z intervalu <0 , 10), že zobrazené číslo zodpovedá číslu uloženému v pamäti počítača. Zápis 1eXX zodpovedá v matematike zápisu $1.10^{XX}$ (symbol e znamená exponent a číta sa „desať na"), funkcia trunc zaokrúhľuje reálne číslo smerom nadol. Pre istotu celá procedúra ešte raz.

```
procedure TForm1.VytvorPoleNahodne;
var i: integer;
begin
setlength (Pole , StrToInt ( InputBox ( 'Vytvor' , 'Počet prvkov poľa' , '10' ) ) );
for i:= 0 to High(Pole) do Pole[i]:= trunc (random*1e14)/1e13;
end;
```

Pred prvým použitím funkcie random nezabudnite použiť príkaz randomize, umiestnite ho do inicializačnej časti programu zápisom

```
initialization
randomize;
end.
```

Najmä pri statických poliach sa používa **nárazník**. Trochu umelo, ale predsa úspešne sme ho použili aj pri dynamickom poli. Odporúčame ho použiť, len ak je pravdepodobnosť, že sa prvok v poli nenachádza, vysoká.

Pred prehľadávaním poľa sa na jeho koniec vloží hľadaná hodnota s úlohou nárazníka – zastaví sa na ňom prehľadávanie poľa, ak sa v poli hľadaná hodnota pôvodne nenachádzala. Pokúste sa pochopiť, ako funguje procedúra LinNaraznik.

```
procedure LinNaraznik (HlHodnota: integer);
var i: integer;
begin
SetLength( Pole, Length(Pole)+1 ); Pole[ High(Pole) ]:= HlHodnota;    //vloženie nárazníka
i := 0;
while Pole[i] <> HlHodnota do inc(i);                          //stačí jedna podmienka
if i < High(Pole) then Pole:= copy( Pole, 0, Length(Pole)-1 );   //ak sa hodnota v pôvodnom poli
end;                                                           //vyskytovala, musíme nárazník odstrániť
```

## Binárne vyhľadávanie

Vyhľadávanie v utriedenom poli možno, oproti lineárnemu vyhľadávaniu, zefektívniť (zrýchliť). Keďže pole je utriedené, ak porovnáme hľadanú hodnotu s hodnotou v strede poľa, hneď môžeme vylúčiť polovicu prvkov poľa z prehľadávania (pole sme rozdelili na dve časti, preto názov binárne). Môžu nastať tri prípady. Po prvé, hodnota v strede poľa sa rovná hľadanej hodnote, môžeme skončiť. Po druhé, ak je hodnota v strede poľa väčšia ako hľadaná hodnota, hľadaná hodnota, ak je vôbec v poli, sa musí vyskytovať v dolnej polovici poľa. Po tretie, ak je hodnota v strede poľa menšia ako hľadaná hodnota, hľadaná hodnota, ak je vôbec v poli, sa musí vyskytovať v hornej polovici poľa a následne stačí tú prehľadať. Opísanú myšlienku delenia poľa môžeme uplatniť aj na dolnú alebo hornú polovicu poľa, následne štvrtinu, osminu,... Delenie úsekov končí nájdením hľadanej hodnoty alebo zistením, že úsek na prehľadávanie „má nulovú veľkosť".
Algoritmus popíšeme ešte raz pri procedúre BinVlozit príklad 11.3.

### Príklad 11.1

Skôr, ako sa pustíme do procedúr binárneho vyhľadávania, potrebujeme na testovanie utriedené pole. Ak nechceme sami zadávať hodnoty neklesajúcej postupnosti (predstavujú utriedené pole), môže nám ich vygenerovať aj počítač.

```
procedure TForm1.VytvorPoleNahodne;
var i: integer;
begin
setlength( Pole, StrToInt( InputBox( 'Vytvor' , 'Počet prvkov poľa' , '20' ) ) );
Pole[0]:= random(3);
for i:=1 to High(Pole) do Pole[i]:= Pole[i-1]+random(10);
end;
```

Ako musíme upraviť výraz `Pole[i-1]+random(10)` aby nasledujúca hodnota bola vždy väčšia ako predchádzajúca?

### Príklad 11.2

Vytvorte efektívny podprogram na zistenie, či sa zadaná hodnota nachádza v utriedenom poli.

```
function BinVyskyt (HlHodnota: integer): boolean;
var DH, HH, Stred: integer;                    //Dolna Hranica, Horna Hranica
begin
DH := 0; HH := High(Pole);
repeat
        Stred := (DH + HH) div 2;
        if Pole[Stred] <> HlHodnota
        then if Pole[Stred] < HlHodnota
              then DH := Stred + 1
             else HH := Stred -1
until (Pole[Stred] = HlHodnota) or (DH > HH);
BinVyskyt := Pole[Stred] = HlHodnota;
end;
```

### Príklad 11.3

Vytvorte efektívny podprogram na zistene miesta výskytu zadanej hodnoty v utriedenom poli.

*Analýza*

Použijeme analogickú úvahu ako v príklade 10.2.

```
function BinMiesto (HlHodnota: integer; var Stred: integer): boolean;
var DH, HH: integer;
begin
DH := 0; HH := High(Pole);
repeat
        Stred := (DH + HH) div 2;
        if Pole[Stred] <> HlHodnota
        then if Pole[Stred] < HlHodnota
              then DH := Stred + 1
              else HH := Stred -1
until (Pole[Stred] = HlHodnota) or (DH > HH);
BinMiesto := Pole[Stred] = HlHodnota;
end;
```

Ktoré miesto výskytu nájde funkcia BinMiesto (prvé, posledné, všetky, nevieme povedať)?

### Príklad 11.4

Vytvorte podprogram, ktorý efektívne zistí, či sa v utriedenom poli vyskytuje zadaná hodnota, a ak nie, nech ju vloží do poľa na také miesto, aby pole zostalo utriedené.

```
procedure BinVlozit (HlHodnota: integer);
var DH, HH, Stred: integer;
begin
DH := 0; HH := High(Pole);
repeat
        Stred := (DH + HH) div 2;
        if Pole[Stred] <> HlHodnota
        then if Pole[Stred] < HlHodnota
              then DH := Stred + 1
              else HH := Stred -1
until (Pole[Stred] = HlHodnota) or (DH > HH);
 // 2. časť
if DH > HH
then begin
        if Pole[Stred] < HlHodnota then Stred := Stred + 1;
        SetLength(Pole, Length(Pole)+1);
        for i := High(Pole)-1 downto Stred do Pole[i+1] := Pole[i] ;
        Pole[Stred] := HlHodnota;
     end;
end;
```

Algoritmus v procedúre BinVlozit sa skladá z dvoch častí. V prvej polovici procedúry zisťuje, či sa HlHodnota nachádza v poli. Premennou Stred sa čoraz viac blíži k miestu, kde by sa mala v utriedenom poli nachádzať HlHodnota. Vypočíta stred medzi indexami DH a HH a ak stredná hodnota Pole[Stred] nie je hľadanou, keďže pole je utriedené, vie rozhodnúť, či má pokračovať v hľadaní v časti Pole[DH] až Pole[Stred-1] alebo Pole[Stred+1] až Pole[HH]. Približovanie končí nájdením hľadanej hodnoty alebo ak už nie je kde hľadať, t.j. DH > HH.

Po skončení prvej časti algoritmu si treba uvedomiť, že ak sa hľadaná hodnota v poli našla, v premennej Stred je jej pozícia v poli a ak nie, čo je pre nás dôležitejšie, premenná Stred „ukazuje“ priamo na miesto alebo tesne pred miesto, kde má byť hľadaná hodnota vložená (keďže v prvej časti sa k tomuto miestu hodnoty Stred blížia). Napríklad pri každom poli, kde HlHodnota je najväčšia, sa Stred zastaví na poslednom prvku poľa, t.j. pred miestom, kde treba vložiť hľadanú hodnotu. Ak teda Stred ukazuje pred miesto, kde má byť HlHodnota vložená, treba Stred posunúť o jednu pozícii doprava (zabezpečí príkaz if). Po zväčšení poľa o jedna následne cyklus for posunie všetky prvky poľa od pozície Stred o jedna doprava, čím spraví miesto pre vloženie hľadanej hodnoty na pozíciu Stred.

Procedúru môžeme napísať aj ako funkciu z ktorej sa vyvezie True, ak bola hodnota do poľa vložená, inak False.
Prečo nemožno použiť nárazník pri vyhľadávaní v utriedenom poli?

V treťom ročníku nepovinná časť:

**Časová výpočtová zložitosť lineárneho a binárneho spôsobu vyhľadávania**

Asymptotická časová výpočtová zložitosť uvedených procedúr lineárneho vyhľadávania je lineárna, t.j. O(n), kde n je počet prvkov poľa (musíme rádovo spracovať všetky prvky poľa).
Asymptotickú časovú výpočtovú zložitosť binárneho vyhľadávania môžeme odhadnúť nasledujúcou úvahou:
Ak by sme sa zahrali hru na uhádnutie mysleného celého čísla napr. od 1 po 1000 a protihráč bude na náš tip oznamov: „Veľa, uber!“ alebo „Málo, pridaj!“, ak uhádneme, samozrejme „Uhádol si!“, musíme myslené číslo uhádnuť, v najhoršom prípade, na desiaty pokus. Podmienkou však je, že musíme za tip vždy vybrať číslo v strede skúmaného intervalu. Takže naše tipy by mali byť: 500, na odpoveď Veľa, uber! 250, na odpoveď Málo, pridaj! 750 atď. (skúste si nakresliť niekoľko úrovní binárneho stromu, ktorý vznikne). k pokusov nám teda umožňuje, pri dôslednom delení skúmaného intervalu na polovice, vybrať správne číslo z $2^k$ čísel. Napríklad najviac desať pokusov nám umožňuje uhádnuť ľubovoľné celé číslo z intervalu 1 až 1024. Ak n je počet čísel a k počet pokusov, zrejme platí $2^k = n$. Keďže nás zaujíma počet potrebných pokusov – porovnaní, t.j. k v rovnici $2^k = n$, zlogaritmovaním so základom 2 dostávame: $\log_2 2^k = \log_2 n$ a po ďalšej matematickej úprave: $k = \log_2 n$. Takže asymptotická časová zložitosť binárneho vyhľadávania je $O(\log_2 n)$ – logaritmická.
Úvaha na odvodenie asymptotickej časovej zložitosti binárneho vyhľadávania by mohla byť aj nasledujúca:
Najprv máme preskúmať celú množinu čísel, t.j. n, po prvom porovnaní už len polovicu z celej množiny (n/2), po druhom štvrtinu (n/4), po treťom porovnaní osminu čísel (n/8) atď. V najhoršom prípade po k preskúmaní – porovnaní zostáva preskúmať jeden prvok. Preto platí $n/2^k = 1$ resp. $2^k = n$, z čoho, po vyššie uvedených úpravách, dostávame logaritmickú časovú zložitosť.

*Poznámky*:

## Triedenie

je činnosť, po skončení ktorej, pre všetky prípustné hodnoty indexu poľa napríklad Pole, platí: Pole[i] ≤ Pole[succ(i)]. Pripomíname, že funkcia succ(i) znamená nasledovník i.
Aj keď prvky môžu byť utriedené **vzostupne** alebo **zostupne** (Pole[i] ≥ Pole[succ(i)]), pod triedením štandardne rozumieme vzostupné usporiadanie utriedených prvkov.
Rozlišujeme algoritmy **vnútorného** a **vonkajšieho** triedenia. Vnútorné triedenie sa nazýva aj triedenie polí a používame ho vtedy, ak sa celá množina dát určená na triedenie zmestí do vnútornej pamäte počítača. Údaje sa uložia do poľa a triediaci algoritmus má k nim priamy prístup. Vonkajšie triedenie pracuje s dátami uloženými v súbore na vonkajšej pamäti, pretože sa všetky nezmestili do vnútornej pamäte počítača. Vonkajším triedením sa nebudeme zaoberať.
Ako pri vyhľadávaní, aj pri triedení môžu mať triedené prvky štruktúru (najčastejšie typu record), potom triedenie prebieha podľa tzv. **kľúča**, rozhodujúcej (kľúčovej) položky, ostatné položky prvku sú s ňou len premiestňované.
Algoritmov vnútorného triedenia je viacej a navzájom sa líšia zložitosťou a tomu zodpovedajúcou efektívnosťou triedenia (čím jednoduchší algoritmus, tým menej efektívny). Pri výbere triediaceho algoritmu možno prihliadať aj na veľkosť vstupných údajov, pri niekoľko sto prvkových poliach je najpomalší triediaci algoritmus prakticky rovnako rýchly ako ten najrýchlejší, ktorého napísanie nám môže trvať výrazne dlhší čas.
Najjednoduchšie vnútorné triediace algoritmy majú niekoľko spoločných rysov - algoritmus je krátky a jednoduchý, časová zložitosť je kvadratická. Sem patria napríklad bublinkové triedenie (triedenie výmenou, Bublesort), triedenie priamym vkladaním (vsúvaním, Insertsort), triedenie priamym výberom (Selectsort).

## Bublinkové triedenie (Bubble sort)

Už samotné bublinkové triedenie môže mať veľa variant, od dvoch vnorených cyklov s pevným počtom opakovaní, až po vonkajší cyklus s opakovaním, pokým dochádza k výmene pri prechode poľom. Porovnávať tiež môžeme prvky Pole[i-1] a Pole[i], alebo Pole[i] a Pole[i+1], ísť od prvého alebo posledného prvku poľa a pod.

### Príklad 12.1

Vytvorte triediaci algoritmus, ktorý bude porovnávať dva vedľa seba stojace prvky, a ak je ľavý väčší ako pravý, vymení ich. Určte potrebný počet prechodov poľom.
Napríklad

| | | | | | |
|---|---|---|---|---|---|
| Pôvodné pole | 5 | 3 | 4 | 2 | 1 |
| Po 1.prechode | 3 | 4 | 2 | 1 | 5 |
| Po 2.prechode | 3 | 2 | 1 | 4 | 5 |
| Po 3.prechode | 2 | 1 | 3 | 4 | 5 |
| Po 4.prechode | 1 | 2 | 3 | 4 | 5 |

Z príkladu vidieť, že poľom treba prejsť pri N prvkoch N-1 krát. Hodnota N je dĺžka poľa, teda N = Length(Pole).

```
procedure BublinkoveTriedenie1;
var Prechod, i, Pom: integer;
begin
for Prechod := 1 to Length(Pole)-1 do
        for i := 0 to High(Pole) - Prechod do
                if Pole[i] > Pole[i+1]
                then begin Pom:= Pole[i]; Pole[i]:= Pole[i+1]; Pole[i+1]:= Pom; end;
end;
```

Pri uvedenej procedúre sa poľom prejde Length(Pole)-1 krát, kde Length(Pole) udáva počet prvkov poľa (vonkajší cyklus). Pri každom prechode poľom (vnútorný cyklus) sa porovnávajú dva vedľa seba ležiace prvky (Pole[i] a Pole[i+1]) a ak je ľavý prvok väčší ako pravý, vymenia sa. Po prvom prechode poľom musí byť na svojom mieste najväčší prvok v poli, po druhom druhý najväčší atď. čo umožňuje skracovať vnútorný cyklus o hodnotu Prechod (po prvom prechode poľom už netreba porovnať posledný prvok, po druhom prechode ani predposledný prvok atď.).
Po Length(Pole)-1 prechodoch poľom (vonkajší cyklus) je na svojich miestach sprava počet-1 prvkov a teda pole je utriedené (najmenší prvok – prvý zľava, nemá inú možnosť, len byť na prvom mieste v poli, keďže všetky väčšie prvky boli presunuté doprava).

Ujasnite si ďalšie možnosti triedenia výmenou (s premenou Pole[i-1], s verziou downto a pod.). Výmenu umiestnite do lokálnej procedúry Vymen s parametrami; ako parametre voľte hodnoty poľa alebo indexy prvkov, ktoré treba vymeniť.
Vytvorte verziu bublinkového triedenia, v ktorej, ak už nenastala výmena pri prechode poľom, triedenie sa ukončí.

### Príklad 12.2

Vytvorte procedúru, ktorá zistí a vypíše K najmenších prvkov poľa (0 < K ≤ počet prvkov poľa).
*Analýza*
Algoritmus bublinkového triedenia možno napísať aj tak, že po prvom prechode poľom bude najmenší prvok na svojom mieste, po druhom prechode druhý najmenší atď. Zrejme po K-prechodoch bude K najmenších prvkov v poli na svojom mieste a stačí len vypísať túto utriedenú časť poľa.

```
procedure TForm1.btKNajmensichClick(Sender: TObject);
var Prechod, i, K: integer;
    Riadok: string;
procedure Vymen(var x, y: integer);                    //lokálna procedúra
        var Pom: integer;
        begin
        Pom:= x; x:= y; y:= Pom;
        end;
begin
K:= StrToInt( InputBox( 'K-miním' , 'Vypísať najmenších' , '5' ) );
for Prechod:=1 to K do                                 // K prechodov poľom
        for i:= High(Pole) downto Prechod do
                if Pole[i-1] > Pole[i] then Vymen( Pole[i-1] , Pole[i] );
//Vypísanie prvých K hodnôt do riadka
Riadok:= '';
for i:= 0 to K-1 do Riadok:= Riadok + Format('%4d',[Pole[i]]);
Memo1.Lines.Add(Riadok);
end;
```

Zadanie nehovorí, že všetkých K najmenších prvkov poľa musí mať rôzne hodnoty. Ako by sme museli upraviť procedúru, aby zistilo a vypísalo K rôznych najmenších prvkov poľa? Uvažujte aj o alternatíve, že v poli nie je K rôznych prvkov.

## Triedenie priamym vkladaním (Insertion sort)

Ako už naznačuje názov algoritmu, postupne berieme druhý, tretí, štvrtý až posledný prvok poľa a vkladáme ich na „správne“ miesto vľavo od pôvodnej pozície vkladaného prvku. „Správne“ miesto pre vkladaný prvok je také, že vľavo sú už len menšie čísla a vpravo (po pôvodnú pozíciu) väčšie alebo rovné hodnote vkladaného prvku. Pole je v tejto časti čiastočne utriedené.

Napríklad

| | | | | | |
|---|---|---|---|---|---|
| Pôvodné pole | 5 | 3 | 4 | 2 | 1 |
| 2.prvok na svoje miesto | 3 | 5 | 4 | 2 | 1 |
| 3.prvok na svoje miesto | 3 | 4 | 5 | 2 | 1 |
| 4.prvok na svoje miesto | 2 | 3 | 4 | 5 | 1 |
| 5.prvok na svoje miesto | 1 | 2 | 3 | 4 | 5 |

Z príkladu vidieť, že poľom treba prejsť pri N prvkoch N-1 krát.

## Príklad 12.3

Vytvorte procedúru na utriedenie poľa priamym vkladaním.

```
procedure TriedenieVkladanim;
var i, j: integer;
    x: integer;                              //typ zložky poľa
begin
for i := 1 to High(Pole) do
begin
        x := Pole[ i ]; j := i-1;
        while Pole[ j ] > x do
        begin
                Pole[ j+1 ]:= Pole[ j ];
                j := j-1;
                if j < 0 then break;
        end;
Pole[ j+1 ] := x;
end
end;
```

Algoritmus postupne berie 2.prvok poľa (s indexom 1), 3.prvok poľa až posledný prvok poľa (s indexom High(Pole)). Odloží vkladanú hodnotu do x (`x := Pole[i];`) a nastaví porovnanie na prvý prvok vľavo od vkladaného (`j:= i-1;`). Každý väčší prvok ako vkladaný sa posunie o jednu pozíciu doprava (`Pole[j+1]:= Pole[j]`), pretože „niekde vľavo“ bude vložený vkladaný prvok. Index j sa zastaví pod miestom, kde treba vložiť prvok (v „najhoršom“ prípade na j = -1), preto sa vkladá na j+1. miesto (`Pole[j+1] := x`). Pri testovaní podmienky while nastáva problém, lebo hodnota Pole[-1] neexistuje (ak vynecháme príkaz `if j < 0 then break;`). Preto pre hodnotu j < 0 musíme „násilne“ príkazom break ukončiť vnútorný cyklus. Inou alternatívou by bolo použitie pomocnej premennej Hladat typu boolean, príslušná časť procedúry

```
...
Hladat:= Pole[j] > x;
while Hladat do
begin
        Pole[j+1]:= Pole[j];
        j:= j-1;
        if j < 0 then Hladat:= False else Hladat:= Pole[j] > x;
end;
Pole[j+1]:= x;
...
```

Keďže úsek poľa od 0 po i-1 je utriedený, možno použiť na nájdenie miesta na vloženie prvku Pole[i] aj binárne vkladanie. Ako by principiálne pracoval daný algoritmus?

Ak je pole už na začiatku viac-menej utriedené, triedenie priamym vkladaním vykoná menej úkonov (je efektívnejšie) ako bublinkové triedenie pre to isté pole.

### Príklad 12.4

Vytvorte program, ktorý umožní postupné vkladanie čísel a po vložení každého čísla vypíše utriedené doposiaľ vložené čísla. Počet vkladaných čísel sa buď zadá po spustení programu alebo vkladanie skončí po vložení tzv. koncovej hodnoty (napríklad nuly).

## Triedenie priamym výberom (Selection sort)

Vyberieme najmenší prvok poľa a vymeníme ho s prvkom na prvom mieste v poli, potom vyberieme druhý najmenší a vymeníme ho s prvkom na druhej pozícii v poli atď. až po predposledný prvok poľa (najväčší prvok poľa musí už byť na poslednom - správnom mieste). Inou možnosťou je vybrať najväčší prvok poľa a vymeniť ho s posledným, následne vybrať druhý najväčší a vymeniť atď. „Efektívnejšie" je vyberať prechodom cez pole hneď minimum aj maximum skúmanej časti poľa, čo robí procedúra TriedenieMinMax.

Symbolika: Pole[DH] ◄-----► Pole[iMin] Pole[iMax] ◄----► Pole[HH]

DH iMin iMax HH

i ⟶

po výmene Pole[DH] a Pole[iMin] a Pole[HH] a Pole[iMax] sa zľava aj sprava zúži prehľadávaný interval na DH := DH+1 a HH := HH-1.

Napríklad

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Pôvodné pole | 5 | 3 | 4 | 2 | 1 | DH = 0 | HH = 4 |
| Po 1.prechode | 1 | 3 | 4 | 2 | 5 | DH = 1 | HH = 3 |
| Po 2.prechode | 1 | 2 | 3 | 4 | 5 | DH = 2 | HH = 2 |

### Príklad 12.5

Vytvorte procedúru na utriedenie poľa priamym výberom pomocou miním aj maxím poľa.

```
procedure TriedenieMinMax;
var i, DH, HH, iMin, iMax: integer;
  procedure Vymen (var x, y: integer);
  var pom: integer;
  begin
    pom := x; x := y; y := pom
  end;
begin
DH := 0; HH := High(Pole);
while DH < HH do
begin
        i := DH;
        iMin := DH;
        iMax := DH;
        while i < HH do
        begin
                i := i + 1;
                if Pole[i] < Pole[iMin] then iMin := i;
                if Pole[i] > Pole[iMax] then iMax := i;
        end;
```

```
        Vymen (Pole[DH], Pole[iMin]);
        if DH <> iMax then Vymen (Pole[HH], Pole[iMax])
        else Vymen (Pole[HH], Pole[iMin]);
        DH := DH + 1; HH := HH – 1;
end;
end;
```

Prečo index maxima (iMax) nastavujeme na DolnúHranicu (DH) a nie HornuHranicu (HH)? Prečo musíme odlíšiť, či DH <> iMax? Ako by vyzeral príkaz if, keby sme začali výmenou Vymen (Pole[HH], Pole[iMax])?

Aj algoritmus triedenia priamym výberom by sme vedeli použiť pri riešení príkladu 12.2.

### Príklad 12.6

Vytvorte procedúru, ktorý zistí počet rôznych hodnôt v zadanom poli.
*Analýza*
Jednoduchým riešením je utriediť pole a postupne porovnať susedné hodnoty. Ak je nasledujúca hodnota väčšia ako predchádzajúca, našla sa ďalšia nová hodnota a zrejme treba zvýšiť počet rôznych hodnôt poľa o jeden.
Po utriedení teda môžeme použiť cyklus:

```
PocetRoznych:= 1;
for i:= Low(Pole) to High(Pole) - 1 do
        if Pole[i+1] > Pole[i] then inc(PocetRoznych);
Memo1.Lines.Add( 'Počet rôznych hodnôt v poli: ' + IntToStr(PocetRoznych) );
```

### Príklad 12.7

Vytvorte procedúru, ktorá zistí počet rôznych znakov zadaného reťazca.

### Príklad 12.8

Vytvorte procedúru, ktorá zistí, či sa v poli vyskytujú dve rovnaké hodnoty.
*Analýza*
Úloha môže mať (ako každá) viacej riešení. Keďže si chceme precvičiť triediace algoritmy, pokúsime sa ich využiť.
Ak pole utriedime, rovnaké hodnoty by mali byť pri sebe. Potom už len stačí prejsť poľom a zistiť, či pre nejaké dovolené i platí Pole[i] sa rovná Pole[i-1]. Najefektívnejšie je zastaviť triedenie hneď, ako budú pri sebe dve rovnaké hodnoty. Využijeme posledný triediaci algoritmus, ktorý zoraďuje prvky od začiatku aj konca poľa. Doplnené časti sme zvýraznili.

```
function RovnakeHodnoty: boolean;
var i, DH, HH, iMin, iMax: integer;
    Stop: boolean;

procedure Vymen (var x, y: integer);
    var pom: integer;
    begin
    pom := x; x := y; y := pom
    end;
```

```
begin            //začiatok funkcie RovnakeHodnoty
DH := 0; HH := High(Pole); Stop:= False;
while (DH < HH) and not Stop do
begin
        i := DH;
        iMin := DH;
        iMax := DH;
        while i < HH do
        begin
                i := i + 1;
                if Pole[i] < Pole[iMin] then iMin := i;
                if Pole[i] > Pole[iMax] then iMax := i;
        end;
        Vymen (Pole[DH], Pole[iMin]);
        if DH <> iMax
        then Vymen (Pole[HH], Pole[iMax])
        else Vymen (Pole[HH], Pole[iMin]);
        if DH>0 then Stop:= ( Pole[DH] = Pole[DH-1] ) or ( Pole[HH] = Pole[HH+1] );
        DH := DH + 1; HH := HH - 1;
end;
if not Stop
then if DH>=HH then Stop:= ( PoleI[DH] = PoleI[DH-1] ) or ( PoleI[HH] = PoleI[HH+1] );
Result:= Stop
end;
```

Keďže test pre **Stop** v cykle prebieha len ak je DH < HH, musíme po skončení cyklu while ešte otestovať, ak sa dovtedy nenašli zhodné prvky, aj prípady, keď DH >= HH.

## Príklad 12.9

Pole obsahuje celé čísla od 1 po N, každé práve raz. Jedna z hodnôt bola prepísaná na nulu. Vytvorte program, ktorý zistí, ktorá hodnota bola prepísaná.

*Analýza*

Majme napríklad pole s hodnotami 1 až 5, takto by vyzeralo po utriedení:

| index | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| hodnota | 1 | 2 | 3 | 4 | 5 |

takto po prepísaní napríklad hodnoty 2 na nulu a utriedení:

| index | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| hodnota | 0 | 1 | 3 | 4 | 5 |

V zmenenom poli po utriedení zrejme na mieste s indexom nula bude nula, s indexom jedna bude jednotka atď. len od určitého miesta-indexu nebude platiť, že Pole[i]=i ale Pole[i]=i+1. Toto miesto hľadáme. Opäť môžeme využiť triediaci algoritmus, v ktorom je po prvom prechode prvý prvok na svojom mieste, po druhom prechode druhý prvok atď. Triedenie zastavíme, keď po prechode poľom nebude platiť Pole[i] = i. Hodnota i je hľadanou prepísanou hodnotou. S využitím bublinkového triedenia má funkcia tvar

```
function ZmenenaHodnota: integer;
var index, i: integer;

procedure Vymen(var x, y: integer);
    var Pom: integer;
    begin
```

```
        Pom:= x; x:= y; y:= Pom;
    end;
begin           //začiatok funkcie ZmenenaHodnota
index:= -1;
repeat
        inc(index);
        for i:= High(Pole) downto 1 do if Pole[i-1] > Pole[i] then Vymen( Pole[i-1] , Pole[i] );
until (Pole[index] = index+1);
Result:= index;
end;
```

Príkazová časť funkcie ZmenenaHodnota bez repeat s použitím príkazu break

```
begin
for index:= 0 to High(Pole) do
begin
        for i:= High(Pole) downto 1 do if Pole[i-1] > Pole[i] then Vymen( Pole[i-1] , Pole[i] );
        if Pole[index] = index+1 then break;
end;
Result:= index;
end;
```

Keďže sa nám nechcelo zadávať hodnoty poľa ručne, vytvorili sme procedúru, ktorá nám pole s požadovanými vlastnosťami vytvorí.

```
procedure VytvorPolePr12_9;
var N, i, j, Miesaj, Pom: integer;
begin
N:= StrToInt( InputBox( 'Vytvor' , 'Počet prvkov poľa' , '100' ) );
SetLength(Pole, N);
for i:= 0 to High(Pole) do Pole[i]:= i+1;          //vytvorí pole s hodnotami 1, 2, 3,..., N
for Miesaj:= 1 to N div 2 do                       //N div 2 krát
begin                                              // náhodne vymení hodnoty s indexami i a j
        i:= random(N); j:= random(N);
        Pom:= Pole[i]; Pole[i]:= Pole[j]; Pole[j]:= Pom;
end;
Pole[random(N)]:= 0;                               // jednu z hodnôt zmení na nulu
end;
```

### Príklad 12.10

Vytvorte program s dvoma poliami. V jednom poli budú mená študentov, v druhom ich priemery zaznamenané s presnosťou na stotiny. Program nech umožňuje utriediť údaje podľa mien aj podľa priemerov.

*Analýza*

V úlohe chceme poukázať na triedenie spolu súvisiacich dát uložených v dvoch poliach. Prislúchajúce údaje v poliach Meno a Priemer sú „prepojené“ rovnakou hodnotou indexu. Znamená to, že ak triedime napríklad pole Meno a vymieňame i-tu a i+1. hodnotu, súčasne musíme v poli Priemer tiež vymeniť i-tu a i+1. hodnotu. Ako zaujímavosť uvádzame vygenerovanie priemerov s presnosťou na stotiny z intervalu < 1 , 5 ). Výraz random(4) + 1 vráti náhodné celé číslo od 1 po 4. Výraz int(random*100) vráti celú časť z náhodného reálneho čísla z intervalu < 0.0 , 100.0 ) a po predelení 100 vráti reálne číslo z intervalu < 0.00 , 1.00 ) v ktorom od tisícin sú nuly.

```
const StartPocet = 10;                  //počet údajov v poliach po spustní programu
var   Meno: array of string[9];         //počet znakov v mene sme ohraničili na 9
      Priemer: array of real;

procedure TForm1.FormActivate(Sender: TObject);
begin
randomize;
Memo1.Font.Name:= 'Courier';            //písmo Courier kvôli krajšiemu výpisu
end;

procedure TForm1.btVytvorPoliaClick(Sender: TObject);
var i: integer;
begin
setlength(Meno, StartPocet);            //hodnoty poľa Meno nemožno jednoducho
Meno[0]:='Juraj';                       //generovať, preto sme si ich zvolili
Meno[1]:='Zuzana';
Meno[2]:='Peter';
Meno[3]:='Jana';
Meno[4]:='Darina';
Meno[5]:='Alexandra';
Meno[6]:='Pavol';
Meno[7]:='Zora';
Meno[8]:='Jozef';
Meno[9]:='Anna';
setlength(Priemer,StartPocet);
for i:= 0 to StartPocet-1 do Priemer[i]:= random(4)+1 + int(random*100)/100;
Memo1.Lines.Clear;
for i:= 0 to High(Meno) do Memo1.Lines.Add( Format( '%-10s %f' , [ Meno[i] , Priemer[i] ] ));
end;

procedure TForm1.btVypisPoliaClick(Sender: TObject);
var i: integer;
begin
Memo1.Lines.Add('---------------');
for i:= 0 to High(Meno) do Memo1.Lines.Add(Format( '%-10s %f' , [ Meno[i] , Priemer[i] ] ));
end;

procedure TForm1.btUtriedPodlaMienClick(Sender: TObject);
var PP, i, Pocet: integer;
    Pom1: string;
    Pom2: real;
begin
Pocet:= length(Meno);
for PP:= 1 to Pocet-1 do                        //bublinkové triedenie
        for i:= 0 to Pocet-PP-1 do
                if Meno[i] > Meno[i+1]          //triedime podľa mien
                then begin
                        Pom1:= Meno[i]; Pom2:= Priemer[i];
                        Meno[i]:= Meno[i+1]; Priemer[i]:= Priemer[i+1];
                        Meno[i+1]:= Pom1; Priemer[i+1]:= Pom2;
                     end;
end;
```

```
procedure TForm1.btUtriedPodlaPriemerovClick(Sender: TObject);
var PP, i, Pocet: integer;
    Pom1: string;
    Pom2: real;
begin
Pocet:= length(Meno);
for PP:= 1 to Pocet-1 do                                    //bublinkové triedenie
        for i:= 0 to Pocet-PP-1 do
                if Priemer[i] > Priemer[i+1]                 //triedime podľa priemerov
                then begin
                        Pom1:= Meno[i]; Pom2:= Priemer[i];
                        Meno[i]:= Meno[i+1]; Priemer[i]:= Priemer[i+1];
                        Meno[i+1]:= Pom1; Priemer[i+1]:= Pom2;
                     end;
end;
```

Nepovinný text

## Rýchle triedenie (Quick sort)

Vyššie uvedené triediace algoritmy boli nerekurzívne a ich časová výpočtová zložitosť bola kvadratická. Najrýchlejším známym triediacim algoritmom s O(n.$\log_2$ n) je quicksort, ktorý jeho autor C.A.Hoare nazval rýchle triedenie. Uvedieme jeho rekurzívnu verziu, aj keď existuje aj nerekurzívna verzia využívajúca dátovú štruktúru zásobník.

### Príklad 12.11

Vytvorte procedúru na utriedenie poľa rekurzívnym algoritmom rýchleho triedenia.

```
procedure Quicksort(DH, HH: integer);
var i, j, X, Pom: integer;
begin
X := Pole [(DH+HH) div 2];
i := DH; j := HH;
while i <= j do
begin
        while Pole [i] < X do i := i + 1;
        while Pole [j] > X do j := j - 1;
        if i <= j
        then begin
                Pom := Pole [i]; Pole [i] := Pole [j]; Pole [j] := Pom;
                i := i + 1;
                j := j – 1;
             end;
end;
if DH < j then Quicksort (DH, j);
if i < HH then Quicksort (i, HH);
end;
```

Algoritmus pracuje nasledovne: do X vloží hodnotu prvku ležiaceho v strede oblasti DH,...,HH; index i nastaví na ľavý okraj skúmanej oblasti, t.j. DH, index j na pravý okraj skúmanej oblasti, t.j. HH; index i zvyšuje, kým nenájde prvok väčší alebo rovný X, index j znižuje, kým nenájde prvok menší alebo rovný X; ak ešte nedošlo k prekrytiu indexov (i ≤ j), vymení medzi sebou prvky

Pole [i] a Pole [j], zvýši index i o 1 a zníži index j o 1; ďalej pokračuje v hľadaní zľava prvkov väčších alebo rovných X a sprava prvkov menších alebo rovných X a v ich následnej výmene; po prejdení celej oblasti (i > j) aplikuje vyššie opísaný algoritmus (preto rekurzia - ten istá činnosť, len so zmenenými hranicami!) na oblasť DH,...,j a na oblasť i,...,HH až kým nie je v skúmanej oblasti len jeden prvok.

Zaujímavosťou uvedeného algoritmu je skutočnosť, že keď v zdanlivo izolovaných oblastiach sa vymenia prvky podľa stredov oblastí (väčšie alebo rovné X doprava, menšie alebo rovné X doľava), pole zostane utriedené.

# Ďalšie úlohy na dynamické pole

## Príklad 13.1

V úvode jednorozmerného poľa sme uviedli motivačný príklad, ku ktorému sme sľúbili vrátiť sa neskôr. V nasledujúcich procedúrach využívame dynamické polia, ktorých hodnoty sú reálne čísla, na spracovanie fyzikálneho merania.

*Analýza*

Nepôjdeme do detailov, potrebná je znalosť spracovania fyzikálnych meraní z prvého ročníka gymnázia. Program počíta aritmetický priemer, absolútnu a relatívnu odchýlku buď zadaného alebo vopred neznámeho počtu nameraných hodnôt. Zobrazí tabuľku nameraných hodnôt.

dx[i] znamená $\Delta x_i$ a deltax znamená $\delta x$.

```
var x: array of real;                          //globálne pole na zapamätanie nameraných hodnôt

//procedúra na zapamätanie nameraných hodnôt
procedure TForm1.btMeranieClick(Sender: TObject);
var N, i: integer; Hodnota: string;
begin
N:= StrToInt(InputBox('Fyzikálne merania','Počet nameraných hodnôt (0 znamená neznámy)','10'));
if N>0
then begin
        SetLength(x, N);
        for i:= 0 to High(x) do x[i]:= StrToFloat(InputBox('FyzMer',IntToStr(i+1)+'. hodnota',''));
      end
else repeat
        Hodnota:= InputBox('Fyzikálne merania',IntToStr(N+1)+'. hodnota (Koniec -> Enter','');
        if Hodnota<>''
        then begin
                SetLength(x, N+1);
                x[N]:= StrToFloat(Hodnota);
                N:= N+1;
             end;
     until Hodnota='';

//zobrazenie zadaných hodnôt v jednoduchej tabuľke
Memo1.Clear;
Memo1.Lines.Add('Namerané hodnoty');
Memo1.Lines.Add('----------------------------');
Memo1.Lines.Add('Číslo merania       x');
Memo1.Lines.Add('----------------------------');
for i:= 0 to High(x) do Memo1.Lines.Add(Format('%7d %17.3f',[i+1,x[i]]));
Memo1.Lines.Add('----------------------------');
end;

//procedúra na spracovanie hodnôt uložených v poli x
procedure TForm1.btVyhodnotClick(Sender: TObject);
var xpriemer, sucetx, sucetdx, dxpriemer, deltax: real;
    i, Presnost: integer;
    dx: array of real;  //pole na uloženie odchýlok jednotlivých meraní od aritmetického priemeru
begin
```

```
Presnost:= StrToInt(InputBox('FyzMer','Presnosť merania - počet desat.miest','2'));
sucetx:= 0;
for i:= 0 to High(x) do sucetx:= sucetx + x[i];
xpriemer:= sucetx / length(x);
SetLength(dx, Length(x));
sucetdx:= 0;
for i:= 0 to High(x) do
begin
        dx[i]:= x[i] - xpriemer;
        sucetdx:= sucetdx + abs(dx[i]);
end;
dxpriemer:= sucetdx / Length(dx);
deltax:= dxpriemer / xpriemer *100;

//zobrazenie nameraných hodnôt aj s výsledkami
Memo1.Clear;
Memo1.Lines.Add('--------------------------------------------------');
Memo1.Lines.Add('Číslo merania          x               dx');
Memo1.Lines.Add('--------------------------------------------------');
for i:= 0 to High(x) do
        Memo1.Lines.Add(Format('%7d %20.*f %20.*f',[i+1,Presnost,x[i],Presnost,dx[i]]));
Memo1.Lines.Add('--------------------------------------------------');
Memo1.Lines.Add(Format('x = %0.*f +/- %0.*f',[Presnost,xpriemer,Presnost, dxpriemer]));
Memo1.Lines.Add(Format('dx = %f%%',[deltax]));
end;
```

### Príklad 13.2

Vytvorte program, ktorý vypíše všetky prvočísla po zadané prirodzené číslo s využitím Eratostenovho algoritmu.

*Analýza*

Eratostenov algoritmus nazývaný tiež Eratostenovo sito pracuje nasledovne:

1. vypíšeme všetky prirodzené čísla od 2 po zadané číslo
2. 2 označíme ako prvočíslo a všetky jej násobky v zozname, t.j. 4, 6, 8, 10,... škrtneme
3. označíme ďalšie neškrtnuté číslo ako prvočíslo a
4. všetky jeho násobky v zozname škrtneme
5. kroky 3 a 4 opakujeme až na koniec zoznamu

Zoznam ľahko vytvoríme ako dynamické pole, z ktorého využijeme prvky s indexami od 2 po zadané číslo. Hodnoty všetkých týchto prvkov nastavíme na True, čo znamená „neškrtnuté“ - predbežne sú prvočíslami. Následne realizujeme 3. a 4. bod algoritmu, t.j. vypíšeme index najbližšieho prvku s hodnotou True a hodnoty všetkých násobkov indexu zmeníme na False – „škrtneme“. Napríklad po „škrtnutí“ násobkov dvojky sú v poli Cisla hodnoty

| Cislo | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Cisla[Cislo] | T | T | F | T | F | T | F | T | F | T | F |

ďalším krokom by bolo vypísanie trojky a zmena True na False pre Cisla[6], Cisla[9] a Cisla[12].

Dynamické pole s hodnotami typu boolean, t.j. s hodnotami False a True nám umožňuje zaznamenať „škrtnuté“ a „neškrtnuté“, pričom index prvku reprezentuje prirodzené číslo a pri hodnote „neškrtnuté“ (True) aj prvočíslo.

```
procedure TForm1.btPrvocislaClick(Sender: TObject);
var Cisla: array of boolean;              //dynamické pole s hodnotami typu boolean
    Po, Cislo, i, Krok: integer;
begin
Po:= StrToInt(Edit1.Text);                //vypísať všetky prvočísla po
inc(Po);                                  //hodnotu Po musíme zvýšiť o jeden, aby sme dostali index Po
SetLength(Cisla, Po);
for Cislo:= 2 to High(Cisla) do Cisla[Cislo]:= True;          //všetky čísla od 2 sú „neškrtnuté“
for Cislo:= 2 to High(Cisla) do
begin
        if Cisla[Cislo] then Memo1.Lines.Add( IntToStr(Cislo) );      //ak je „neškrtnuté“ vypíš
        Krok:= Cislo; i:= Cislo + Krok;                       //výpočet prvého násobku prvočísla
        while i <= Po do                                      //pokiaľ nie sme na konci poľa
        begin
                Cisla[i]:= False;                             //“škrtnutie“ násobku prvočísla
                inc(i,Krok);                                  //ďalší násobok prvočísla
        end;
end;
end;
```

Príkaz Cisla[i]:= False; by sme mohli nahradiť aj príkazom if Cisla[i] then Cisla[i]:= False; t.j. „škrtali“ by sa len „neškrtnuté“.

## Príklad 13.3

Najmä ak hodnoty dynamického poľa majú byť typu string, je ťažšie generovať zmysluplné reťazce, preto uvádzame príklad so zadanými počiatočnými hodnotami. Prvky do tohto dynamického poľa možno pridávať, triediť, možno použiť vyhľadávanie atď.

```
implementation
{$R *.dfm}
var Pole: array of string;

procedure TForm1.btVypisClick(Sender: TObject);
var i: integer;
begin
Memo1.Clear;
for i:= 0 to High(Pole) do Memo1.Lines.Add(Pole[i]);
end;

procedure TForm1.btPridajClick(Sender: TObject);
begin
SetLength(Pole, Length(Pole)+1);
Pole[High(Pole)]:= InputBox( 'Vstup' , 'Pridať meno' , '' );
end;

initialization
SetLength(Pole,3);
Pole[0]:='Jano';
Pole[1]:='Fero';
Pole[2]:='Adam';
end.
```

**Príklad 13.4**

Vytvorte program, ktorý efektívne vytvorí dve celočíselné polia (užívateľ zadá počet prvkov jednotlivých polí), vypíše ich a vytvorí tretie pole, ktoré obsahuje najprv prvky prvého poľa a za nimi prvky druhého poľa.

*Riešenie*

Pre efektívne riešenie tohto príkladu je nevyhnutné predovšetkým definovať vlastný údajový typ TPole a „zastrešiť" podeň všetky tri polia A, B a C. Existencia údajového typu s vlastným menom umožňuje v procedúrach použiť parameter Pole. Deklarovanie polí A a C ako rovnaký údajový typ umožňuje použiť príkaz C:= A.

```
type TPole = array of integer;
var   A, B, C: TPole;

procedure Vytvor(var Pole: TPole);
var i: integer;
begin
Setlength( Pole, StrToInt( InputBox( 'Vstup' , 'Počet prvkov' , '5' ) ) );
for i:= 0 to High(Pole) do Pole[i]:= random(100);
end;

procedure Vypis(var Pole: TPole);
var i: integer;
begin
Form1.Memo1.Lines.Add( 'Výpis:' );
for i:= Low(Pole) to High(Pole) do Form1.Memo1.Lines.Add( IntToStr(Pole[i]) );
end;

procedure TForm1.btVYtvorAClick(Sender: TObject);
begin
Vytvor(A);
Vypis(A);
end;

procedure TForm1.btVytvorBClick(Sender: TObject);
begin
Vytvor(B);
Vypis(B);
end;

procedure TForm1.btVytvorCClick(Sender: TObject);
var i: integer;
begin
C:= A;                                    //vykoná automaticky aj setlength( C , length(A) )!
Setlength( C , length(A) + length(B) );   //poradie príkazov nemožno zameniť!
for i:= length(A) to High(C) do C[i]:= B[ i - length(A) ];
Vypis(C);
end;
```

## Statické pole

Dynamické pole má dve výrazné obmedzenia. Index musí byť typu integer a musí začínať od nuly. Tieto obmedzenia nemá statické pole, je to však za cenu straty dynamickosti poľa. Počet a typ prvkov statického poľa musíme poznať už pri písaní programu. Pri spustení programu sa vyhradí pamäť potrebná na uloženie prvkov poľa (počet prvkov * počet bajtov pre prvok) a tá je rezervovaná počas behu programu, či sa plne využije alebo nie. Ak pred spustením programu nepoznáme presný počet prvkov poľa (veľkosť poľa), je pamäťovo efektívnejšie použiť dynamické pole.
Indexom statického poľa môže byť akýkoľvek ordinálny typ, teda okrem typu integer aj typ char, boolean, typ interval a programátorom definovaný typ. Aj index statického poľa môže začínať od ľubovoľnej hodnoty integer, najčastejšie od jednotky, pri type char najčastejšie od znaku A alebo prvého zobraziteľného znaku – medzery.

**Deklarácia premennej typu statické pole má tvar:**

var *IdentifikatorPremennej* : array [ *RozsahIndexu* ] of *TypPrvkuPola*;
pričom *RozsahIndexu* musí byť ordinálneho typu.

Napríklad

```
var A: array [1..10] of integer;          //prem.A typu statické pole so štruktúrou 10 čísel typu integer
    B: array [-5..5] of real;             //prem.B typu statické pole so štruktúrou 11 čísel typu real
    C: array ['A'..'Z'] of integer;       //prem.C typu statické pole so štruktúrou 26 čísel typu integer
    D: array [chr(32)..chr(255)] of integer;   //prem.D typu stat.pole so štr. 224 čísel typu integer
    E: array [byte] of string;            //prem.E typu statické pole so štruktúrou 256 reťazcov
    F: array [2..1000] of boolean;        //prem.F typu stat. pole so štr. 999 hodnôt True alebo False
```

Použitie - definovanie konštanty napríklad

```
const MaxPocPrvkov = 100;
var   Pole: array [ 1..MaxPocPrvkov] of string;
```

**Definícia typu statické pole má tvar:**

type *MenoTypuPole* = array [ *RozsahIndexu*] of *TypPrvkuPola*;
pričom *RozsahIndexu* musí byť ordinálneho typu.

Napríklad

```
type tPole = array [1..20] of integer;                //typ stat.pole so štrukt. 20 čísel typu integer

const MaxPocPrvkov = 100;
type  tPole2 = array [1..MaxPocPrvkov] of real;       //typ stat.pole so štruktúrou 100 čísel typu real
```

Deklarácia premennej pomenovaného typu má tvar:

var *IdentifikatorPremennej* : *MenoTypuPole*;

Napríklad

```
var Pole: tPole;
    X: tPole2;
```

Pripomíname, že typ statické pole musíme definovať, t.j. pole musí mať svoj vlastný názov, ak chceme premennú typu pole použiť ako parameter procedúry.

Napríklad

procedure Zluc(var A, B, C: tPole);    nie    procedure Zluc(var A, B, C: array [1..20] of integer);

**Sprístupnenie prvkov poľa sa realizuje zápisom:**

*MenoPola* [ *IndexovyVyraz* ]

kde *IndexovyVyraz* je rovnakého typu (ordinálneho), ako je typ *RozsahIndexu*.

Napríklad A[1]:= 5;   A[2*i-1]:= i;  B[i+1]:= B[i-1]+B[i]; B[MaxPocPrvkov]:= 'Jano';

### Príklad 14.1

Vytvorte program na zistenie počtu jednotlivých písmen anglickej abecedy (malé a veľké sa nerozlišujú) v zadanom reťazci.

*Analýza*

Napríklad pre reťazec Abeceda zjedla deda by mal výsledok vyzerať nasledovne

A 4
B 1
C 1
D 4
E 4
atď.

Je výhodné použiť pole, v ktorom bude uložený počet jednotlivých písmen, t.j. v prvej hodnote poľa bude uložený počet písmen A v reťazci, v druhej hodnote počet písmen B atď. až v poslednej (26.) hodnote poľa bude uložený počet písmen Z v zadanom reťazci. Zviazať hodnotu poľa s písmenom, ku ktorému patrí, možno použitím písmena ako indexu poľa. Keďže poznáme počet veľkých písmen anglickej abecedy (26), použijeme statické pole resp. premennú Pocet deklarovanú var Pocet: array ['A'..'Z'] of integer; Premenná Pocet[Znak] obsahuje celé číslo – počet výskytov znaku Znak, premenná Pocet[ Retazec[i] ] zrejme obsahuje celé číslo – počet výskytov znaku Retazec[i].

```
procedure TForm1.btZistiClick(Sender: TObject);
var Retazec: string;
    i: integer;
    Pocet: array ['A'..'Z'] of integer;
    Znak: char;
begin
Retazec:= UpperCase(Edit1.Text);              //všetky písmená angl.abecedy zmeníme na veľké
for Znak:='A' to 'Z' do Pocet[Znak]:= 0;      //vynulovanie počtu jednotlivých písmen
//cyklus „berie“ z reťazca znak po znaku a ak je písmenom angl.abecedy, zvýši počet o jeden

for i:= 1 to Length(Retazec) do if Retazec[i] in ['A'..'Z'] then inc(Pocet[Retazec[i]]);

for Znak:='A' to 'Z' do                       //vypísanie nenulových počtov jednotlivých písmen
        if Pocet[Znak] > 0 then Memo1.Lines.Add( Format( '%s     %3d', [ Znak, Pocet[Znak] ] ) );
end;
```

Ak by nás zaujímali počty jednotlivých znakov od medzery (kód 32) až po 256. znak tabuľky ASCII, museli by sme statické pole deklarovať var Pocet: array [chr(32)..chr(255)] of integer; V procedúre nižšie je reťazec uložený v Edit1.Text.

```
procedure TForm1.btZistiClick(Sender: TObject);
var Pocet: array[chr(32)..chr(255)] of integer;
    i: integer;
    Znak: char;
begin
for Znak:= chr(32) to chr(255) do Pocet[Znak]:= 0;
```

```
for i:= 1 to Length(Edit1.Text) do
begin
        Znak:= Edit1.Text[i];                    // i-ty znak z reťazca Edit1.Text
        inc( Pocet[Znak] );
end;
for Znak:= chr(32) to chr(255) do
        if Pocet[Znak] > 0 then Memo1.Lines.Add(Znak + '    ' + IntToStr(Pocet[Znak]));
end;
```

## Príklad 14.2*

Sú dané dva reťazce (bez diakritiky, malé a veľké písmená sa nerozlišujú). Určte, či je jeden permutáciou druhého, t.j. či je možné získať druhý len zmenou poradia znakov v prvom reťazci.

*Analýza*

Reťazce abeceda a adabece sú zrejme permutáciami. Nutnou podmienkou je, aby oba reťazce mali rovnaký počet znakov. Snáď prvým algoritmom, ktorý nám napadne, je utriediť oba reťazce a zistiť, či sa rovnajú. Výrazne časovo efektívnejší je však algoritmus využívajúci pole Pocet: array [chr(32)..chr(127)] of byte; Ak je jeden reťazec permutáciou druhého, musia obsahovať rovnaké počty identických znakov. Algoritmus možno ešte zefektívniť použitím len jedného poľa Pocet. Tu je riešenie:

```
procedure TForm1.btPermutaciaClick(Sender: TObject);
var Retazec1, Retazec2: string;
   i: integer;
   znak: char;
   Pocet: array [chr(32)..chr(127)] of byte;
   Vynulovane: boolean;
begin
Retazec1 := UpperCase(Edit1.Text);
Retazec2 := UpperCase(Edit2.Text);
if length(Retazec1) <> length(Retazec2)
then Memo1.Lines.Add( 'Permutácia neexistuje (rôzna dĺžka reťazcov!)' )
else begin
        //vynulovanie poľa Pocet
        for znak := chr(32) to chr(127) do Pocet[znak] := 0;
        //naplnenie poľa Pocet počtami jednotlivých znakov z prvého reťazca
        for i := 1 to length(Retazec1) do inc( Pocet[ Retazec1[i] ] );
        //odčítanie počtov nájdených znakov v druhom reťazci z poľa Pocet
        for i := 1 to length(Retazec2) do dec( Pocet[ Retazec2[i] ] );
        //predpokladáme, že počty znakov v poli Pocet sú nulové
        Vynulovane := True;
        //zisťovanie, či sa nájde nenulový počet niektorého znaku, mohol by byť aj cyklus while
        for znak := chr(32) to chr(127) do Vynulovane := Vynulovane and (Pocet[znak] = 0);
        //vypísanie výsledku
        if Vynulovane then  Memo1.Lines.Add( 'Permutácia existuje!' )
        else Memo1.Lines.Add( 'Permutácia neexistuje!' )
     end;
end;
```

### Typizované konštanty

Ak sa hodnota objektu počas behu programu nemení, používame typizované konštanty.
Typizované konštanty definujeme:

const *IdentifikatorKonštanty* : *typ = hodnota*;

Pr.: const Meno: string = 'Peter';

## Konštantné pole

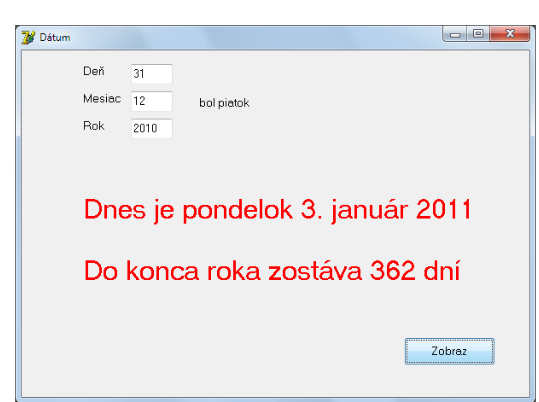
je typizovaná konštanta typu pole, ktorej hodnoty sú uvedené v okrúhlych zátvorkách, oddelené čiarkami. Konštantné pole používame, ak nám vyhovuje uloženie konštánt do poľa, vďaka čomu máme prístup k jednotlivým hodnotám cez index poľa.

Napríklad

```
const   Mesiac: array [1..12] of integer = (31,28,31,30,31,30,31,31,30,31,30,31);
        //obsahuje počty dní v jednotlivých mesiacoch nepriestupného roka, index určuje mesiac
        Den: array [1..7] of string = ('pondelok','utorok','streda','štvrtok','piatok','sobota', 'nedeľa');
        //obsahuje názvy dní v týždni, podobne by mohlo obsahovať aj názvy mesiacov v roku
        Hodnota: array [1..15] of real = (500,200,100,50,20,10,5,2,1,0.50,0.20,0.10,0.05,0.02,0.01);
        //obsahuje nominálne hodnoty euro bankoviek a mincí
        Morse : array ['A'..'Z'] of string[6] = ( '.-' , '-...' , '-.-.' , atď., '--..' );
        //obsahuje 26 znakov Morseho abecedy, index je príslušné písmeno
        A : array [1..10] of 0..1 = (1,0,0,1,1,0,0,1,0,1);
        Farba : array [0..4] of TColor = (clWidth,clRed,clBlue,clYellow,clBlack);
```

### Príklad 15.1

Vytvorte program, ktorý vypíše (pozri vpravo)
Dnes je... (aktuálny deň a dátum)
Do konca roka zostáva ... dní
a po zadaní dátumu aj
bol/je/bude názov dňa.




*Analýza*

Využijeme viacej procedúr na prácu s dátumom.
Funkcia Date vracia aktuálny (systémový) dátum,
funkcia DateToStr konvertuje dátumový formát TDateTime na reťazec,
procedúra DecodeDate konvertuje dátum vo formáte TDateTime do premenných Rok, Mes a Den,
funkcia DayOfWeek vracia poradové číslo dňa v týždni (začína nedeľou).
Procedúry si môžete podrobne pozrieť v helpe.
Jedná sa o ukážku použitia konštantného poľa a funkcií na prácu s dátumom.

```
procedure TForm1.btZobrazClick(Sender: TObject);
const Mesiac: array [1..12] of integer = (31,28,31,30,31,30,31,31,30,31,30,31);
      cDen: array [1..7] of string = ('nedeľa','pondelok','utorok','streda','štvrtok','piatok','sobota');
var   Datum: TDateTime;
      Den, Mes, Rok: word;
      M, Uplynulo: integer;
      CisloDna: 1..7;
begin
//nastavenie formátu dátumu, napr. dddd znamená zobraz celý názov dňa
ShortDateFormat := 'dddd d. mmmmm yyyy';
```

```
Label1.Caption := 'Dnes je ' + DateToStr(Date);

//výpočet počtu dní do konca roka
DecodeDate(Date, Rok, Mes, Den);
Uplynulo:= Den;
for M:= 1 to Mes-1 do Uplynulo:= Uplynulo + Mesiac[M];
Label2.Caption := 'Do konca roka zostáva '+ IntToStr(365-Uplynulo) + ' dní';

//zobrazenie názvu dňa po zadaní dátumu aj s časovaním
Datum := EncodeDate(StrToInt(Edit3.Text), StrToInt(Edit2.Text), StrToInt(Edit1.Text));
CisloDna:= DayOfWeek(Datum);
if Datum < Date
then case CisloDna of
        1, 4, 7: Label3.Caption := 'bola ' + cDen[CisloDna];
        else Label3.Caption := 'bol ' + cDen[CisloDna]
      end
else if Datum = Date
        then Label3.Caption := 'je ' + cDen[CisloDna]
        else Label3.Caption := 'bude ' + cDen[CisloDna]
end;
```

Pri výpočte dní do konca roka sme zanedbali priestupné roky, môžete doplniť a tiež skloňovanie (deň/dni/dní).

### Príklad 15.2

Vytvorte program simulujúci tzv. mincovku. Po zadaní sumy (celé nezáporné číslo) program vypíše minimálny počet 500, 200, 100, 50, 20, 10, 5, 2 a 1 „euroviek“, potrebných na vyplatenie zadanej sumy.

*Analýza*

Celé riešenie problému je dvoch krátkych procedúrach. Funkcia Pocet vráti počet bankoviek dovezenej hodnoty pomocou funkcie div a zníži sumu o zarátanú hodnotu. Napríklad pri sume 1 523 € a hodnote 500 € vráti číslo 3 a sumu zníži na 23 €. Procedúra btSpracujClick volá funkciu Pocet postupne pre všetky nominálne hodnoty uložené v konštantnom poli Hodnota a vypisuje počty jednotlivých „euroviek“. Ostatné procedúry len zvyšujú komfort programu.

```
const PocHodnot = 9;
      Hodnota: array [1..PocHodnot] of integer = (500,200,100,50,20,10,5,2,1);
var   Suma: integer;

function Pocet(Hodnota: integer):integer;
begin
Pocet:= Suma div Hodnota;
Suma:= Suma mod Hodnota;
end;
//procedúra nedovolí vložiť do poľa komponentu Edit1 iný znak ako cifru alebo stlačiť BackSpace
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
if not ( Key in [ '0'..'9' , #8 ] ) then Key:= #0
end;
```

```
procedure TForm1.btSpracujClick(Sender: TObject);
var i: 1..PocHodnot;
begin
if Edit1.Text <> '' then
begin
        Suma:= StrToInt(Edit1.Text);
        for i:= 1 to PocHodnot do
            Memo1.Lines.Add(Format('%3d €...........%3d',[ Hodnota[i], Pocet(Hodnota[i]) ]));
end;
end;
//procedúra nastaví kurzor pri aktivácii formulára do poľa komponentu Edit1
procedure TForm1.FormActivate(Sender: TObject);
begin
Edit1.SetFocus;
end;
```

Program môžete doplniť o zadanie počtu zamestnancov a príslušného počtu súm a výpočet výslednej mincovky.

## Príklad 15.3

Reťazec zapísaný do komponentu Edit1.Text (len veľké písmená anglickej abecedy a medzery) sa má prepísať znak po znaku do Morseho kódu. Ak je znakom medzera, má sa vypísať kód celého slova a prejsť vo výstupe na nový riadok. Kódy jednotlivých písmen v slove oddeľte lomkou /.
Napríklad reťazec ABECEDA ZJEDLA DEDA sa zobrazí:

```
.- / -... / . / -.-. / . / -.. / .- /
--.. / .--- / . / -.. / .-.. / .- /
-.. / . / -.. / .- /
```

Riešenie

```
var Morse: array ['A'..'Z'] of string[4]=('.-','-...','-.-.','-..','.','..-.','--.','....','..','.---','-.-','.-..','--','-.','---','.--.','--.-','.-.','...','-','..-','...-','.--','-..-','-.--','--..');

procedure TForm1.btKodujClick(Sender: TObject);
var i: integer;
    Znak: char;
    Slovo: string;
begin
Slovo:='';
for i:=1 to length(Edit1.Text) do
begin
        Znak:= UpCase(Edit1.Text[i]);
        if Znak in ['A'..'Z']
        then Slovo:= Slovo + Morse[Znak] + ' / '
        else if Znak<>' '
             then Slovo:= Slovo + '?' + ' / '
             else begin
                       Memo1.Lines.Add(Slovo);
                       Slovo:=''
                  end;
end;
```

```
Memo1.Lines.Add(Slovo);
end;
```

## Hodnoty premenných zadané pri ich deklarácii

Ak poznáme počiatočnú hodnotu premennej už pri písaní programu, môžeme jej ju priradiť v úseku deklarácií zápisom

var *IdentifikatorPremennej* : *typ* = ***hodnota***;

Pr.: var Pole : array [1..6] of string = ('Peter' , 'Jana' , 'Adam' , 'Dalibor' , 'Zuzana' , 'Barbora');

Na rozdiel od konštantného poľa možno zmeniť hodnoty tohto poľa počas behu programu, napr. utriediť, prepísať, nemožno však napríklad pridať nový prvok - zväčšiť pole (vtedy musíme použiť dynamické pole s priradením počiatočných hodnôt).

## Otvorené pole ako parameter procedúry

Pri **statických** poliach umožňuje Delphi ako formálny parameter procedúry použiť pole bez udania veľkosti poľa, čo umožňuje vytvárať všeobecnejšie procedúry. V deklarácii formálneho parametra procedúry sa uvedie len typ poľa. Napríklad

```
procedure NulujPole (var Pole: array of integer);          //parameter Pole je otvorené pole
var i: integer;
begin
for i:= Low(Pole) to High(Pole) do Pole[i]:= 0;
end;
```

Indexy **otvorených polí** vždy začínajú 0 (nulou). Ak je statické pole deklarované napríklad od 1 (napríklad globálne var A: array [1..100] of integer;) parameter Pole si hodnotu Pole[0] vymyslí. Preto pri otvorených poliach vždy používajte funkciu Low(Pole).

*Pamätaj*!
Parameter typu pole sa vždy snažíme nahradzovať odkazom (referenciou) a nie hodnotou, pri ktorej sa vytvára nová kópia poľa zaberajúca ďalšie miesto v pamäti.
Tak, ako nebolo v Pascale dovolené deklarovať statické pole pri písaní formálneho parametra, nie je to dovolené ani v Delphi, t.j. deklarácia napríklad

```
procedure NulujPole (var A: array [1..100] of integer);
```

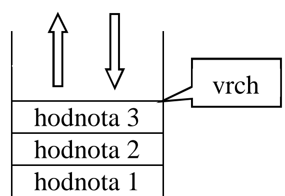
**nie je dovolená**.
Pri statickom poli je dovolené v deklarácii formálneho parametra použiť len **meno** definovaného typu pole, napríklad

globálne
```
type tPole = array [1..100] of integer;          //tPole je meno typu pole
var Pole: tPole;
```

v procedúre
```
procedure NulujPole (var A:  tPole);
```

*Poznámky*:

# Zásobník




je dynamická dátová štruktúra, na ktorej sú dovolené len operácie:

- vytvoriť prázdny zásobník
- pridať prvok na vrch zásobníka
- odobrať prvok z vrchu zásobníka

Zásobník nám umožňuje dočasne uložiť údaje a spracovať ich až neskôr - v opačnom poradí, ako boli vložené. Zásobník sa označuje aj ako štruktúra LIFO (Last In - First Out, posledný dnu – prvý von). Zásobník, ako typ pamäte, sa používa na odkladanie návratových adries pred odchodom do podprogramov, na odkladanie hodnôt lokálnych premenných, na odkladanie „vonkajších častí" vyhodnocovaných aritmetických výrazov pri vnáraní do nich, pri metóde „rozdeľ a panuj", pri prehľadávaní do hĺbky a pod.

### Príklad 16.1

Simulujte operácie zásobníka na jednorozmernom statickom poli.

*Analýza*

Uvedomme si, že statické pole má vyhradenú pamäť hneď od spustenia programu, teda zásobník hneď existuje (obsahuje náhodné hodnoty). Hodnotu premennej Vrch môžeme využiť na „orientáciu" v poli-zásobníku. Ak má hodnotu nula, pole-zásobník je prázdny, ak má hodnotu 1, zásobník obsahuje jednu hodnotu (v poli na indexe 1), ak má hodnotu veľkosti poľa, zásobník je plný. V premennej Vrch je uložený index ostatne vloženej hodnoty a len s ním je dovolené pracovať – vybrať hodnotu z vrchu, vložiť hodnotu „nad" vrch. Prechádzať zásobníkom nie je možné.

Pre jednoduchosť uvádzame rovno „kompletný program" doplnený komentárom. Možný vizuálny návrh je vpravo.

```
const  MAXPP = 10;                           //maximálny počet prvkov v zásobníku 10
var    Zas: array [1..MAXPP] of string;      //deklarácia zásobníka, obsahom budú reťazce
       Vrch: 0..MAXPP;                       //Vrch, údajový typ interval, môže byť aj typ integer

procedure VytvorPrazdny;                             //vytvorenie prázdneho zásobníka
begin
Vrch:= 0;                                            //zásobník neobsahuje hodnoty
end;
procedure TForm1.btVytvorClick(Sender: TObject);     //volanie procedúry VytvorPrazdny
begin
VytvorPrazdny;
end;

procedure Vloz(Vlozit: string);                      //vloženie hodnoty do zásobníka
begin
if Vrch = MAXPP                                      //Je zásobník plný?
then ShowMessage('Zásobník je plný.')                //áno - zobraz správu Zásobník je plný
else begin                                           //nie - môžeme vložiť hodnotu
     inc(Vrch);                                //vložiť „nad" vrch znamená zvýšiť index o 1
     Zas[Vrch]:= Vlozit;                       //vloženie hodnoty do poľa
     end;
end;
```

```
procedure TForm1.btVlozClick(Sender: TObject);      //volanie procedúry Vloz
var Hodnota: string;
begin
Hodnota:= InputBox('zasobnik','hodnota','');
Vloz(Hodnota);
end;

function Vyber(var Vybrat: string):boolean;         //vybratie hodnoty zo zásobníka ak nie je prázdny
begin
if Vrch = 0                                          //Je prázdny?
then Vyber:= False                                   //áno – funkcia Vyber vráti hodnotu False
else begin                                           //nie - možno vybrať hodnotu z vrchu zásobníka
        Vyber:= True;                                //funkcia Vyber vráti hodnotu True
        Vybrat:= Zas[Vrch];                          //cez parameter Vybrat sa vyvezie hodnota
        dec(Vrch);                                   //hodnota v premennej Vrch sa zníži o 1
   end;
end;

procedure TForm1.btVyberClick(Sender: TObject);     //volanie funkcie Vyber
var Hodnota: string;
begin
if Vyber(Hodnota)                                    //Ak Vyber má logickú hodnotu
then Memo1.Lines.Add ( 'Vybral som ' + Hodnota )    //True – vypíše sa odobratá hodnota z vrchu
else Memo1.Lines.Add ( 'Zasobnik je prazdny.' );    //False – nebolo čo odobrať, zásobník bol prázdny
end;

initialization                                       //vykonať po spustení programu
Vrch:= 0;                                            // „vytvorí“ prázdny zásobník
end.
```

### Príklad 16.2

Pre zaujímavosť uvedieme aj zásobník modelovaný jednorozmerným dynamickým poľom. Pre takto vytvorený zásobník teoreticky nemôže nastať situácia, aby bol plný. Tiež stráca význam použitie premennej Vrch resp. jej funkciu môže zastúpiť funkcia High(Zas) a hodnota nil, ktorá pri dynamických poliach znamená, že pole je prázdne.

„Kompletný“ program, tentoraz bez komentára:

```
var Zas: array of string;
procedure TForm1.btVytvorPrazdnyClick(Sender: TObject);
begin
Zas:= nil;
end;

procedure TForm1.btOdoberClick(Sender: TObject);
begin
if Zas = nil
then Memo1.Lines.Add( 'Zásobník je prázdny.' )
else begin
        Memo1.Lines.Add( 'Odobral som hodnotu ' + Zas[ High(Zas) ] );
        if High(Zas) > 0
```

```
        then Zas:= copy( Zas , 0 , length(Zas)-1 )
        else Zas:= nil
    end;
end;

procedure TForm1.btVlozClick(Sender: TObject);
begin
setlength( Zas , length(Zas)+1 );
Zas[ High(Zas) ]:= InputBox( 'Zásobník' , 'Vložiť hodnotu' , '' );
end;

initialization
Zas:= nil;
end.
```

**Príklad 16.3**

Vytvorte efektívny program, ktorý umožní vkladať hodnoty do dvoch zásobníkov modelovaných statickými poliami. Program však obsahuje len jedno tlačidlo Odobrať, ktoré prednostne odoberá z prvého zásobníka a len ak je prázdny, odoberá z druhého zásobníka (ak nie je prázdny).

Možné riešenie:

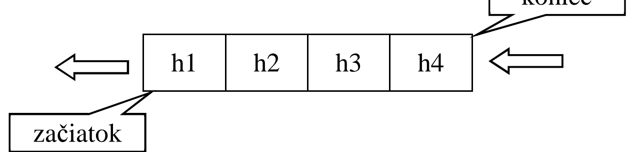
```
implementation
{$R *.dfm}

const MAXPP = 5;
type  Zasobnik = array [1..MAXPP] of string;
var   Zas1, Zas2: Zasobnik;
      Vrch1, Vrch2: integer;

procedure Vloz(var Kam: Zasobnik; var Vrch: integer; Hodnota: string);
begin
if Vrch = MAXPP
then ShowMessage('Zásobník je plný!')
else begin
       inc(Vrch);
       Kam[Vrch] := Hodnota;
     end
end;

procedure TForm1.btVlozZ1Click(Sender: TObject);
var Vlozit: string;
begin
Vlozit := InputBox('Dva zásobníky','Vložiť','');
Vloz(Zas1,Vrch1,Vlozit);
end;

procedure TForm1.btVlozZ2Click(Sender: TObject);
var Vlozit: string;
begin
Vlozit := InputBox('Dva zásobníky','Vložiť','');
Vloz(Zas2,Vrch2,Vlozit);
```

```
end;

function Odober(var Hodnota: string):boolean;
begin
if Vrch1 = 0
then if Vrch2 = 0
    then begin
             ShowMessage('Oba zásobníky sú prázdne!');
            Odober:= false;
           end
    else begin
           Odober:= true;
           Hodnota := Zas2[Vrch2];
           dec(Vrch2)
         end
else begin
        Odober:= true;
        Hodnota := Zas1[Vrch1];
        dec(Vrch1)
     end
end;

procedure TForm1.btOdoberClick(Sender: TObject);
var Odobral: string;
begin
if Odober(Odobral)
then Memo1.Lines.Add('Odobral som ' + Odobral)
end;

end.
```

# Rad

je dynamická dátová štruktúra, na ktorej sú dovolené len operácie:

- vytvoriť prázdny rad
- pridať prvok na koniec radu
- odobrať prvok zo začiatku radu




Rad nám umožňuje dočasne uložiť údaje a spracovať ich neskôr v poradí, ako boli vložené. Označuje aj ako štruktúra FIFO (Firs In - First Out, prvý dnu – prvý von). Rad, ako typ pamäte, sa používa na radenie úloh prichádzajúcich s požiadavkou na tlač (s rovnakou prioritou), pri prehľadávaní do šírky a pod. Český názov pre rad je fronta.

Hodnoty radu môžeme odkladať aj do jednorozmerného poľa. Rad pomocou jednorozmerného poľa môže byť modelovaný viacerými spôsobmi, môžeme použiť dynamické alebo statické pole, ktoré môže byť ešte akési uzavreté a pod. Problémom pri použití statického poľa je skutočnosť, že rad môže byť prakticky prázdny (odoberieme napríklad stý prvok v maximálne sto prvkovom poli) a napriek tomu nemožno vložiť ďalší prvok do radu, pretože sme na konci poľa. Aby sme sa vyhli takejto situácii, je najjednoduchšie vytvoriť uzavreté pole, v ktorom po maximálnom indexe nasleduje opäť index 1. Samozrejme, ani toto pole nemôže mať viacej prvkov, ako je veľkosť poľa.

### Príklad 17.1

Simulujte operácie radu na jednorozmernom uzavretom statickom poli.

*Analýza*

Ako pri zásobníku si treba uvedomiť, že pole bude existovať hneď po spustení programu. Ktoré hodnoty poľa sú hodnotami radu budeme sledovať a určovať pomocou premenných začiatok radu (Zac) a koniec radu (Kon). Použijeme premenné

- Zac, v ktorej bude uložený index poľa prvku, ktorý sa má pri požiadavke odobrať odobrať z poľa
- Kon, v ktorej bude uložený index poľa prvku, ktorý bol posledný vložený do radu
- Pocet, v ktorej bude uložený počet prvkov v rade, ak nadobudne hodnotu konštanty MAXPP (maximálny počet prvkov poľa), rad je plný.

Pojem uzavreté pole nie je oficiálny, preto si vysvetlíme, čo máme na mysli. Nech pole nie je plné, t.j. Pocet < MAXPP. Keď v premennej Kon bude hodnota MAXPP (hodnota maximálneho indexu poľa) a chceme do radu pridať ďalšiu hodnotu, musí byť voľné miesto na začiatku poľa a preto pokračovať vo vkladaní môžeme na začiatku poľa, t.j na indexe 1. Tým sme pole „uzavreli do kruhu“ (po indexe MAXPP nasleduje index 1). Či je rad plný, musíme sledovať pomocou premennej Pocet.
Keď nadobudne hodnotu MAXPP, rad je plný.
Ak ste všetko správne pochopili, vyskúšajte sa na obrázku, v ktorom je znázornený rad, do ktorého boli postupne vložené hodnoty 1, 2, 3, 4, 5, 6 a 7; medzitým aj vybrané hodnoty 1, 2 a 3. MAXPP má hodnotu 5, Pocet hodnotu 4.

| Rad | | Kon = 2 | | Zac = 4 | |
|---|---|---|---|---|---|
| Hodnoty v rade | **6** | **7** | **-** | **4** | **5** |
| Pole (indexy) | 1. | 2. | 3. | 4. | 5.<br>MAXPP |

```
const MAXPP = 10;
var Rad: array [1..MAXPP] of string;
    Zac, Kon, Pocet: integer;
procedure TForm1.btVytvorPrazdnyClick(Sender: TObject);
```

```
begin
Zac:= 1; Kon:= MAXPP; Pocet:= 0;                  //všimnite si nastavenie Zac a Kon
end;

procedure Vloz (Vlozit: string);                  //procedúru Vlozit možno volať, len ak je
begin                                             //v poli voľné miesto, t.j. Pocet < MAXPP
if Kon = MaxPP                                    //ak sme na konci poľa
then Kon:= 1
else inc(Kon);
Rad[Kon]:= Vlozit;                                //v rade sa vkladá na koniec
inc(Pocet)                                        //vložili sme hodnotu do radu
end;

procedure TForm1.btVlozClick(Sender: TObject);    //volanie procedúry Vloz
var Hodnota: string;
begin
if Pocet = MAXPP                                  //ak je rad plný
then ShowMessage( 'Rad je plný!' )
else begin
        Hodnota:= InputBox( 'Rad' , 'Vložiť' , '' );
        Vloz(Hodnota)
     end;
end;

function Odober: string;                          //vráti odobratú hodnotu
begin
Odober:= Rad[Zac];                                //v rade sa odoberá zo začiatku
if Zac = MAXPP                                    //ak sme na konci poľa
then Zac:= 1
else inc(Zac);
dec(Pocet)                                        //odobrali sme hodnotu z radu
end;

procedure TForm1.btOdoberClick(Sender: TObject);  //použitie funkcie Odober
begin
if Pocet > 0                                      //ak je čo odobrať
then Memo1.Lines.Add( 'Odobral som hodnotu ' + Odober )
else Memo1.Lines.Add( 'Rad je prázdny!' )
end;

initialization
Zac:= 1; Kon:= MAXPP; Pocet:= 0;                  //nastaví sa po spustení programu
end.
```

Môžete si vytvoriť program simulujúci rad pomocou dynamického poľa, začiatok radu bude mať vždy hodnotu 0, koniec hodnotu High(Rad). Pridať hodnotu na koniec radu-poľa je zrejmé a odobrať znamená použiť príkaz Rad:= copy (Rad, 1, length(Rad)-1); sledovať počet prvkov poľa nemá zmysel.

Môžete si vytvoriť aj rad pomocou jednorozmerného statického poľa v ktorom, po dosiahnutí konca poľa, už nebudú môcť byť vkladané hodnoty do radu.

**Príklad 17.2**

Zákazník príde do čakárne a klikne na tlačidlo Príchod. Vytvorte program, ktorý umožní

1. evidovať čas príchodu zákazníka do čakárne,
2. vyradenie z radu po obslúžení, prípadne vypísať čas strávený v zariadení,
3. vypísať časy príchodov čakajúcich v rade.

Obsluha zariadenia je schopná denne obslúžiť najviac 5 zákazníkov.

*Analýza*

Keďže je obsluha zariadenia schopná obslúžiť najviac 5 zákazníkov, je vhodné použiť statické pole s konštantou MAPP = 5. Po kliknutí na tlačidlo Príchod treba zapísať na koniec radu čas príchodu zákazníka. Do poľa môžeme ukladať zadaný čas ako typ TDateTime. Rozhodli sme sa, po kliknutí na tlačidlo Príchod, zapísať do poľa aktuálny (systémový) čas, ktorý obsahuje premenná Time a je typu TDateTime. Pre jeho zobrazenie, napríklad v Memo, treba použiť konverznú funkciu TimeToStr.

Možná alternatíva programu

```
const MAXPP = 5;
var    Rad: array[1..MAXPP] of TDateTime;
       Zac, Kon: integer;

procedure TForm1.btPrisielZakaznikClick(Sender: TObject);
begin
if Kon = MAXPP
then Showmessage('Dnes Vás už neobslúžime.')
else begin
          inc(Kon);
          Rad[Kon]:= Time;
          Memo1.Lines.Add(TimeToStr(Rad[Kon]))
      end;
end;
procedure TForm1.btObsluzenyClick(Sender: TObject);
begin
if Zac > Kon
then Showmessage('Nik nečaká.')
else begin
          Memo1.Lines.Add('Obslúženie trvalo ' + TimetoStr(Time-Rad[Zac]));
          inc (zac);
      end;
end;
procedure TForm1.btCakajuClick(Sender: TObject);
var i:integer;
begin
if Zac > Kon
then Memo1.Lines.Add('Nik nečaká.')
else begin
          Memo1.Lines.Add('Čakajú na obslúženie:');
          for i:= Zac to Kon do Memo1.Lines.Add(TimeToStr(Rad[i]));
      end;
end;
```

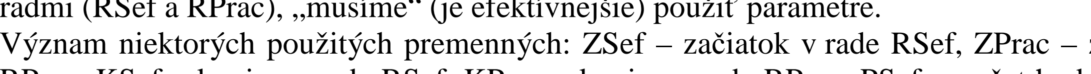
```
initialization
Zac:=1; Kon:=0;
end.
```

## Príklad 17.3

Vytvorte program s dvoma radmi – šéfov a pracovníkov. Tlačidlo Vlož do radu šéf vloží zadaný reťazec do šéfovho radu, tlačidlo Vlož do radu pracovník vloží zadaný reťazec do pracovníkovho radu (na vloženie do radov môžete použiť len jednu procedúru). Po kliknutí na tlačidlo Odober sa prednostne odoberá zo šéfovho radu, len ak ten je prázdny, začne sa odoberať z pracovníkovho radu.

*Analýza*

Môžeme použiť vzorový program z príkladu 17.1. Keďže však procedúry Vloz a Odober majú pracovať s dvoma radmi (RSef a RPrac), „musíme" (je efektívnejšie) použiť parametre.



Význam niektorých použitých premenných: ZSef – začiatok v rade RSef, ZPrac – začiatok v rade RPrac, KSef – koniec v rade RSef, KPrac – koniec v rade RPrac, PSef – počet hodnôt v rade PSef a PPrac – počet hodnôt v rade RPrac.

```
const MAXPP = 5;                                  //maximálny počet hodnôt v rade
type  tRad = array [1..MAXPP] of string;          //definovanie typu tRad ako stat.poľa reťazcov
var   RSef, RPrac: tRad;                          //deklarovanie dvoch premenných typu tRad
      ZSef, KSef, PSef, ZPrac, KPrac, PPrac: integer;

procedure Vloz (var Rad: tRad; var Kon, Pocet: integer; X: string);     //univerzálna procedúra Vloz
begin
if Kon = MaxPP
then Kon:= 1
else inc(Kon);
Rad[Kon]:= X;
inc(Pocet)
end;

function Odober (var Rad: tRad; var Zac, Pocet: integer): string;       //univerzálna funkcia Odober
begin
Odober:= Rad[Zac];
if Zac = MAXPP then Zac:= 1
else inc(Zac);
dec(Pocet)
end;

procedure TForm1.btVlozDoSefClick(Sender: TObject);                     //vloženie hodnoty do radu RSef
var Hodnota: string;
begin
if PSef = MAXPP
then ShowMessage( 'Šéfov rad je plný!' )
else begin
        Hodnota:= InputBox( 'Rad' , 'Vložiť' , '' );
        Vloz(RSef, KSef, PSef, Hodnota)
     end;
end;
```

```
procedure TForm1.btVlozDoPracClick(Sender: TObject);        //vloženie hodnoty do radu RPrac
var Hodnota: string;
begin
if PPrac = MAXPP
then ShowMessage('Rad pracovníkov je plný!')
else begin
        Hodnota:= InputBox('Rad','Vložiť','');
        Vloz(RPrac, KPrac, PPrac, Hodnota)
    end;
end;

procedure TForm1.btOdoberClick(Sender: TObject);            //odobratie z radu
var Hodnota: string;
begin
if PSef > 0                                                  //prednostne z RSef
then Memo1.Lines.Add( 'Odobral som hodnotu ' + Odober(RSef, ZSef, PSef) )
else if PPrac > 0                                            //až potom z RPrac
    then Memo1.Lines.Add( 'Odobral som hodnotu ' + Odober(RPrac, ZPrac, PPrac) )
    else Memo1.Lines.Add( 'Oba rady sú prázdne!' )
end;

initialization                                               //nastavenie počiatočných hodnôt
ZSef:= 1;  KSef:= MAXPP;  PSef:= 0;
ZPrac:= 1; KPrac:= MAXPP; PPrac:= 0;
end.
```

### Príklad 17.4

Vytvorte program simulujúci prácu dvoch radov R1 a R2 modelovaných uzavretými poliami (MAXPP = 5). Do radov možno vkladať celočíselné hodnoty. Po kliknutí na tlačidlo Odober sa odoberie väčšia z hodnôt na začiatkoch radov R1 a R2.

*Analýza*

Program je do značnej miery analógiou predchádzajúceho príkladu.

Funkcia Pozri vracia hodnotu z volaného radu bez toho, aby ju odobrala z radu.

Pri riešení problému, z ktorého radu treba odobrať hodnotu, môže nastať až päť možností.

Neštandardne sme vytvorili aj procedúru na vypísanie obsahov radov, ktorej algoritmus nie je triviálny.

```
implementation
{$R *.dfm}

const MAXPP = 5;
type  tRad = array [1..MAXPP] of integer;
var   R1, R2: tRad;
      Z1, K1, Po1, Z2, K2, Po2: integer;

procedure Vloz (var Rad: tRad; var Kon, Pocet: integer; X: integer);
begin
if Kon = MAXPP
then Kon:= 1
else inc(Kon);
```

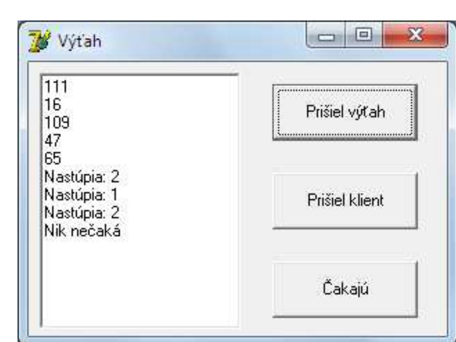
```
Rad[Kon]:= X;
inc(Pocet)
end;

function Odober (var Rad: tRad; var Zac, Pocet: integer): integer;
begin
Odober:= Rad[Zac];
if Zac = MAXPP
then Zac:= 1
else inc(Zac);
dec(Pocet)
end;

procedure TForm1.btVlozDoPrvehoClick(Sender: TObject);
var Hodnota: integer;
begin
if Po1 = MAXPP
then ShowMessage('Prvý rad je plný!')
else begin
        Hodnota:= StrToInt(InputBox('Rad','Vložiť',''));
        Vloz(R1, K1, Po1, Hodnota)
     end;
end;

procedure TForm1.btVlozDoDruhehoClick(Sender: TObject);
var Hodnota: integer;
begin
if Po2 = MAXPP
then ShowMessage('Druhý rad je plný!')
else begin
        Hodnota:= StrToInt(InputBox('Rad','Vložiť',''));
        Vloz(R2, K2, Po2, Hodnota)
     end;
end;

function Pozri(Rad: tRad; Zac: integer): integer;
begin
Result:= Rad[Zac];
end;

procedure TForm1.btOdoberClick(Sender: TObject);
begin
if (Po1 = 0) and (Po2 = 0)
then Memo1.Lines.Add('Oba rady sú prázdne!')
else if Po1 = 0
     then Memo1.Lines.Add('Odobral som hodnotu ' + IntToStr(Odober(R2, Z2, Po2)))
     else if Po2 = 0
          then Memo1.Lines.Add('Odobral som hodnotu ' + IntToStr(Odober(R1, Z1, Po1)))
          else if Pozri(R1, Z1) > Pozri(R2, Z2)
               then Memo1.Lines.Add('Odobral som hodnotu '+ IntToStr(Odober(R1, Z1, Po1)))
               else Memo1.Lines.Add('Odobral som hodnotu ' + IntToStr(Odober(R2, Z2, Po2)))
end;
```

```
procedure TForm1.btVypisRadyClick(Sender: TObject);
var i: integer;
    Riadok: string;
begin
Riadok:= 'Prvý rad:';
if Po1 = 0 then Memo1.Lines.Add(Riadok + ' je prázdny')
else begin
        if Z1 <= K1
        then for i:= Z1 to K1 do Riadok:= Riadok + ' ' + IntToStr(R1[i])
        else begin
                for i:= Z1 to MAXPP do Riadok:= Riadok + ' ' + IntToStr(R1[i]);
                for i:= 1 to K1 do Riadok:= Riadok + ' ' + IntToStr(R1[i]);
            end;
        Memo1.Lines.Add(Riadok);
    end;

Riadok:= 'Druhý rad:';
if Po2 = 0 then Memo1.Lines.Add(Riadok + ' je prázdny')
else begin
        if Z2 <= K2
        then for i:= Z2 to K2 do Riadok:= Riadok + ' ' + IntToStr(R2[i])
        else begin
                for i:= Z2 to MAXPP do Riadok:= Riadok + ' ' + IntToStr(R2[i]);
                for i:= 1 to K2 do Riadok:= Riadok + ' ' + IntToStr(R2[i]);
            end;
        Memo1.Lines.Add(Riadok);
    end;
end;

initialization
 Z1:= 1; K1:= MAXPP; Po1:= 0;
 Z2:= 1; K2:= MAXPP; Po2:= 0;
end.
```

## Príklad 17.5

Pred výťahom je elektronická tlaková váha, ktorá automaticky určí hmotnosť došlého klienta (nech program náhodne vygeneruje kliknutím na tlačidlo Prišiel klient celé číslo od 15 do 150 kg a zaradí do radu). Po kliknutí na tlačidlo Prišiel výťah program oznámi, koľkí z čakajúcich môžu nastúpiť (súčet ich hmotností nesmie prekročiť nosnosť výťahu). Ak je v rade viac ako 10 osôb, nech program oznámi: Použite schodište.

Riešenie bez komentára




```
const MAXPOCET = 10;
      NOSNOST = 150;
var   Rad: array [1..MAXPOCET] of integer;
      Zac, Kon, Pocet: integer;

procedure TForm1.btPrisielKlientClick(Sender: TObject);
begin
if Pocet = MAXPOCET
then ShowMessage( 'Použite schodište.' )
else begin
```

```
        if Kon = MAXPOCET then Kon:= 1
        else inc(Kon);
        Rad[Kon]:= random(136) + 15;
        inc(Pocet);
     end;
end;

function Odober: integer;
begin
Odober:= Rad[Zac];
if Zac = MAXPOCET then Zac:= 1
else inc(Zac);
dec(Pocet);
end;

function PrvyVRade: integer;
begin
PrvyVRade:= Rad[Zac];
end;

procedure TForm1.btPrisielVytahClick(Sender: TObject);
var Sucet, Nastupia: integer;
begin
if Pocet = 0 then Memo1.Lines.Add( 'Nik nečaká' )
else begin
        Sucet:= PrvyVRade; Nastupia:= 0;
        while (Sucet <= NOSNOST) and (Pocet > 0) do
        begin
                Odober;
                inc(Nastupia);
                Sucet:= Sucet + PrvyVRade;
        end;
        Memo1.Lines.Add( 'Nastúpia: ' + IntToStr(Nastupia) );
     end;
end;

procedure TForm1.btCakajuClick(Sender: TObject);
var i: integer;
begin
Memo1.Clear;
if Pocet>0
then if Zac<=Kon
        then for i:= Zac to Kon do Memo1.Lines.Add( IntToStr(Rad[i]) )
        else begin
                for i:= Zac to MAXPOCET do Memo1.Lines.Add( IntToStr(Rad[i]) );
                for i:= 1 to Kon do Memo1.Lines.Add( IntToStr(Rad[i]) )
            end;                // pre if Pocet>0 možno dopísať vetvu else s oznamom 'Nik nečaká'
end;

initialization
randomize;
Zac:= 1; Kon:= MAXPOCET; Pocet:= 0;
end.
```

# Textový súbor

Slovo súbor sa používa vo viacerých významoch. Z pohľadu operačného systému ide o skupinu údajov s vlastným názvom (meno súboru . prípona naznačujúca obsah súboru) uloženú na pamäťovom médiu (pevný disk, CD, DVD,...). Údaje sú na médiu uložené **sekvenčne** (údaj za údajom). Z pohľadu programovacích jazykov je súbor údajový typ, ktorého premenné nám umožňujú pracovať s údajmi uloženými v súboroch na vonkajších pamätiach. Fyzický súbor existuje na pamäťovom médiu a má svoje **fyzické meno súboru.** V programe, v ktorom s ním pracujeme, má súbor **logické meno súboru**, tak ako každá iná premenná používaná v programe. Logické meno súboru treba uviesť v deklaráciách premenných, napr. `var F: TextFile;`

Pre človeka je bežné písať texty zložené zo znakov a členené na riadky. Súbor s takýmto obsahom nazývame **textový súbor** a jeho výhodou je, že sa dá ľahko editovať (vytvoriť, zobraziť aj upraviť). Pre ilustráciu sme zobrazili dvojriadkový textový súbor s hodnotami:

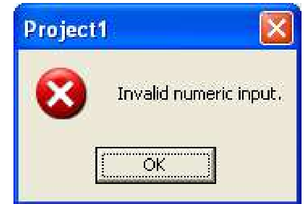
Juro
-1.2  34  7                    // tri čísla oddelené medzerami

| J | u | r | o | na nový riadok | - | 1 | . | 2 |  | 3 | 4 |  | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Čítacia hlava → dovolený smer jej postupu

Kľúčom k pochopeniu práce s textovým súborom je vedieť niečo o tom, ako sa do premenných rôznych typov načítavajú z textových súborov hodnoty. Najjednoduchšie je načítanie **znaku**. Do premennej typu char sa načíta jeden znak z pozície čítacej hlavy, potom sa čítacia hlava presunie na ďalší znak. Pri čítaní **reťazca** sa do premennej typu string uloží reťazec z aktuálneho riadka. Ak v deklarácii bola stanovená dĺžka reťazca (`var Retazec: string[` *počet znakov reťazca* `]`), načíta sa len zadaný počet znakov. Ak dĺžka reťazca nebola obmedzená, načítajú sa do premennej všetky znaky až do konca riadka! Najzaujímavejšie je plnenie premenných niektorého z číselných typov (predovšetkým **integer** a **real**). Platí:

- čítacia hlava ignoruje všetky vložené medzery, tabulátory a koniec riadka a zastaví sa až na prvom „nebielom" znaku (číslici, písmene, interpunkčnom znamienku a pod.). Ak takýmto znakom nie je znak, ktorým by mohol začínať zápis čísla očakávaného typu (iné ako znamienko plus alebo mínus, cifra, desatinná bodka), zobrazí sa chybové hlásenie (obrázok vpravo).




- po načítaní prvého „dobrého" znaku príslušnej číselnej hodnoty pokračuje čítanie, až kým čítacia hlava nenarazí na prvý znak, ktorý už nemôže zodpovedať zápisu čísla očakávaného typu (najčastejšie medzera).
- „dobré" znaky sú uložené ako platná hodnota premennej zadaného číselného typu.

Poznámka
Hodnoty typu boolean nie je možné čítať zo vstupného súboru.

Keď si už vieme predstaviť textový súbor, môžeme si niečo povedať o deklarácii premenných typu textový súbor. Typ textový súbor má jednoduché meno `TextFile`. Mohli by sme ho charakterizovať ako **štruktúrovaný údajový typ zložený z teoreticky neobmedzeného počtu údajov** (prakticky je počet údajov v súbore obmedzený kapacitou vonkajšej pamäte, na ktorú chceme údaje uložiť). Pre textový súbor je charakteristické členenie na riadky (aj keď nemusí byť nevyhnutne členený na riadky). Textový súbor nie je totožný so súborom, ktorého položky sú typu char, a teda sa spracúvajú znak po znaku. V textovom súbore sú čítané údaje automaticky konvertované na typy uvedené v deklarácii. Napríklad, ak X je typu integer, tak program načítanú skupinu znakov automaticky konvertuje na celé číslo!

Deklarácia premennej typu textový súbor má tvar

var *MenoPremennej* : TextFile;

V nasledujúcich ukážkach budeme používať deklaráciu

var F : TextFile; // ak F je globálna premenná, môžeme použiť deklaráciu var F: Text;

Automaticky pri deklarácii premennej F vzniká aj tzv. prístupová premenná F^, a jej úlohu možno prirovnať k úlohe čítacej hlavy použitej vyššie. V každom okamihu spracovania súboru je prístupný len jeden znak súboru, na ktorý „ukazuje" prístupová premenná súboru. Aj prístupová premenná sa môže posúvať len od začiatku súboru smerom k jeho koncu (štandardne znak po znaku).

Deklaráciou premennej typu súbor vznikne len abstraktný vonkajší súbor (chýba meno súboru na disku a jeho umiestnenie). K spojeniu mena logického súboru s menom súboru na vonkajšom pamäťovom médiu dôjde príkazom AssignFile ( *MenoLogickéhoSúboru* , *MenoFyzickéhoSúboru* ), kde *MenoLogickéhoSúboru* je premenná typu TextFile a *MenoFyzickéhoSúboru* je reťazec obsahujúci názov súboru na disku, prípadne doplnený aj o cestu k súboru, prípadne to je premenná typu string.

Napríklad: assignfile ( F, ´mena.txt´); assignfile(SUB,´C:\Projekty\Sutaz\skoky.txt´);
assignfile (F, InputBox( ´Vstup´ , ´Meno súboru na disku: ´ , ´´ )); assignfile(G, MenoSuboru);

Príkaz AssignFile nezisťuje, či v zadanom priečinku existuje súbor so zadaným menom!

Po príkaze AssignFile musí nasledovať **otvorenie** súboru. Súbor možno otvoriť na čítanie alebo na zápis, pričom pri zápise rozlišujeme zápis do nového súboru, alebo pripísanie údaja na koniec už existujúceho súboru za posledný údaj v súbore.

Ak chceme zo súboru čítať údaje, otvoríme ho príkazom Reset (F). Súbor musí fyzicky existovať v aktuálnom priečinku alebo v zadanej ceste! Prístupová premenná F^ sa nastaví na začiatok súboru.

Ak chceme vytvoriť nový súbor, použijeme príkaz Rewrite (F). Súbor nemusí fyzicky existovať na disku, príkazom rewrite bude vytvorený a otvorený na zápis (prístupová premenná nemá definovanú hodnotu). Ak už súbor existoval na disku, jeho údaje nebudú prístupné!

Vlastné čítanie zo súboru prebieha príkazom Read (F , *premenná* ), kde premenná musí byť rovnakého typu ako typ načítavaného údaja. Do premennej sa načíta jeden údaj typu, ako je typ premennej. Ako sa príkaz read správa pri čítaní rôznych údajových typov sme popísali na prvej strane.

Aby sme mohli efektívne opakovane čítať údaje zo súboru, potrebujeme ešte poznať funkciu EOF (End Of File). Funkcia eof(F) vráti hodnotu true, ak je prístupová premenná na konci súboru, inak vracia hodnotu false.

<u>Programová schéma na čítanie číselných hodnôt zo súboru (ak nás nezaujíma jeho členenie na riadky)</u>:

```
reset(F);                    // otvor súbor a nastav čítanie na začiatok
while not eof(F) do          // pokiaľ nie je koniec súboru opakuj
begin
  read(F,X);                 // prečítaj údaj zo súboru do premennej X
  // spracovanie X           // napr. Memo1.Lines.Add(IntToStr(X))
end;
```

Uvedená programová schéma je dobrá na čítanie číselných hodnôt zo súboru, pričom „si nevšíma", či sú čísla v jednom riadku alebo vo viacerých riadkoch - číta od prvého čísla po posledné v súbore (čísla samozrejme musia byť od seba oddelené napríklad medzerou alebo ďalšie číslo na novom riadku).

Schému nemožno použiť na čítanie hodnôt typu string, keďže prvým vykonaním príkazu read(F,X) sa načíta do premennej X celý riadok a v schéme nie je príkaz prikazujúci pokračovať v čítaní na novom riadku. Pri čítaní číselných hodnôt je prechod na nový riadok zabezpečený vlastnosťou

čítania číselných údajov, keď pri „hľadaní" ďalšej číselnej hodnoty čítacia hlava ignoruje koniec riadka a pokračuje automaticky na nasledujúcom riadku!

Poznámka:
Aby sme boli dôslední, možno uvedenú programovú schému použiť napríklad aj na čítanie reťazcov zadanej dĺžky, napriek tomu, že sú uvedené v jednom riadku. V súbore je v riadku napríklad napísané JanoFeroMaraZuzaRobo. Ak X je typu string[4], budú postupne správne načítané jednotlivé mená.
Príkaz read možno použiť aj v tvare read ( F, *premenná1*, *premenná2* , ... , *premennáN* ) čo vlastne zodpovedá sekvencii begin read(F,*premenná1*); read(F,*premenná2*); ... read(F,*premennáN*) end

Ak by ste si chceli vyššie uvedenú programovú schému vyskúšať, potrebujete ešte poznať príkaz CloseFile. Príkazom closefile(F) sa uzavrie súbor. Pred skončením programu by sa mali všetky otvorené súbory uzavrieť. Takže uvedenú programovú schému treba doplniť už len o príkaz assignfile a o príkaz closefile. Zavrieť súbor sa nám zdá praktické po každom skončení práce so súborom (v každej procedúre). Je to nevyhnutné napríklad aj vtedy, keď si chceme pozrieť, čo bolo do súboru zapísané a nechceme ukončiť ešte program. Systém totiž nemusí „bežať" s každým údajom hneď na disk (kvôli efektívnosti), príkazom closefile ho však prinútime zapísať na disk všetky na zápis poslané údaje.
Príkaz closefile možno chápať ako „párový" k príkazu na otvorenie súboru (reset, rewrite, append). Keďže pred každou prácou so súborom potrebujem prístupovú premennú nastaviť na vhodné miesto v súbore (začiatok alebo koniec), a teda použiť príkaz na otvorenie súboru, môžem ho, po skončení čítania alebo zápisu, pokojne zavrieť.
Otváranie a zatváranie súboru nemá vplyv na príkaz assignfile, ten zostáva neustále v platnosti počas existencie premennej F alebo kým nepoužijeme príkaz assignfile(F,...) s iným názvom súboru na disku.

A ešte niekoľko dobrých rád:
Navyknite si po spustení Delphi hneď kliknúť na tlačidlo Save **All**, vytvoriť nový priečinok, a do neho uložiť vytváraný Unit1 aj Project1. Do tohto priečinka ukladajte aj textové súbory a v ňom tiež hľadajte súbory, ktoré boli programom vytvorené. Vyhnete sa tým chybovým hláseniam Súbor nenájdený a tiež nemusíte zadať cestu k súboru, ktorá by po prenesení programu na iný počítač pravdepodobne neplatila.

## Príklad 18.1

Procedúra na výpočet aritmetického priemeru čísel zapísaných v textovom súbore cisla.txt (nepotrebujeme vedieť, či sú čísla v jednom riadku alebo „rozhádzané" vo viacerých riadkoch).

```
procedure TForm1.btPriemerClick(Sender: TObject);
var F: TextFile;
     X, Sucet: real;
    Pocet: integer;
begin
assignfile( F , 'cisla.txt' );
reset(F);
Pocet:= 0;
Sucet:= 0;
while not eof(F) do
begin
        read(F,X);
        Sucet:= Sucet + X;
        Pocet:= Pocet + 1;
end;
```

```
Memo1.Lines.Add( Format( 'Spracovaných bolo %d čísel a ich priemer je %f' , [Pocet,Sucet/Pocet] ) );
closefile(F);
end;
```

Súbor cisla.txt ľahko vytvoríte nasledovným postupom:

1. prejdeme do priečinka, v ktorom máme uložený projekt
2. klikneme pravým tlačidlom myši do voľnej plochy a z kontextovej ponuky si vyberiem Nový, z podponuky Textový dokument, ktorý hneď premenujeme na cisla. Príponu txt pripojí systém sám napriek tomu, že ju nevidíme (pozor, aby sa súbor nevolal cisla.txt.txt, keď nie je zobrazovaná prípona).
3. súbor cisla otvoríme napríklad dvojklikom na ikonu a vpíšeme čísla oddelené medzerami. Pozor! Za posledným číslom v riadku stlačíme **Enter** (nesmie tam byť napríklad medzera!). Za posledným číslom v súbore nedáme **Enter**! Súbor končí posledným vloženým číslom a nie novým riadkom.
4. súbor uložíme (Súbor - Uložiť) a zavrieme.

Častejšie je však textový súbor členený na riadky. Funkcia, ktorá je schopná rozpoznať koniec riadka má označenie EOLN (End Of Line). Funkcia eoln(F) vráti hodnotu true, ak je prístupová premenná na konci riadka, inak vracia hodnotu false. Príkaz ReadLn nastaví prístupovú premennú na začiatok nového riadka.

<u>Programová schéma na čítanie hodnôt z textového súboru po riadkoch:</u>

```
assignfile(F, 'test.txt');            // čítať sa bude zo súboru test.txt
reset(F);                             // otvor súbor a nastav čítanie na začiatok
while not eof(F) do                   // pokiaľ nie je koniec súboru opakuj
begin
        while not eoln(F) do          // pokiaľ nie je koniec riadka opakuj
        begin
                read(F,X);            // prečítaj údaj zo súboru do premennej X
                // spracovanie X
        end;
readln(F);                            // prejdi na nový riadok
end;
closefile(F);                         // zavri súbor
```

Sekvenciu

```
        read( F, premenná );
        readln ( F );
```

možno nahradiť aj jediným príkazom

```
        readln ( F, premenná );
```

Najprv sa do premennej načíta príslušný údaj a potom sa prístupová premenná t.j. čítanie nastaví na začiatok nového riadka. Ak boli v riadku, z ktorého sa čítalo, ďalšie hodnoty, budú ignorované (preskočené).
Príkaz možno použiť aj v tvare readln ( F, premenná1, premenná2 , ... , premennáN ).
Pomocou príkazu readln možno aj „preskočiť" čítanie z riadka.

## Príklad 18.2

Procedúra na výpočet aritmetických priemerov čísel v jednotlivých riadkoch súboru cisla.txt (predpokladáme, že v každom riadku je aspoň jedno číslo,ináč treba ošetriť delenie nulou).

```
procedure TForm1.btPriemeryClick(Sender: TObject);
var F: TextFile;
    X, Sucet: real;
    Pocet: integer;
begin
assignfile( F , 'cisla.txt' );
reset(F);
while not eof(F) do
begin
        Pocet:= 0;
        Sucet:= 0;
        while not eoln(F) do
        begin
                read(F,X);
                Sucet:= Sucet + X;
                Pocet:= Pocet + 1;
        end;
        readln(F);
        Memo1.Lines.Add(Format('Počet čísel v riadku: %d, ich priemer: %f',[Pocet,Sucet/Pocet]));
end;
closefile(F);
end;
```

Stačí, aby sme v súbore cisla.txt stlačili ENTER za posledným číslom v súbore (vložili nový riadok), a procedúra v príklade 18.1 dá zlý výsledok (vyskúšajte). Ak vložíme medzeru za posledné číslo v ktoromkoľvek riadku, procedúra v príklade 18.2 dá zlé výsledky (vyskúšajte). Popíšeme aspoň jednu zo spomenutých situácií. Ak za posledným číslom v súbore je stlačený kláves ENTER (t.j. vložený ďalší riadok), čítacia hlava po načítaní číselného údaja je nastavená na koniec riadka a preto funkcia eof(F) v cykle while vráti hodnotu false. K slovu teda opäť príde príkaz read(F,X), čo je chyba, pretože už žiaden číselný údaj nenájde (len koniec súboru). V podstate by sme potrebovali funkciu, ktorá by bola schopná rozpoznať, že za posledným číselným údajom v súbore už nie je žiaden platný údaj a hneď vrátiť true u funkcie eof. Všeobecnejšie: ktorá by bola schopná preskočiť balast (medzery, tabulátory, prázdne riadky) a nastaviť čítanie buď na nasledujúci číselný údaj alebo na koniec súboru. Túto vlastnosť má funkcia SeekEof. Pri volaní funkcie SeekEof(F) funkcia preskočí všetky medzery, tabulátory, konce riadkov a až keď nájde prvý nemedzerový znak (aj koniec riadka je interpretovaný ako medzera!), zavolá funkciu Eof(F) a vráti jej hodnotu ako svoju. Ak vráti true, čítanie je na konci súboru, ak vráti false, čítanie je na prvom znaku vstupnej hodnoty (ak nie je očakávaného typu, zobrazí sa chybová správa Invalid ... input.).
Podobná situácia nastáva v riadkoch, keď za posledným číselným údajom v riadku je vložená ešte medzera alebo stlačený tabulátor (cyklus while not eoln(F) do... nebude správne ukončený). Na odstránenie uvedenej nepríjemnosti bola zavedená funkcia SeekEoln. Funkcia SeekEoln(F) zabezpečí „preskočenie balastu“ v riadku za posledným platným údajom. Potom zavolá funkciu eoln(F) a vráti jej hodnotu ako svoju.

Po tomto poučení už vieme, že dokonalejšia programová schéma na čítanie z textového súboru po riadkoch má tvar:

```
reset(F);                          // otvor súbor a nastav čítanie na začiatok
while not seekeof(F) do            // pokiaľ nie je koniec súboru opakuj
begin
        while not seekeoln(F) do   // pokiaľ nie je koniec riadka opakuj
```

```
    begin
        read(F,X);                  // prečítaj údaj zo súboru do premennej X
        // spracovanie X
    end;
    readln(F);                      // prejdi na nový riadok, program vďaka seekeof bude
                                    //správne fungovať aj bez tohto príkazu!
end;
closefile(F);                       // zavri súbor
```

Program je schopný textový súbor aj vytvoriť, prípadne pripísať hodnoty na jeho koniec. Na zápis do textového súboru slúžia príkazy Write a WriteLn. Príkaz write má tvar write(F, výraz), kde výraz je rovnakého typu ako je typ údaja, ktorý sa má do súboru zapísať. Dovolené sú typy char, string, boolean, integer a real (samozrejme aj ich modifikácie a intervaly). Vykonanie príkazu write(F, výraz) je jednoduché. Vyhodnotí sa výraz a jeho hodnota sa zapíše do súboru F.
Príkaz writeln(F) zapíše do súboru F „hodnotu" koniec riadka. Nasledujúci zápis bude pokračovať na začiatku nového riadka. Príkaz writeln(F, výraz) zapíše do aktuálneho riadka hodnotu výrazu a potom „hodnotu" koniec riadka.
Dovolené sú aj tvary write(F, výraz1, výraz2, ... , výrazN) resp. writeln(F, výraz1, výraz2, ... , výrazN), ale pozor! Ak by boli v premenných x, y a z napríklad tri celé čísla, príkazom write(F, x, y, z) sa ich hodnoty zapíšu do súboru F za sebou bez oddeľujúcich znakov a k ich pôvodným hodnotám sa už sotva dostaneme. Vyriešiť tento problém môžeme napríklad zápisom write(F, x, ' ', y, ' ', z) kde sme medzi hodnoty x a y a medzi hodnoty y a z dali zapísať medzery.

## Príklad 18.3

Procedúra na zapísanie zadaného počtu náhodne vybraných celých čísel z intervalu <0,99> do súboru cisla.txt. Najjednoduchším riešením je zapísať číslo do riadku a nastaviť zápis na nový riadok.

```
procedure TForm1.btZapisPodSebaClick(Sender: TObject);
var F: TextFile; Pocet, i: integer;
begin
Pocet:= StrToInt( InputBox( 'Vstup' , 'Do súboru zapísať čísel (počet):' , '10' ) );
assignfile(F , 'cisla.txt' );
rewrite(F);                                        // začína nový prázdny súbor
for i := 1 to Pocet do writeln( F , random(100) );  // vygeneruje číslo a zapíše do súboru
closefile(F);
end;
```

## Príklad 18.4

Procedúra na zapísanie zadaného počtu náhodne vybraných celých čísel z intervalu <0,99> do súboru cisla.txt. Čísla budú zapísané do jedného riadka.

```
procedure TForm1.btZapisDoRiadkaClick(Sender: TObject);
var F: TextFile; Pocet, i: integer;
begin
Pocet:= StrToInt(InputBox( 'Vstup' , 'Do súboru zapísať čísel (počet):' , '10' ) );
assignfile( F , 'cisla.txt' );
rewrite(F);
for i := 1 to Pocet-1 do write( F, random(100) , ' ' );      // čísla sú oddelené medzerou
write( F , random(100) );              // zapíše sa posledné číslo, za ktorým už nebude medzera!
closefile(F);
end;
```

### Príklad 18.5

Procedúra na zapísanie zadaného počtu náhodne vybraných celých čísel z intervalu <0,99> do súboru cisla.txt. Do riadka sú zapisované nepárne čísla až po prvé párne číslo, to sa ešte zapíše do riadka a zápis sa nastaví na začiatok nového riadka.

```
procedure TForm1.btZapisPoParneClick(Sender: TObject);
var F: TextFile; Pocet, i, x: integer;
begin
Pocet:= StrToInt( InputBox( 'Vstup' , 'Do súboru zapísať čísel (počet):' , '10' ) );
assignfile( F ,'cisla.txt' );
rewrite(F);
for i := 1 to Pocet-1 do          // spracuje sa Pocet-1 čísel
begin
     x:= random(100);             // vygeneruje celé číslo z intervalu <0,99> a uloží do x
     write(F, x);                 // zapíše vygenerované číslo do súboru
     if x mod 2 = 0
     then writeln(F)              // ak je x párne, nastaví zápis na začiatok nového riadka
     else write(F,' ')            // inak zapíše medzeru, lebo do riadka bude zapísané ďalšie číslo
end;
write(F, random(100));            // zapíše posledné číslo bez „balastu“
closefile(F);
end;
```

Ak chceme do existujúceho textového súboru (trebárs aj prázdneho) pridať hodnotu, môžeme tak urobiť zápisom len na koniec súboru. Vtedy musíme súbor otvoriť príkazom **Append**. Príkaz append(F) otvorí textový súbor na zápis a zároveň nastaví prístupovú premennú na koniec súboru. Následne príkazom write alebo writeln môžeme zapísať hodnotu (hodnoty) na koniec súboru.

### Príklad 18.6

Procedúra zapíše na koniec súboru cisla.txt číslo 100. Pred číslo 100 bude zapísaná medzera, číslo bude zapísané v aktuálnom riadku.

```
procedure TForm1.btZapisNaKoniecClick(Sender: TObject);
var F: TextFile; Pocet, i, x: integer;
begin
assignfile( F , 'cisla.txt' );
append(F);                        // otvor súbor a nastav zápis na koniec súboru
write( F , ' 100' );              // zapíš medzeru a 100
closefile(F);
end;
```

### Príklad 18.7

Procedúra zapíše na koniec súboru cisla.txt číslo 100. Číslo 100 bude zapísané na začiatok nového riadka.

```
procedure TForm1.btZapisNaKoniecClick(Sender: TObject);
var F: TextFile; Pocet, i, x: integer;
begin
assignfile(F,'cisla.txt');
append(F);                        // otvor súbor a nastav zápis na koniec súboru
writeln(F);                       // prejdi na nový riadok
write(F,'100');                   // zapíš 100
closefile(F);
end;
```

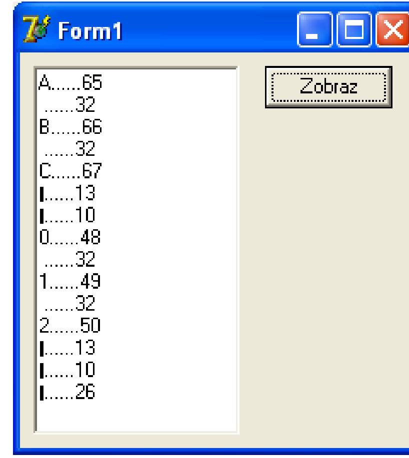
Zhrnutie zásad zápisu do súboru:

- ak vytvárame celkom nový súbor, pred prvým zápisom musíme použiť príkaz rewrite (súbor na disku nemusí existovať)
- ak chceme do existujúceho súboru pripísať hodnoty na koniec súboru, musíme pred zápisom použiť príkaz append
- najjednoduchšie je zapísať jeden údaj do jedného riadku, t.j. použiť príkaz writeln(F, výraz)
- jeden údaj v jednom riadku odporúčame použiť pri type string[2] a pri type boolean (číta sa zo súboru ako string)
- použitie príkazu writeln znamená, že za posledným údajom v súbore bude ešte zapísaný kód koniec riadka a preto pri čítaní z takéhoto súboru sa musí použiť funkcia seekeof
- pri zápise viacerých číselných údajov do jedného riadka musíme zapísať aj oddeľovače, najjednoduchšie medzery
- ak zapíšeme medzeru aj za posledný údaj v riadku, musíme pri čítaní použiť funkciu seekeoln

Na záver úvah o textových súboroch sa pokúsme zistiť, ako je zapísaný v súbore koniec riadka. Jednou z možností je dať si vypísať ordinálne čísla znakov od prvého až po posledný v súbore, ktorý obsahuje aj koniec riadka. Takýto program môže mať tvar:

```
procedure TForm1.btZobrazClick(Sender: TObject);
var Z: char;
begin
assignfile(F , 'test.txt' );
reset(F);
while not eof(F) do
begin
        read( F , Z );
        Memo1.Lines.Add( Z + '......' + IntToStr( ord(Z) ) );
end;
read(F,Z);
Memo1.Lines.Add( Z + '......' + IntToStr( ord(Z) ) );
closefile(F);
end;
```

Do súboru test.txt sme zapísali:
A medzera B medzera C ENTER
0 medzera 1 medzera 2 ENTER
teda súbor obsahuje dvakrát koniec riadka. Pri analýze výstupu v komponente Memo (obrázok) vidíme, že znak A má kód 65, medzera má kód 32, znak B kód 66, medzera 32, znak C má poradové číslo 67 a na „hodnotu“ koniec riadka zostávajú dve čísla, 13 a 10. Potvrdzuje to aj koniec výpisu, kde sú opäť čísla 13 a 10 (a zatiaľ záhadné číslo 26). Ak poznáte tabuľku na kódovanie znakov ASCII alebo Unicode (pozri napríklad v Microsoft Word, Vložiť - Symbol... - Symboly (Normálny text)), viete, kde sa zobrali uvedené ordinálne čísla. Ordinálne číslo znaku je číslo zodpovedajúce poradiu načítaného znaku v kódovacej tabuľke. Koniec koncov, ak použijete ľavé ALT a 65 vložíte znak A, ľavé ALT + 32 vložíte medzeru, možno používate napríklad ľavé ALT a 64 na vloženie znaku @ alebo ľavé ALT + 39 na vloženie apostrofu v editore Delphi v slovenskej klávesnici,... Znaky v kódovacích




[2] Samozrejme z reťazca v riadku dokážeme podľa potreby vyrezať „viacej údajov“, prečo si však komplikovať život.

tabuľkách od 0 po 31 sú riadiace a zobrazia sa napríklad ako v našom Memo (čierne „paličky“). Pod poradovým číslom 13 je znak CR (Carriage Return) a pod číslom 10 znak LF (Line Feed). Tieto dva znaky v operačných systémoch MS DOS - Windows znamenajú koniec riadka. Tento poznatok môžeme aj užitočne využiť napríklad v Delphi v komponente ShowMessage. Vyskúšajte si príkaz ShowMessage('Text v prvom riadku' + chr(13) + chr(10) + 'text v druhom riadku'). Programátori poznajú aj kratší zápis ShowMessage('Text v prvom riadku'#13#10'text v druhom riadku'); (bez medzier!). Znak # vložíte v slovenskej klávesnici ako pravé ALT + X (v ASCII tabuľke ho nájdete pod číslom 35).
No a záhadné číslo 26 na konci výpisu? Ak ste šikovní, uhádnete, že to je znak koniec súboru. Museli sme ho osobitne načítať po skončení cyklu while not eof(F) do... v poslednej procedúre.

Častou chybou pri pokuse vykonať príkaz reset je nenájdenie súboru (File not found), ktorého meno prípadne cesta sú uvedené v príkaze assignfile. Je to spôsobené buď neexistenciou súboru alebo zle zadanou cestou (ak cesta nie je zadaná, súbor je hľadaný v aktuálnom priečinku, t.j. v priečinku, kde sú súbory Unit1, Project1,...). Je viacej možností na ošetrenie tejto chyby. Uvedieme snáď najzákladnejšie ošetrenie, principiálne použiteľné v Pascale, Turbo Pascale, Delphi.

```
assignfile( F , 'test.txt' );
{$I-} reset(F); {$I+}                    //vypnutie kontroly vstupu pred reset, zapnutie po reset
if IOResult > 0      //IOResult je premenná systému, ukladá sa do nej číslo chyby (0-chyba nenastala)
then begin
        ShowMessage ('Súbor nenájdený, končím!');
        close                            //prakticky ukončenie behu programu v Delphi
     end
else...                                  //pokračuj, všetko v poriadku, súbor bol otvorený
```

Ošetrenie tzv. výnimky pomocou try ... except v Delphi:

```
assignfile( F , 'test.txt' );
try
  reset(F)                        // problémový príkaz, ktorý môže vyvolať chybu - výnimku
except                            // príkazy, ktoré sa majú vykonať, ak nastala výnimka
  if MessageDlg('Súbor nenájdený! Vytvoriť nový?',mtWarning,[mbYes,mbNo],0) = mrYes
  then rewrite(F)                 // vytvor nový súbor
  else Application.Terminate;     // ukonči program
end;
// príkazy, ktoré sa majú vykonať po vykonaní príkazu try
```

**Pozor**! V prostredí Delphi musí byť vypnuté **Stop on Delphi Exceptions (Tools - Debugger Options... - záložka Language Exceptions)**, čím programátor preberá na seba ošetrenie výnimky. Vyššie uvedené ošetrenie je možné použiť napríklad na inicializáciu programu. Ak textový súbor neexistuje (napríklad pri prvom spustení programu), program ponúkne vytvoriť nový a môžeme do neho začať zapisovať, ak existuje, môžeme začať pracovať s jeho obsahom.

Delphi ponúka "kopec" ďalších možností na otvorenie súboru, najmä v kombinácii s komponentom OpenDialog v záložke Dialogs alebo s funkciou FileExists (FileName: string): boolean.

### Príklad 18.8

Vytvorte program na spracovanie súťaže skoku do diaľky. V textovom súbore skoky.txt je v riadku zapísané číslo pretekára (trojciferné), meno pretekára (max. 20 znakov) a skratka štátu (3 znaky). V nasledujúcom riadku sú zapísané dĺžky jeho skokov v cm (celé čísla). Ak pretekár nenastúpil na súťaž, má zapísanú hodnotu -1, ak mal neplatný skok, má zapísanú 0. Príklad súboru skoky.txt:

001Korčuľa IvanUKR
0 0 0 0 0
123Machuľa DaliborSVK
0 679 786 0 800
333Latka JánSVK
-1
999Sandokaan GavrilovičXXX
798 801

```
Skoky
Číslo Meno pretekára        Štát          [Vypíš pretekárov]
 001  Korčuľa Ivan          UKR           [Kvalifikovali sa]
 123  Machuľa Dalibor       SVK           [Najdlhší pretekárov skok]
 333  Latka Ján             SVK
 999  Sandokaan Gavrilovič  XXX

Čísla, mená a prísl. pretekárov, ktorí
sa kvalifikovali do finále (budú
zapísaní aj do finale.txt):
 123  Machuľa Dalibor       SVK
 999  Sandokaan Gavrilovič  XXX

Mená pretekárov a ich najdlhšie skoky:
Korčuľa Ivan           nemal platný skok
Machuľa Dalibor        800
Latka Ján              nenastúpil
Sandokaan Gavrilovič   801
```

↑ Formulár s výstupmi z troch procedúr

Podstatná časť programu:

```
var F: TextFile;
    Skok, KvalDlzka: integer;
    Meno: string[20];
    Cislo, Stat: string[3];

// procedúra na zistenie, či sa našiel súbor skoky.txt
procedure TForm1.FormActivate(Sender: TObject);
begin
assignfile(F,'skoky.txt');
try
  reset(F);
except
  ShowMessage( 'Súbor skoky.txt nenájdený!' );
  Application.Terminate;
end;
end;

// procedúra na vypísanie čísel pretekárov, ich mien a štátu, za ktorý pretekajú
procedure TForm1.btVypisClick(Sender: TObject);
var Riadok: string[26];          // riadok má maximálne 26 znakov, 3 číslo + max. 20 meno + 3 štát
begin
Memo1.Lines.Add('Číslo Meno pretekára      Štát');             // hlavička výpisu
reset(F);
while not seekeof(F) do
begin
  readln(F,Riadok);
  Memo1.Lines.Add( Format( '%4s  %-20s %s' , [ Copy(Riadok,1,3) , Copy(Riadok,4,length(Riadok)-6) ,
                    Copy( Riadok,length(Riadok)-2,3) ] ) );
  readln(F)                                       // preskočenie riadka so skokmi, ten nás teraz nezaujíma
end;
closefile(F);
end;

// procedúra, ktorá do súboru finale.txt zapíše všetkých pretekárov, ktorí skočili kvalifikačnú dĺžku
procedure TForm1.btKvalifikClick(Sender: TObject);
var G: TextFile;
    Riadok: string[26];
    Prvy: boolean;               // premenná, ktorá "sleduje", či zápis do finale.txt bude prvý alebo nie
begin
KvalDlzka:= StrToInt( InputBox( 'Skoky' , 'Kvalifikačná dĺžka v cm' , '800' ) );   // zadanie kvalif.dĺžky
assignfile(G,'finale.txt'); rewrite(G);
```

```
Memo1.Lines.Add('Čísla, mená a príslušnosť pretekárov, ktorí sa kvalifikovali do finále (budú zapísaní aj
do finale.txt):');
Prvy:= true;
reset(F);
while not seekeof(F) do
begin
   readln(F,Riadok);
   repeat
          read(F,Skok);                      // čítanie z riadka s dĺžkami skokov
          if Skok >= KvalDlzka
          then begin                  // ak pretekár splnil kval.dĺžku, treba ho zobraziť a zapísať do finale.txt
                    Memo1.Lines.Add(Format('%4s  %-20s %s' , [ Copy(Riadok,1,3) , Copy(Riadok , 4 ,
                                             length(Riadok)-6) , Copy(Riadok , length(Riadok)-2,3) ] ) );
                    if not Prvy
                    then writeln(G)          // ak to nie je prvý zápis, treba presunúť "kurzor" na nový riadok
                    else Prvy:= false;       // realizuje sa prvý zápis, ďalší už nebude prvý
                    write(G, Riadok)         // po zápise zostávame v tom istom riadku!
                end;
   until (Skok >= KvalDlzka) or seekeoln(F);          // efektívne ukončenie čítania v riadku so skokmi
   readln(F);                                          // prechod na nový riadok v súbore skoky.txt
end;
closefile(F);
closefile(G);
end;
```

```
// procedúra, ktorá vypíše mená pretekárov a ich najdlhšie skoky, prípadne nemá platný skok alebo
nenastúpil
procedure TForm1.btNajdlhsiClick(Sender: TObject);
var Max: integer;                // premenná, do ktorej sa ukladá najdlhší skok pretekára
     Riadok: string[26];
begin
reset(F);
Memo1.Lines.Add('Mená pretekárov a ich najdlhšie skoky:');
while not seekeof(F) do
begin
   readln(F,Riadok);             // načítanie riadku s číslom, menom a štátom pretekára
   Max:= -1;
   while not seekeoln(F) do
   begin                         // hľadanie najväčšieho čísla v riadku so skokmi
          read(F,Skok);
          if Skok > Max then Max := Skok
   end;
   readln(F);                    // "vystúpenie" z riadka so skokmi na nasledujúci riadok (s menami)
   if Max = -1
   then Memo1.Lines.Add( Format( '%-20s  nenastúpil' , [ Copy(Riadok , 4 , Length(Riadok)-6) ] ) );
   if Max = 0
   then Memo1.Lines.Add( Format( '%-20s  nemal platný skok' , [ Copy(Riadok,4,Length(Riadok)-6) ] ) );
   if Max>0
   then Memo1.Lines.Add( Format( '%-20s  %d' , [ Copy(Riadok,4,Length(Riadok)-6) , Max ] ) );
end;
closefile(F);
```

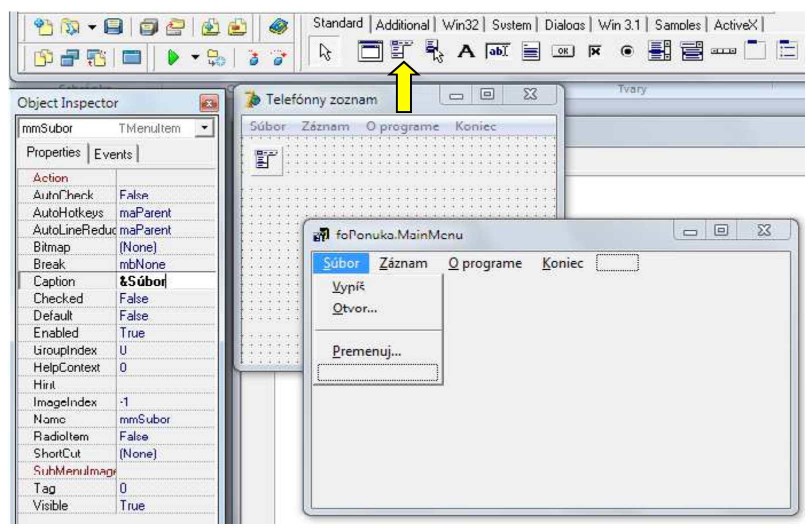
```
end;
end.
```

Poznámka:
Funkcia Copy(*s*, *od*, *kolko*) vráti podreťazec z reťazca *s* (string) začínajúci *od* (integer) znaku s počtom znakov *kolko* (integer). Nech napríklad s = '123Machuľa DaliborSVK'
Copy(s,1,3) = 123, Copy(s,4,length(s)-6) = Machuľa Dalibor a Copy(s, length(s)-2,3) = SVK.
Funkcia length(s) vráti dĺžku (integer) reťazca s, napríklad length('123Machuľa DaliborSVK') = 21.
Obe funkcie možno ľahko naprogramovať pomocou for a while.

## Príklad 18.9

Vytvorte program simulujúci elektronický telefónny zoznam. Program nech umožňuje v záznamoch evidovať priezvisko a meno, adresu a telefónne číslo. V položke Súbor nech umožňuje vypísať databázu - súbor, uložiť alebo premenovať. V položke Záznam nech umožňuje pridať nový, editovať existujúce záznamy prípadne odstrániť záznam z databázy.

*Riešenie*




Príklad je vhodný na predstavenie nových komponentov. Prvým je MainMenu, ktorý nájdeme v záložke Additional (v obrázku naň ukazuje žltá šípka). Štandardne ho vložíme do formulára. Po dvojkliku na jeho ikonu vo formulári môžeme ponuku upravovať. Vlastnosť Caption reprezentuje názov položky. Znak & pred zvoleným písmenom položky nastavuje tzv. horúci kláves (rýchlu voľbu pri stlačenom ľavom klávese ALT). Nezabudnite najprv kliknúť do „prázdnej“ položky, ktorú idete upravovať. Vytvorte položky Súbor, Záznam, O programe a Koniec. Ak sa pomýlite, použite kontextovú ponuku, ktorú vyvoláte stlačením pravého tlačidla myši nad položkou. Udalosť „užívateľ klikol na položku“ sa programuje rovnako, ako napríklad pri tlačidlách, t.j. stačí dvojkliknúť na položku. Pre jednoduchšiu orientáciu sme začali pomenúvať objekty, napríklad Unit1 sme premenovali na Ponuka, Form1 na foPonuka a jednotlivé položky MainMenu na mmNázovPoložky.

Začneme od najjednoduchšej položky po najzložitejšiu. Po dvojkliku na Koniec sa do Unit1 (u nás premenovaného na Ponuka) vloží procedúra

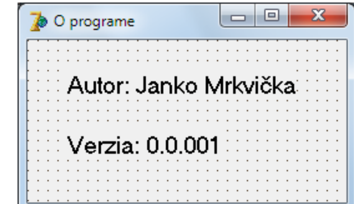
```
procedure TForm1.Koniec1Click(Sender: TObject);    //resp. procedure TfoPonuka.Koniec1Click(...);
begin

end;
```

v ktorej sme vlastnosť Name zmenili na mmKoniec a medzi begin a end dopísali Application.Terminate;. Tým máme funkčnú prvú položku ponuky a môžeme ju vyskúšať.

V ďalšom kroku si ukážeme použitie modálneho formulára. Po kliknutí na položku O programe chceme, aby sa zobrazil nový formulár s údajmi o programe, napríklad meno autora a verzia. Formulár môže byť modálny, čo znamená, že kým ho nezavrieme, nemôžeme zaktívniť iný formulár programu. Nemodálny formulár dovoľuje kliknúť v programe aj na iný formulár bez toho, aby sme nemodálny zavreli.

Postup na vytvorenie modálneho formulára je nasledovný:

1. cez položky File – New – Form vložíme nový formulár do programu, napríklad Form2 (programovaný v Unit2)
2. v Unit1 (Ponuka) na začiatku interfejsovej časti v uses dopíšeme Unit2;
3. dvojklikneme na ikonu MainMenu a na položku O programe
4. v Unit1 sa vytvorí procedúra
```
procedure TForm1.Oprograme1Click(Sender: TObject);
begin

end;
```
5. do procedúry Oprograme1Click dopíšeme
```
var Modal: TForm2;
begin
Modal:= TForm2.Create(Application);
try
        Modal.ShowModal;
finally
        Modal.Free;
end;
end;    //procedúry
```
   Pre lokálnu premennú Modal si môžete zvoliť aj iný názov, blok tejto procedúry však budeme viackrát kopírovať, preto je snáď vhodnejšie použiť univerzálny názov. Uvedené príkazy zabezpečia zobrazenie formulára Form2 a po jeho zavretí uvoľnenie obsadenej časti pamäte.
   Teraz už môžeme vyskúšať funkčnosť nášho riešenia, po spustení programu a kliknutí na položku O programe by sa mal zobraziť modálny formulár Form2.
6. po ukončení programu môžeme formulár Form2 upraviť do požadovaného tvaru, ako to vidieť napríklad na obrázku vpravo
7. ak v sekundárnom unite využívate údaje z primárneho unitu, musíte do implementačnej časti sekundárneho unitu hneď pod slovo implementation dopísať uses Unit1; (resp. uses Ponuka;)!




Celý postup si zopakujeme pri programovaní položky Vypíš. Jej úlohou je vypísanie obsahu textového súboru v Memo nového formulára pomocou príkazu Memo1.Lines.LoadFromFile(); Vykonáme „prípravné“ práce. V interfejsovej časti primárneho unitu (Ponuka) zadeklarujeme globálnu premennú F typu TextFile a NazovSuboru typu string. Vytvoríme inicializačnú časť na konci unitu

```
initialization
NazovSuboru:= 'zoznam.txt';
assignfile(F,NazovSuboru);
try
        reset(F);                       //problémový príkaz; ak súbor nenájde, nastane výnimka
except                                  //ošetrenie výnimky
        ShowMessage('Súbor nenájdený, vytváram nový!');
        rewrite(F);
end;
closefile(F);
end.
```

Môžeme sa pustiť do vytvorenia formulára na vypísanie obsahu súboru s názvom NazovSuboru.

1. vložíme do programu nový formulár (Form3)
2. do uses v primárnom unite dopíšeme Unit3
3. dvojkliknime na ikonu MainMenu v primárnom formulári a následne na položku Vypíš

4. do novovytvorenej procedúry prekopírujeme blok vytvárajúci modálny formulár a prepíšeme dvakrát TForm2 na TForm3
5. upravíme Form3 podľa požiadaviek (vložíme Memo1)
6. keďže primárny a sekundárny unit majú spolupracovať (v Unit3 sa použije hodnota premennej NazovSuboru), musíme na začiatok implementačnej časti v Unit3 dopísať uses Unit1;
7. hneď po otvorení formulára Form3 sa má zobraziť v Memo1 obsah súboru, preto využijeme udalosť OnActivate resp.FormActivate. Ukážka implementačnej časti unitu

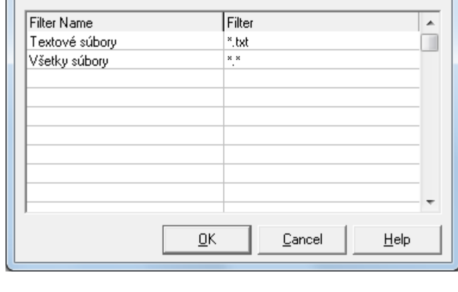
```
implementation
{$R *.dfm}
uses Ponuka;

procedure TfoVypis.FormActivate(Sender: TObject);
begin
Memo1.Lines.LoadFromFile(NazovSuboru);
end;

end.
```

Do Mema zapíšte aspoň jeden databázový záznam, napríklad

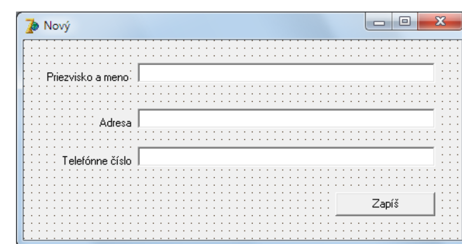
Kováč Ján
Nitra, Dlhá 12
037/654321

t.j. každý údaj na samostatný riadok (za telefónnym číslom nie je stlačený ENTER).

Keďže Memo nám umožňuje robiť vo výpise súboru zmeny, doplnili sme unit Vypis o procedúru, ktorá sa, po kliknutí na zavrieť formulár Vypíš (udalosť FormClose), opýta, či sa má obsah Mema uložiť, ak áno, otvorí sa štandardné windowsovské okno Uložiť ako.... Aby sa tak stalo, musíme do formulára Vypíš vložiť komponent SaveDialog, ktorý nájdete v záložke Dialogs. Ak chcete, aby zobrazoval prednostne len textové súbory, jeho vlastnosť Filter doplňte podľa ukážky. Symbol * v operačnom systéme nahrádza ľubovoľnú skupinu znakov.




```
procedure TfoVypis.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
if MessageDlg('Uložiť?',mtWarning, [mbYes, mbNo], 0) = mrYes
then if SaveDialog1.Execute
     then begin
               NazovSuboru:= SaveDialog1.FileName;
               if ExtractFileExt(NazovSuboru)='' then NazovSuboru:= NazovSuboru + '.txt';
               Memo1.Lines.SaveToFile(NazovSuboru);
               AssignFile(F,NazovSuboru);
          end;
end;
```



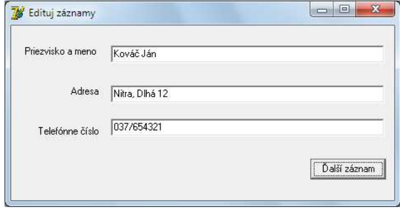

Poslednou položkou, ktorú podrobnejšie popíšeme, je Záznam - Nový, ktorá umožňuje pridať na koniec súboru (databázy) ďalší záznam, t.j. priezvisko a meno, adresu a telefónne číslo. Urobíme tak zapísaním údajov z formulára Nový – obrázok vpravo, do súboru, ak aspoň údaj Priezvisko a meno je

neprázdny. Musíme však rozlíšiť dve situácie, keď sa ide zapísať prvý záznam do súboru alebo keď sa ide pridať záznam pod iný záznam. V druhej situácii totiž musíme najprv zápis nastaviť na nový riadok.

```
implementation
{$R *.dfm}
uses Ponuka;

procedure TfoNovy.btNovyClick(Sender: TObject);
begin
if Edit1.Text<>''                //ak je čo zapísať
then begin
        reset(F);                //nastav čítanie na začiatok súboru, aby sme zistili, či je niečo v súbore
        if eof(F)                //ak je zároveň koniec súboru, súbor neobsahuje ani jeden záznam
        then rewrite(F)          //nastav zapisovanie od začiatku resp. mohlo by byť aj append(F)
        else begin
                append(F);       //nastav zapisovanie za posledný údaj
                writeln(F);      //nastav zápis na začiatok nového riadka
            end;
        writeln(F , Edit1.Text);         //zapíš údaj, ktorý nájdeš v Edit1.Text a prejdi na nový riadok
        writeln(F , Edit2.Text);         //zapíš údaj, ktorý nájdeš v Edit2.Text a prejdi na nový riadok
        write(F , Edit3.Text);           //zapíš údaj, ktorý nájdeš v Edit3.Text
        closefile(F);                    //zavri súbor
    end;
close;                           //zavri formulár
end;
end.
```

Pomerne jednoduché je naprogramovať položky Súbor – Otvor a Súbor – Premenuj. V prvej sme použili komponent OpenDialog zo záložky Dialogs, v druhej príkaz rename (súbor musí byť uzavretý).

```
procedure TfoPonuka.OtvorClick(Sender: TObject);
begin
if OpenDialog1.Execute
then NazovSuboru:= ExtractFileName(OpenDialog1.FileName);
end;

procedure TfoPonuka.PremenujClick(Sender: TObject);
begin
NazovSuboru:= InputBox( 'Premenuj' , 'Nový názov súboru (bez prípony)' , '' ) + '.txt';
rename(F , NazovSuboru);
assignfile(F , NazovSuboru);
end;
```

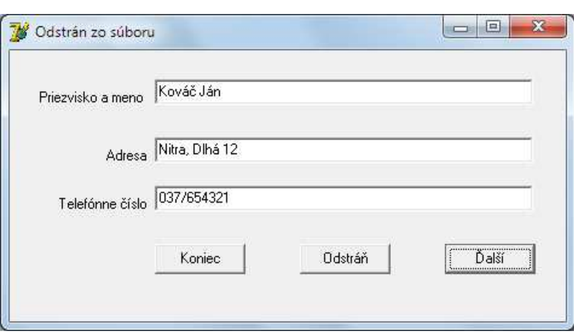



Pomerne komplikované je naprogramovanie položky Záznam - Edituj, ktorá má umožniť meniť údaje v uložených záznamoch. Princípom je čítanie záznamov zo súboru F (NazovSuboru) a ich zápis do súboru Pom (pomocny.txt) a na záver premenovanie súboru pomocny.txt na NazovSuboru. Tlačidlo Ďalší záznam sa po zobrazení posledného záznamu skryje a zobrazí sa tlačidlo Koniec.

```
implementation
{$R *.dfm}
uses Ponuka;
var FPom: TextFile;
    PrvyZapis: boolean;                    //opäť musíme rozlíšiť prvý zápis od ostatných zápisov
    PrM, Adr, TC: string;

procedure TfoEdituj.FormActivate(Sender: TObject);
begin
assignfile(FPom , 'pomocny.txt');
rewrite(FPom);
PrvyZapis:= True;

reset(F);
if not eof(F)
then begin
          readln(F,PrM);
          Edit1.Text:= PrM;
          readln(F, Adr);
          Edit2.Text:= Adr;
          readln(F, TC);
          Edit3.Text:= TC;
          btKoniec.Visible:= False;
      end
else btKoniec.Visible:= True;
end;

procedure TfoEdituj.btDalsiClick(Sender: TObject);              //tlačidlo Ďalší
begin
if PrvyZapis
then PrvyZapis:= False
else writeln(FPom);
writeln(FPom , Edit1.Text);
writeln(FPom , Edit2.Text);
write(FPom , Edit3.Text);

if not eof(F)
then begin
          readln(F, Prm);
          Edit1.Text:= PrM;
          readln(F, Adr);
          Edit2.Text:= Adr;
          readln(F, TC);
          Edit3.Text:= TC;
      end
else begin
          btDalsi.Visible:= False;
          btKoniec.Visible:= True;
     end;
end;
```

```
procedure TfoEdituj.btKoniecClick(Sender: TObject);          //tlačidlo Koniec
begin
closefile(F);
closefile(FPom);
erase(F);                            //vymaže súbor na disku
rename(FPom , NazovSuboru);          //premenuje na vymazaný
assignfile(F , NazovSuboru);         //pripraví F na ďalšie používanie v programe
close;                               //zavrie formulár Edituj záznam
end;

end.
```

Vo formulári Odstáň zo súboru, položka Záznam – Odstrániť, má užívateľ k dispozícii tri tlačidlá. Tlačidlo Ďalší najprv zapíše zobrazený záznam do pomocného súboru FPom a načíta ďalší záznam zo súboru F a zobrazí ho, tlačidlo Odstráň nezapíše aktuálny (zobrazený) záznam do pomocného súboru, čím ho odstráni zo súboru a na tlačidlo Koniec sa zapíšu do pomocného súboru všetky zvyšné neprezreté záznamy, aby sa nestratili. Na záver sa pomocný súbor premenuje na pracovný súbor.




```
implementation
{$R *.dfm}
uses Ponuka;

var FPom: TextFile;
    PrvyZapis: boolean;
    PrM, Adr, TC: string;

//procedúra, ktorá sa vykoná pri aktivácii formulára Odstráň, t.j. prvá po voľbe Záznam - Odstráň
procedure TfoOdstran.FormActivate(Sender: TObject);
begin
assignfile(FPom , 'pomocny.txt');
rewrite(FPom);                       //vytvorí prázny pomocný súbor
PrvyZapis:= True;
reset(F);
if not eof(F)                        //načíta sa a zobrazí prvý záznam z pracovného súboru
then begin
        readln(F,PrM);
        Edit1.Text:= PrM;
        readln(F, Adr);
        Edit2.Text:= Adr;
        readln(F, TC);
        Edit3.Text:= TC;
     end;
btKoniec.Visible:= True;
end;
```

```
//procedúra, ktorá sa vykoná po kliknutí na tlačidlo Ďalší
procedure TfoOdstran.btDalsiClick(Sender: TObject);
begin
if not eof(F)
then begin
         if PrvyZapis
         then PrvyZapis:= False
         else writeln(FPom);
         writeln(FPom,Edit1.Text);        //zapíše sa aktuálny záznam do pomocného súboru
         writeln(FPom,Edit2.Text);
         write(FPom,Edit3.Text);

         readln(F, Prm);                  //načíta sa a zobrazí ďalší záznam z pracovného súboru
         Edit1.Text:= PrM;
         readln(F, Adr);
         Edit2.Text:= Adr;
         readln(F, TC);
         Edit3.Text:= TC;
      end
else begin
         btDalsi.Visible:= False;
         btOdstran.Visible:= False;
     end;
end;

//procedúra, ktorá sa vykoná po kliknutí na tlačidlo Odstráň
procedure TfoOdstran.btOdstranClick(Sender: TObject);
begin
if MessageDlg('Odstrániť?',mtWarning, [mbYes, mbNo], 0) = mrNo
then begin                                //vykoná sa, ak sme na otázku Odstrániť? odpovedali NIE
         if PrvyZapis
         then PrvyZapis:= False
         else writeln(FPom);
         writeln(FPom,Edit1.Text);        //záznam sa zapíše do pomocného súboru
         writeln(FPom,Edit2.Text);
         write(FPom,Edit3.Text);
      end;
//else chýba, nemá sa nič zapísať, ak sme na otázku Odstrániť? odpovedali ÁNO

if not eof(F)
then begin                                //načíta sa  a zobrazí ďalší záznam z pracovného súboru
         readln(F, Prm);
         Edit1.Text:= PrM;
         readln(F, Adr);
         Edit2.Text:= Adr;
         readln(F, TC);
         Edit3.Text:= TC;
         btKoniec.Visible:= True;
      end
else begin                                //ak je koniec pracovného súboru
```

```
          closefile(F);                     //oba súbory sa zavrú
          closefile(FPom);
          erase(F);                         //odstráni sa pracovný súbor
          rename(FPom,NazovSuboru);         //premenovaním sa pomocný súbor stáva pracovným
          assignfile(F,NazovSuboru);
          close;                            //zavrie sa formulár Odstráň zo súboru
      end;
end;

//procedúra, ktorá sa vykoná po kliknutí na tlačidlo Koniec
procedure TfoOdstran.btKoniecClick(Sender: TObject);
begin
if PrvyZapis
then PrvyZapis:= False
else writeln(FPom);
writeln(FPom,Edit1.Text);                   //záznam sa zapíše do pomocného súboru
writeln(FPom,Edit2.Text);
write(FPom,Edit3.Text);
while not eof(F) do                         //do pomocného súboru sa zapíšu všetky záznamy
begin                                       //z pracovného súboru, ktoré ešte neboli zobrazené
          readln(F, Prm);
          readln(F, Adr);
          readln(F, TC);
          writeln(FPom);
          writeln(FPom,PrM);
          writeln(FPom,Adr);
          write(FPom,TC);
end;
closefile(F);                               //oba súbory sa zavrú
closefile(FPom);
erase(F);                                   //odstráni sa pracovný súbor
rename(FPom,NazovSuboru);                   //premenovaním sa pomocný súbor stáva pracovným
assignfile(F,NazovSuboru);
close;                                      //zavrie sa formulár Odstráň zo súboru
end;

end.
```

Kompletný primárny unit Ponuka (tučným písmom sme vyznačili časti dopísané do interfejsovej časti):

```
unit Ponuka;                                //pred premenovaním Unit1
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, StdCtrls, Oprograme, VypisSubor, EditujZaznam,  NovyZaznam, Odstran;
type
  TfoPonuka = class(TForm)
    MainMenu: TMainMenu;
    mmSubor: TMenuItem;                     //mmX znamená položka X MainMenu
```

```
    mmVypis: TMenuItem;
    mmKoniec: TMenuItem;
    mmZaznam: TMenuItem;
    mmEdituj: TMenuItem;
    mmOprograme: TMenuItem;
    mmOtvor: TMenuItem;
    OpenDialog1: TOpenDialog;
    mmNovy: TMenuItem;
    mmOdstran: TMenuItem;
    mmPremenuj: TMenuItem;
    N1: TMenuItem;
    procedure mmKoniecClick(Sender: TObject);
    procedure mmOprogrameClick(Sender: TObject);
    procedure mmVypisClick(Sender: TObject);
    procedure mmOtvorClick(Sender: TObject);
    procedure mmPremenujClick(Sender: TObject);
    procedure mmNovyClick(Sender: TObject);
    procedure mmEditujClick(Sender: TObject);
    procedure mmOdstranClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  foPonuka: TfoPonuka;                    //foPonuka je pred premenovaním Form1
  F: TextFile;
  NazovSuboru: string;

implementation
{$R *.dfm}

//ukončenie behu programu po kliknutí na položku Koniec
procedure TfoPonuka.mmKoniecClick(Sender: TObject);
begin
Application.Terminate;
end;

//vytvorenie formulára O programe
procedure TfoPonuka.mmOprogrameClick(Sender: TObject);
var Modal: TfoOprograme;
begin
Modal:= TfoOprograme.Create(Application);
try
  Modal.ShowModal;
finally
  Modal.Free;
end;
end;
```

```
//vytvorenie formulára Vypíš na vypísanie obsahu pracovného súboru do Memo
procedure TfoPonuka.mmVypisClick(Sender: TObject);
var Modal: TfoVypis;
begin
Modal:= TfoVypis.Create(Application);
try
  Modal.ShowModal;
finally
  Modal.Free;
end;
end;

//zobrazenie windowsovského okna Otvorenie a nastavenie nového pracovného súboru
procedure TfoPonuka.mmOtvorClick(Sender: TObject);
begin
if OpenDialog1.Execute
then NazovSuboru:= ExtractFileName(OpenDialog1.FileName);
assignfile(F,NazovSuboru);
end;

//premenovanie pracovného súboru
procedure TfoPonuka.mmPremenujClick(Sender: TObject);
begin
NazovSuboru:= InputBox( 'Premenuj' , 'Nový názov súboru (bez prípony)' , '' ) + '.txt';
rename(F,NazovSuboru);
assignfile(F,NazovSuboru);
end;

//vytvorenie formulára Nový na pridanie záznamu na koniec súboru
procedure TfoPonuka.mmNovyClick(Sender: TObject);
var Modal: TfoNovy;
begin
Modal:= TfoNovy.Create(Application);
try
  Modal.ShowModal;
finally
  Modal.Free;
end;
end;

//vytvorenie formulára Edituj záznam na úpravu záznamov z pracovného súboru
procedure TfoPonuka.mmEditujClick(Sender: TObject);
var Modal: TfoEdituj;
begin
Modal:= TfoEdituj.Create(Application);
try
  Modal.ShowModal;
finally
  Modal.Free;
end;
end;
```

```
//vytvorenie formulára Odstráň zo súboru na odstránenie záznamov z pracovného súboru
procedure TfoPonuka.mmOdstranClick(Sender: TObject);
var Modal: TfoOdstran;
begin
Modal:= TfoOdstran.Create(Application);
try
  Modal.ShowModal;
finally
  Modal.Free;
end;
end;

initialization
NazovSuboru:= 'zoznam.txt';          //názov pracovného súboru po spustení programu
assignfile(F,NazovSuboru);
try
  reset(F);                          //“pokus“ otvoriť súbor
except                               //nastala výnimka, súbor nebol nájdený v aktuálnom priečinku
  ShowMessage('Súbor nenájdený, vytváram nový!');
  rewrite(F);                        //vytvorenie prázdneho pracovného súboru
end;
closefile(F);

end.
```

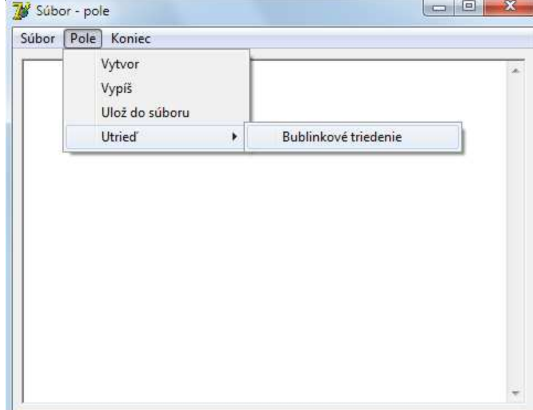
Ešte raz upozorňujeme, že v Delphi musíte mať **vypnuté** Tools – Debugger Options... – Language Exceptions – Stop on Delphi Exceptions, aby program nehavaroval v try - except, keď nenájde súbor zoznam.txt.

## Príklad 18.10

Vytvorte program umožňujúci uložiť vygenerované celočíselné pole do súboru a opačne, načítať celočíselné údaje z textového súboru do poľa na ďalšie spracovanie.

*Riešenie*

Na úvod musíme povedať, že štandardne sa údaje zo súboru neukladajú do poľa. Najmä pri databázach, teda rozsiahlejších skupinách údajov, by sa jednoducho nezmestili do operačnej pamäte. Preto existujú aj vonkajšie triediace algoritmy, ktoré sa na stredných školách neučia a slúžia na triedenie rozsiahlych súborov, ktoré sa naraz nezmestia do operačnej pamäte. Predpokladajme, že pracujeme len s neveľkou skupinou jednoduchých dát (maximálne niekoľko tisíc). Preto môžeme údaje uložiť do poľa a použiť aj vnútorné triediace algoritmy.




Zdokonalíme sa v používaní komponentu MainMenu. Na obrázku je ponuka obsahujúca položky Súbor (Vypíš a Ulož do poľa) a Pole (Vytvor, Vypíš, Ulož do súboru a Utrieď). Utrieď obsahuje submenu, zatiaľ s jedinou položkou, Bublinkové triedenie. Pri zvýraznenej položke Utrieď submenu vložíme vyvolaním kontextovej ponuky alebo stlačením Ctrl + šípka vpravo.

Program je pomerne jednoduchý, preto ho uvádzame bez podrobnejšieho komentára.

```
implementation
{$R *.dfm}
var F: TextFile;
    Pole: array of integer;

procedure TForm1.mmKoniecClick(Sender: TObject);
begin
Application.Terminate;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
try
        reset(F);                                   //existuje súbor data.txt?
except                                              //ak súbor neexistuje
        mmVypisSubor.Enabled:= false;               //vypni položku Vypíš súbor a
        mmUlozDoPola.Enabled:= false;               //vypni položku Ulož do poľa
end;
mmVypisPole.Enabled:= false;                        //vypni položky Vypíš pole,
mmUlozDoSuboru.Enabled:= false;                     //              Ulož do súboru a
mmUtried.Enabled:= false;                           //              Utrieď
end;

procedure TForm1.mmVytvorPoleClick(Sender: TObject);
var Pocet, i: integer;
begin
Pocet:= StrToInt( InputBox( 'Vytvor' , 'Počet prvkov' , '100' ) );
SetLength(Pole, Pocet);
for i := 0 to High(Pole) do Pole[i]:= random(100);
mmVypisPole.Enabled:= true;                         //zapni položky Vypíš pole a
mmUlozDoSuboru.Enabled:= true;                      //              Utrieď
mmUtried.Enabled:= true;
end;

procedure TForm1.mmVypisPoleClick(Sender: TObject);
var i: integer;
    Riadok: string;
begin
Memo1.Lines.Add( '-------------------------------------' );
Riadok:= IntToStr(Pole[0]);
for i := 1 to High(Pole) do Riadok:= Riadok + Format( '%3d' , [ Pole[i] ] );
Memo1.Lines.Add(Riadok);
Memo1.Lines.Add('--------------------------------------');
end;

procedure TForm1.mmUlozDoSuboruClick(Sender: TObject);
var i: integer;
begin
rewrite(F);
for i:= 0 to High(Pole) do write( F , ' ' , Pole[i] );     //čísla budú zapísané do riadka, oddelené medzerami
```

```
closefile(F);
mmVypisSubor.Enabled:= true;
mmUlozDoPola.Enabled:= true;
end;

procedure TForm1.mmVypisSuborClick(Sender: TObject);
begin
Memo1.Lines.LoadFromFile( 'data.txt' );
end;

procedure TForm1.mmUlozDoPolaClick(Sender: TObject);
var X, Index: integer;
begin
reset(F);
Index:= 0;
while not eof(F) do
begin
        read(F, X);
        SetLength(Pole, Index+1);
        Pole[Index]:= X;
        inc(Index);
end;
closefile(F);
mmVypisPole.Enabled:= true;
mmUlozDoSuboru.Enabled:= true;
mmUtried.Enabled:= true;
end;

procedure TForm1.mmBublinkoveClick(Sender: TObject);
var Prechod, i: integer;
    Pom: integer;
begin
for Prechod:= 1 to High(Pole) do
        for i:= 0 to High(Pole) - Prechod do
                if Pole[i] > Pole[i+1]
                then begin Pom:= Pole[i]; Pole[i]:= Pole[i+1]; Pole[i+1]:= Pom; end;
end;

initialization
randomize;
assignfile(F,'data.txt');
end.
```

# Dvojrozmerné pole

V definíciách typu pole:

type *MenoTypu* = array [ *TypIndexu* ] of *TypZložky* ; alebo type *MenoTypu* = array of *TypZložky*;
sme doteraz dosadzovali za typ zložky len jednoduchý typ alebo typ reťazec. V princípe typ zložky môže byť akýkoľvek, okrem typu súbor. V prípade, že typ zložky bude opäť typ pole, dostávame tzv. typ dvojrozmerné pole.

Definícia typu dvojrozmerné **statické** pole má tvar:

type mt = array [ ti1 ] of array [ ti2 ] of tz;

kde mt je meno typu – identifikátor, ti1 a ti2 sú ordinálne typy indexov typu a tz je typ zložky.
Dovolený je aj skrátený zápis definície typu pole:

type mt = array [ ti1 , ti2 ] of tz;

Napríklad:

```
type     tMatica1 = array [1..10 ] of array [1..10] of integer;
         tMatica2 = array [ 1..10 , 1..10 ] of integer;            // definícia rovnocenná s tMatica1
         tSachovnica =array[ ´a´..´h´ , 1..8 ] of boolean;
         tNezmysel =array[ boolean , boolean ] of char;
```

Definícia typu dvojrozmerné **dynamické** pole má tvar:

type mt = array of array of tz;

kde mt je meno typu – identifikátor a tz je typ zložky.
Pre prácu s dvojrozmerným dynamickým poľom platia rovnaké pravidlá a procedúry ako pre prácu s jednorozmerným dynamickým poľom, t.j. predovšetkým veľkosť poľa treba nastaviť pred jeho prvým použitím príkazom SetLength (pozri príklad nižšie).

Dvojrozmerné pole nazývame aj pole polí. Názov vznikol zrejme z poznatku, že jednorozmerné pole, resp. každý jeho prvok je znova jednorozmerným poľom. Z hľadiska štruktúry dvojrozmernému poľu zodpovedá matematický pojem matica.

Matica typu N x M („en krát em“) obsahuje N riadkov a M stĺpcov

- pre zmenu riadku, t.j. pohyb v stĺpci používame premenné Riadok, Row, I
- pre zmenu stĺpca, t.j. pohyb v riadku používame premenné Stlpec, Col, J
- schematicky (matica typu N x M):

*Stlpec sa mení od 1 po M (pohyb v riadku, zmena stĺpca)* →

*Riadok sa mení od 1 po N (pohyb v stĺpci, zmena riadku)* ↓

| | | | | | | |
|---|---|---|---|---|---|---|
| $a_{1,1}$ | $a_{1,2}$ | $a_{1,3}$ | … | $a_{1,j}$ | … | $a_{1,m}$ |
| $a_{2,1}$ | $a_{2,2}$ | $a_{2,3}$ | … | $a_{2,j}$ | … | $a_{2,m}$ |
| $a_{3,1}$ | $a_{3,2}$ | $a_{3,3}$ | … | $a_{3,j}$ | … | $a_{3,m}$ |
| … | … | … | … | … | … | … |
| $a_{i,1}$ | $a_{i,2}$ | $a_{i,3}$ | … | $a_{i,j}$ | … | $a_{i,m}$ |
| … | … | … | … | … | … | … |
| $a_{n,1}$ | $a_{n,2}$ | $a_{n,3}$ | … | $a_{n,j}$ | … | $a_{n,m}$ |

Z hľadiska algoritmických konštrukcií pre prácu s maticou väčšinou potrebujeme cyklus v cykle. Vonkajší cyklus zabezpečuje najčastejšie nastavenie príslušného riadku (for Riadok := 1 to N do) a vnútorný cyklus pohyb v nastavenom riadku t.j. zmenu stĺpca od 1 po M (for Stlpec := 1 to M do).

Aj pri dvojrozmerných poliach pracujeme s indexovanými premennými a používame zápis, napríklad pre pole A: A[i][j] resp. skrátený zápis A[i,j] - prvok poľa A v i-tom riadku a j-tom stĺpci.

### Príklad 19.1

Demonštračné vytvorenie dvojrozmerného dynamického poľa.

```
var A: array of array of string;

procedure TForm1.btZobrazClick(Sender: TObject);
var  i, j : integer;  Riadok: string;
begin
SetLength(A, 5);                  //počet riadkov poľa A
for i:= 0 to High(A) do
begin
   SetLength(A[i], i+1);          //počet stĺpcov v riadku
   Riadok:= '';
   for j:= 0 to High(A[i]) do
   begin
        A[i,j] := IntToStr(i) + ',' + IntToStr(j) + '      ';
        Riadok:= Riadok + A[i,j];
   end;
   Memo1.Lines.Add(Riadok);
end;
end;
```

Form1

```
0,0
1,0   1,1
2,0   2,1   2,2
3,0   3,1   3,2   3,3
4,0   4,1   4,2   4,3   4,4
```

Zobraz

Výpis vznikne spustením procedúry btZobrazClick

### Príklad 19.2

Napíšte procedúru, ktorá vytvorí maticu 5 riadkov x 6 stĺpcov obsahujúcu najviac dvojciferné nezáporné celé čísla.

*Riešenie*

```
const MAXRia = 5;
        MAXStl = 6;
type   tMatica = array [1..MAXRia,1..MAXStl] of integer;
var     A: tMatica;

procedure TForm1.btVytvorClick(Sender: TObject);
var Ria, Stl: integer;
begin
for Ria:= 1 to MAXRia do
        for Stl:= 1 to MAXStl do A[Ria,Stl]:= random(100);
end;

initialization
randomize;
end.
```

### Príklad 19.3

Napíšte procedúru, ktorá vypíše prvky dvojrozmerného poľa z predchádzajúceho príkladu vo forme tabuľky (matice).

*Riešenie*

Procedúru sme doplnili o dialógovú funkciu MessageDlg, ktorá sa opýta, či zmazať Memo.
Najprv sa v premennej Riadok typu string vytvorí riadok matice a na záver vypíše do Mema. Použili sme príkaz `Str`, ktorý hodnotu z číselnej premennej (`A[Ria,Stl]`) uloží do premennej (`pomr`) typu string so zadaným počtom znakov resp. zľava doplní medzerami.

```
procedure TForm1.btVypisClick(Sender: TObject);
var Ria, Stl: integer;
    Riadok,pomr: string;
begin
if MessageDlg('Zmazať Memo?',mtConfirmation, [mbYes, mbNo], 0) = mrYes
then Memo1.Lines.Clear
else Memo1.Lines.Add('');
for Ria:= 1 to MAXRia do
begin
        Riadok:= '';
        for Stl:= 1 to MAXStl do
        begin
                if Stl = 1
                then Str(A[Ria,Stl]:2 , pomr)    // musí byť nastavené v Memo písmo Courier, aby boli
                else Str(A[Ria,Stl]:5 , pomr);   // všetky znaky (aj medzery) rovnako široké!!!
                Riadok:= Riadok + pomr
        end;
        Memo1.Lines.Add(Riadok)
end;
end;
```

### Príklad 19.4

Napíšte procedúru, ktorá nájde a vypíše najväčšie číslo v každom riadku matice. Využite predchádzajúce príklady na vytvorenie a vypísanie dvojrozmerného poľa.

*Riešenie*

Uvedomte si rozdiel medzi nájdením maxima v jednorozmernom poli a „vo viacerých jednorozmerných poliach pod sebou“.

```
procedure TForm1.btMaxVRiadkuClick(Sender: TObject);
var Ria, Stl, Max: integer;
begin
for Ria:= 1 to MAXRia do                    // rieš 1., 2.,... jednorozmerné pole
begin                                       // odtiaľto je Ria „konštanta“ mení sa len Stl
        Max:= A[Ria,1];                     // nastavenie počiatočnej hodnoty – prvý prvok v riadku
        for Stl:= 2 to MAXStl do if A[Ria,Stl]> Max then Max:= A[Ria,Stl];
        Memo1.Lines.Add('Maximum v ' + IntToStr(Ria) + '. riadku: ' + IntToStr(Max));
end;
end;
```

### Príklad 19.5

Napíšte procedúru, ktorá nájde a vypíše najväčšie číslo v každom stĺpci matice. Využite predchádzajúce príklady na vytvorenie a vypísanie dvojrozmerného poľa.

*Riešenie*

Algoritmus je rovnaký, ako v predchádzajúcom príklade, len prechádzame zvisle maticou, t.j. zvyšujeme riadok pri pohybe v stĺpci.

```
procedure TForm1.btMaxVStlpciClick(Sender: TObject);
var Ria, Stl, Max: integer;
begin
for Stl:= 1 to MAXStl do                    // nastavenie stĺpca, ktorý skúmame
begin
        Max:= A[1,Stl];                     // nastavenie počiatočnej hodnoty – prvý prvok v stĺpci
```

```
        for Ria:= 2 to MAXRia do if A[Ria,Stl]> Max then Max:= A[Ria,Stl];
        Memo1.Lines.Add('Maximum v ' + IntToStr(Stl) + '. stĺpci: ' + IntToStr(Max));
end;
end;
```

### Príklad 19.6

Vytvorte program, ktorý utriedi prvky v jednotlivých riadkoch matice. V obrázku je pôvodná matica a matica po utriedení prvkov v riadkoch.

Matica

```
45   8  13  55  76  91
45  34  99  67  92   9
15  65   1   5  78  37
82  21  97  98  37  64
94  57  33  25  51  23

 8  13  45  55  76  91
 9  34  45  67  92  99
 1   5  15  37  65  78
21  37  64  82  97  98
23  25  33  51  57  94
```

*Riešenie*

Napíšeme si univerzálnu procedúru Bubblesort pre utriedenie jednorozmerného poľa.

```
procedure Bubblesort(var Pole: array of integer);
var i, PP: integer;
    Pom: integer;
begin
for PP:=1 to High(Pole) do
        for i:= Low(Pole) to High(Pole) - PP do
                if Pole[i]>Pole[i+1]
                then begin
                        Pom:= Pole[i]; Pole[i]:= Pole[i+1]; Pole[i+1]:= Pom;
                     end;
end;
```

Využijeme fakt, že matica typu NxM je jednorozmerné pole s N prvkami (N riadkov), pričom každý riadok je jednorozmerné pole, ktoré chceme utriediť. Vyplýva to z definície dvojrozmerného poľa ( ...array [riadky] of array [stĺpce] of...). Jednoduché riešenie:

```
procedure TForm1.btTriedVRiadkuClick(Sender: TObject);
var Ria: integer;
begin
for Ria:= 1 to MAXRia do Bubblesort(A[Ria]);          // pozor, A[Ria] je jeden konkrétny riadok!
end;
```

Dvojrozmerné pole, pre naše potreby, by sme mohli definovať aj nasledovne:

```
const MAXRia = 5;
      MAXStl = 6;
type  tStlpec = array [1..MAXStl] of integer;
      tMatica = array [1..MAXRia] of tStlpec;
var   A: tMatica;
```

Potom by parametrom v procedúre Bubblesort mohol byť aj typ tStlpec, t.j. zápis var Pole: tStlpec a mohli by sme v cykloch procedúry použiť konštantu MAXStl.

„O 90° otočený pohľad“ neplatí, t.j. matica nie je jednorozmerné pole prvého riadka, v ktorom je každý prvok jednorozmerným poľom v smere stĺpca. Preto v nasledujúcom príklade už musíme postupovať „klasicky“.

### Príklad 19.7

Vytvorte program, ktorý utriedi prvky v jednotlivých stĺpcoch matice. V obrázku je pôvodná matica a matica po utriedení prvkov v stĺpcoch.

*Riešenie*

```
procedure TForm1.btTriedVStlpciClick(Sender: TObject);
var Ria, Stl, PP: integer;
     Pom: integer;
begin
for Stl:= 1 to MAXStl do
begin
          for PP:= 1 to MAXRia - 1 do
                    for Ria:= 1 to MAXRia - PP do
                              if A[Ria,Stl]> A[Ria+1,Stl]
                              then begin
                                        Pom:= A[Ria, Stl]; A[Ria,Stl]:= A[Ria+1,Stl]; A[Ria+1,Stl]:= Pom;
                                   end;
end;
end;
```

```
Matica
22  51  58  19  99  63
60  68  24  57  27  61
 7  60  15  35  27  93
59   2  97  63  56  53
49   9  24  35   5  38

 7   2  15  19   5  38
22   9  24  35  27  53
49  51  24  35  27  61
59  60  58  57  56  63
60  68  97  63  99  93
```

## Príklad 19.8

Vytvorte program, ktorý načíta celočíselné údaje zapísané v textovom súbore vo forme tabuľky do dvojrozmerného poľa.

*Riešenie*

Nech textový súbor obsahuje napríklad riadky

```
0    1    0    0    0    1
1    0    1    0    1    1
0    1    0    1    1    0
0    0    1    0    1    0
0    1    1    1    0    1
1    1    0    0    1    0
```

Môžeme sa rozhodnúť pre statické alebo dynamické dvojrozmerné pole. Ak to je možné, kvôli jednoduchosti aj efektívnosti, používame statické dvojrozmerné pole. Vytvorenie aj naplnenie poľa hodnotami sme naprogramovali v inicializačnej časti.

Uvedieme celú implementačnú časť

```
implementation
{$R *.dfm}
const   N = 6;
var     M: array [ 1..N , 1..N ] of 0..1;
        Ria, Stl: 1..N;
        F: TextFile;

initialization
assignfile( F , 'subor.txt' );
reset(F);
for Ria:= 1 to N do
        for Stl:= 1 to N do
                read( F , M[ Ria , Stl ] );
closefile(F);
end.
```

Pri použití dynamického dvojrozmerného poľa by deklarácia poľa mala tvar
`var M: array of array of 0..1;`

a inicializačná časť obsahovala príkazy:

```
initialization
assignfile(F , 'subor.txt' );
reset(F);
Ria:= 0;
while not seekeof(F) do
begin
        setlength( M , length(M) + 1 );
        Stl:= 0;
        while not seekeoln(F) do
        begin
                setlength( M[i] , length(M[i]) + 1 );
                read( F , M[i , j] );
                inc(Stl);
        end;
        readln(F);
        inc(Ria);
end;
closefile(F);
```

### Príklad 19.9

Aj dvojrozmerné pole môže byť zadané vymenovaním prvkov v úseku deklarácií alebo ako konštantné pole.
Pole z príkladu 19.8 by bolo zadané

```
const   N = 6;
var     M: array [1..N,1..N] of integer = ((0,1,0,0,0,1),
                                           (1,0,1,0,1,1),
                                           (0,1,0,1,1,0),
                                           (0,0,1,0,1,0),
                                           (0,1,1,1,0,1),
                                           (1,1,0,0,1,0));
```

alebo ako konštantné dvojrozmerné pole

```
const   N = 6;
        M: array [1..N,1..N] of integer = ((0,1,0,0,0,1),
                                           (1,0,1,0,1,1),
                                           (0,1,0,1,1,0),
                                           (0,0,1,0,1,0),
                                           (0,1,1,1,0,1),
                                           (1,1,0,0,1,0));
```

Rozdiel v použití konštantného a deklarovaného poľa sme uviedli pri jednorozmernom poli.

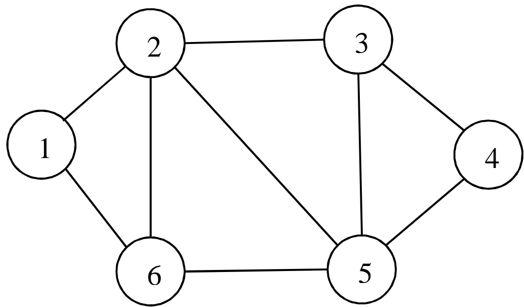
# Grafy

**Graf** je sústava bodov (tzv. uzlov, vrcholov), medzi ktorými vedú spojnice (hrany grafu).
Uvažujeme len konečné grafy. Graf je určený počtom vrcholov grafu a dvojicami vrcholov, ktoré sú spojené hranou. Grafy môžu byť **neorientované** a **orientované**. V orientovaných grafoch majú hrany priradenú orientáciu, t.j. z ktorého do ktorého vrcholu vedú (v nakreslenom grafe vyznačujeme šípkou – vektorom). Grafy môžu byť **neohodnotené** a **ohodnotené**. V ohodnotenom grafe je každej hrane priradené tzv. ohodnotenie, čo je väčšinou jedno číslo, napr. dĺžka hrany. **Cestou** z vrcholu X do vrcholu Y rozumieme takú postupnosť hrán $h_1$, $h_2$, ..., $h_n$, že hrana $h_1$ vychádza z vrcholu X, vedie do vrcholu, z ktorého vychádza hrana $h_2$, ... až hrana $h_n$ vedie do vrcholu Y. Cestami sa zaoberať nebudeme.

### Reprezentácia grafu v programe

Máme graf s N vrcholmi a M hranami (N, M konštanty). Vrcholy grafu sú označené číslami od 1 po N, prípadne hrany číslami od 1 po M.

Príklad neorientovaného neohodnoteného grafu (počet vrcholov N = 6, počet hrán M = 9)




### Matica susednosti

umožňuje reprezentovať neohodnotené neorientované aj orientované grafy. Priesečník riadka – jeden vrchol a stĺpca – druhý vrchol obsahuje informáciu, či sú vrcholy spojené hranou. Ide o maticu veľkosti N x N s hodnotami false/true alebo 0/1. Pri neorientovaných grafoch je matica susednosti symetrická.
Príklad deklarácie var M: array [1..N, 1..N] of 0..1;

## Príklad 20.1

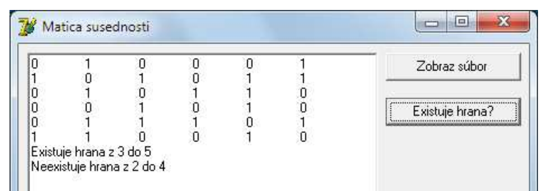
Vytvorte reprezentáciu vyššie zobrazeného grafu pomocou matice susednosti a procedúru na zistenie, či medzi zadanými vrcholmi existuje hrana.
*Riešenie*
Matica susednosti má tvar

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 1 | (z 1 vedie hrana do 2 a 6) |
| 1 | 0 | 1 | 0 | 1 | 1 | (z 2 vedie hrana do 1, 3, 5 a 6) |
| 0 | 1 | 0 | 1 | 1 | 0 | (z 3 vedie hrana do 2, 4 a 5) |
| 0 | 0 | 1 | 0 | 1 | 0 | (zo 4 vedie hrana do 3 a 5) |
| 0 | 1 | 1 | 1 | 0 | 1 | (z 5 vedie hrana do 1, 3, 4 a 6) |
| 1 | 1 | 0 | 0 | 1 | 0 | (zo 6 vedie hrana do 1, 2, a 5) |

Kódovanie hrán číslami 0 a 1 sme zvolili kvôli tomu, že hodnoty true a false nemožno čítať z textového súboru, ak by bol graf uložený v textovom súbore vo forme tabuľky.

Ako sa matica susednosti dostane do dvojrozmerného poľa sme vyriešili v príkladoch 19.8 a 19.9. Zistiť, či medzi zadanými vrcholmi existuje hrana znamená zistiť, či v priesečníku zadaného riadka (odkiaľ) a stĺpca (kam) je hodnota 0 alebo 1.

```
procedure TForm1.btHranaClick(Sender: TObject);
var Odkial, Kam: 1..N;
begin
Odkial:= StrToInt(InputBox('Matica susednosti','Odkial',''));
Kam:= StrToInt(InputBox( 'Matica susednosti' , 'Kam' , '' ));
if M[ Odkial , Kam ] = 1
then Memo1.Lines.Add(Format( 'Existuje hrana z %d do %d' , [ Odkial , Kam ] ))
else Memo1.Lines.Add(Format( 'Neexistuje hrana z %d do %d' , [ Odkial , Kam ] ));
end;
```




**Matica vzdialeností**
je analógiou matice susednosti s tým rozdielom, že v matici sú uložené ohodnotenia jednotlivých hrán. Ak hrana medzi príslušnými vrcholmi neexistuje, jej ohodnotenie je napríklad 0 (nula). Matica vzdialeností reprezentuje ohodnotené grafy.
Príklad deklarácie var M: array [1..N, 1..N] of integer;

## Príklad 20.2

V textovom súbore cesty.txt je daná cestná sieť – ohodnotený graf vo forme tabuľky. Vytvorte program, ktorý

a) skontroluje, či sú všetky cesty obojsmerné,
b) vypíše celkovú dĺžku cestnej siete,
c) vypíše dĺžku najdlhšej spojnice (hrany) a ktoré mestá spája,
d) vypíše mesto, z ktorého vychádza najviac spojníc,
e) vypíše mesto, z ktorého vychádza najviac spojníc, ak existuje viacej takýchto miest, vypíše to, pre ktoré je súčet ciest najväčší.

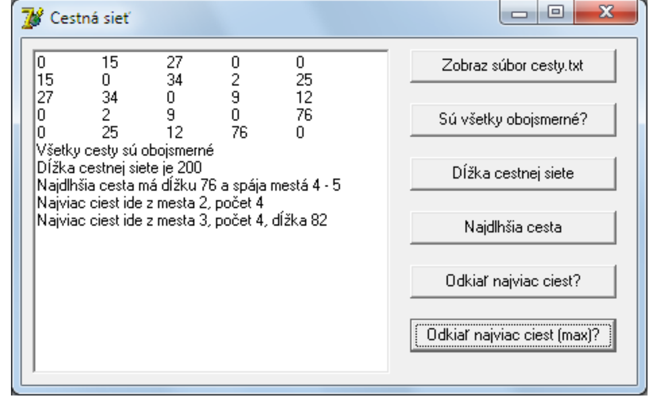
Ak má úloha viac riešení, stačí vypísať ľubovoľné z nich. Môžete predpokladať, že všetky cesty sú obojsmerné.

*Analýza*

Nech súbor cesty.txt obsahuje hodnoty

```
0    15   27   0    0
15   0    34   2    25
27   34   0    9    12
0    2    9    0    76
0    25   12   76   0
```

Odpovede
a) všetky cesty sú obojsmerné
b) 200
c) 76, mestá 4 - 5
d) mesto 2 alebo 3, počet spojníc 4
e) mesto 3, počet spojníc 4, dĺžka 82




Zistiť, či sú všetky cesty obojsmerné znamená zistiť, či je matica symetrická podľa hlavnej diagonály, t.j. či sa M[Ria,Stl] = M[Stl,Ria] pre všetky riadky a stĺpce matice. Zrejme stačí zobrať indexy z trojuholníka [2 , 1], [N , 1] a [N , N-1], t.j. od druhého po posledný riadok a v každom riadku len k hlavnej diagonále.
Zistiť celkovú dĺžku cestnej siete znamená sčítať všetky hodnoty v tabuľke a vydeliť ich dvoma, alebo opäť sčítať len hodnoty „z dolného trojuholníka".
Zistiť najdlhšiu spojnicu a ktoré mestá spája znamená nájsť najväčšiu hodnotu v tabuľke a zistiť, v ktorom je riadku a stĺpci.
Zistiť mesto, z ktorého vychádza najviac spojníc, znamená nájsť riadok, v ktorom je najviac nenulových hodnôt. Ak existuje takýchto riadkov viacej, treba zistiť súčet týchto hodnôt a vypísať riadok, v ktorom je súčet najväčší.

*Riešenie*

```
const N = 5;
var     M: array [ 1..N , 1..N ] of integer;

//načítanie tabuľky z textového súboru do matice s ošetrením možnej výnimky
procedure TForm1.FormCreate(Sender: TObject);
var Ria, Stl: integer;
    F: TextFile;
begin
assignfile(F,'cesty.txt');
try
        reset(F);
except
        ShowMessage( 'Súbor cesty.txt nenájdený!' );
        Application.Terminate;
end;

for Ria:= 1 to N do
        for Stl:= 1 to N do
                read( F , M[Ria,Stl] );
closefile(F);
end;

//zobrazenie súboru cesty.txt
procedure TForm1.btZobrazSuborClick(Sender: TObject);
begin
Memo1.Lines.LoadFromFile('cesty.txt');
end;

//zistenie, či existuje jednosmerná cesta
{
Zistiť, či sú všetky cesty obojsmerné znamená zistiť, či je matica symetrická podľa hlavnej diagonály, t.j. či sa M[Ria,Stl] = M[Stl,Ria] pre všetky riadky a stĺpce matice. Zrejme stačí zobrať indexy z trojuholníka [2 , 1],  [N , 1] a [N , N-1], t.j. od druhého po posledný riadok a v každom riadku len k hlavnej diagonále.
}
procedure TForm1.btObojsmerneClick(Sender: TObject);
        function ExistJednosmerna: boolean;
        var Ria, Stl: integer;
        begin
        Result:= False;
        for Ria:= 2 to N do
                for Stl:= 1 to Ria-1 do
                        if M[Ria, Stl] <> M[Stl, Ria] then Result:= True;
        end;
begin
if ExistJednosmerna
then Memo1.Lines.Add( 'Existuje jednosmerná cesta' )
else Memo1.Lines.Add( 'Všetky cesty sú obojsmerné' )
end;
```

```
//zistenie dĺžky cestnej siete
{
Zistiť celkovú dĺžku cestnej siete znamená sčítať všetky hodnoty „z dolného trojuholníka“.
}
procedure TForm1.btDlzkaClick(Sender: TObject);
var Dlzka, Ria, Stl: integer;
begin
Dlzka:= 0;
for Ria:= 2 to N do
        for Stl:= 1 to Ria-1 do Dlzka:= Dlzka + M[Ria, Stl];
Memo1.Lines.Add( Format( 'Dĺžka cestnej siete je %d' , [ Dlzka ] ) );
end;

//zistenie najdlhšej cesty
{
Zistiť najdlhšiu spojnicu a ktoré mestá spája znamená nájsť najväčšiu hodnotu v matici a zistiť, v ktorom je riadku a stĺpci. Môžeme využiť symetrickosť matice a prejsť len napríklad „horný trojuholník“ matice (nad hlavnou diagonálou).
}
procedure TForm1.btNajdlhsiaClick(Sender: TObject);
var Max, Odkial, Kam, Ria, Stl: integer;
begin
Max:= -1;
for Ria:= 1 to N-1 do
        for Stl:= Ria + 1 to N do
                if M[Ria, Stl] > Max
                then begin
                        Max:= M[Ria, Stl];
                        Odkial:= Ria; Kam:= Stl;
                     end;
Memo1.Lines.Add(Format( 'Najdlhšia cesta má dĺžku %d a spája mestá %d - %d' , [Max, Odkial, Kam] ));
end;

//zistenie mesta, z ktorého vychádza najviac spojníc
{
Zistiť mesto, z ktorého vychádza najviac spojníc, znamená nájsť riadok, v ktorom je najviac nenulových hodnôt.
}
procedure TForm1.btNajviac1Click(Sender: TObject);
var Ria, Stl, Pocet, MaxPocet, MaxMesto: integer;
begin
MaxPocet:= -1;
for Ria:= 1 to N do
begin
        Pocet:= 0;                  //na začiatku prehľadávania riadka je počet nenulových spojníc 0
        for Stl:= 1 to N do         //prechod riadkom
                if M[Ria, Stl] > 0 then inc(Pocet);
        if Pocet > MaxPocet         //ak je počet spojníc z ostatného riadka väčší ako dovt. najväčší
        then begin
                MaxPocet:= Pocet;   //nový najväčší počet nenulových hodnôt v riadku pre maticu
                MaxMesto:= Ria;     //zapamätanie, v ktorom to je riadku
```

```
          end;
end;
Memo1.Lines.Add(Format( 'Najviac ciest ide z mesta %d, počet %d' , [ MaxMesto , MaxPocet ] ));
end;

//zistenie mesta, z ktorého vychádza najviac spojníc a súčet ich dĺžok je najväčší
{
```

Zistiť mesto, z ktorého vychádza najviac spojníc, znamená nájsť riadok, v ktorom je najviac nenulových hodnôt. Ak existuje takýchto riadkov viacej, treba zistiť súčet týchto hodnôt a vypísať číslo riadka, v ktorom je súčet najväčší.

```
}
procedure TForm1.btNajviac2Click(Sender: TObject);
var Ria, Stl, Pocet, MaxPocet, MaxMesto, Sucet, MaxSucet: integer;
begin
MaxPocet:= -1; MaxSucet:= 0;          //nastavenie počiat. hodnôt pred prechodom maticou
for Ria:= 1 to N do                                //pre 1., 2.,..., posledný riadok zisti...
begin
      Pocet:= 0; Sucet:= 0;            //nastavenie počiat. hodnôt pred prechodom riadkom
      for Stl:= 1 to N do                          //prejdi riadkom a zisti...
            if M[Ria, Stl] > 0
            then begin
                  inc(Pocet);                      //počet nenulových hodnôt a
                  Sucet:= Sucet + M[Ria, Stl];     //súčet týchto hodnôt
                end;
      if (Pocet > MaxPocet) or (Pocet = MaxPocet) and (Sucet > MaxSucet)
      then begin
            MaxPocet:= Pocet;      //nový najväčší počet nenulových hodnôt v riadku pre maticu
            MaxSucet:= Sucet;      //nový najväčší súčet nenulových hodnôt v riadku pre maticu
            MaxMesto:= Ria;        //zapamätanie, v ktorom to je riadku
          end;
end;
Memo1.Lines.Add(Format( 'Najviac ciest ide z mesta %d, počet %d, dĺžka %d' , [ MaxMesto, MaxPocet,
MaxSucet ] ));
end;
```

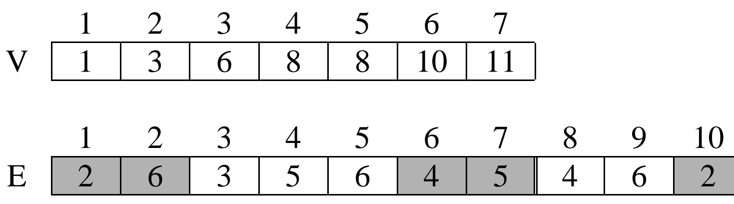
Všetky z uvedených úloh príkladu 20.2 sa nemusia riešiť pomocou dvojrozmerného poľa, prakticky len prvá, ostatné by sa dali vyriešiť len prechodom textovým súborom. Vo všeobecnosti úlohy, v ktorých vystačíme s čítaním údajov po riadkoch, sa dajú riešiť prechodom textovým súborom bez načítania hodnôt do dvojrozmerného poľa.

**Zoznam nasledovníkov**

Ku každému vrcholu grafu si pamätáme zoznam tých vrcholov, do ktorých z neho vedie hrana. Reprezentácia pracuje s dvoma poliami: pole E o veľkosti M obsahuje čísla všetkých nasledovníkov, pole V o veľkosti N+1 obsahuje indexy určujúce, kde sú v poli E uložení nasledovníci toho - ktorého vrcholu.

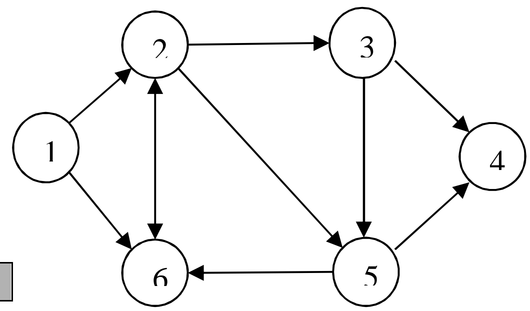
Zoznam nasledovníkov neumožňuje reprezentovať ohodnotené grafy, je však vhodný na zisťovanie ciest medzi vrcholmi.

Napríklad grafu vpravo zodpovedá nasledujúci zoznam nasledovníkov:



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| V | 1 | 3 | 6 | 8 | 8 | 10 | 11 |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| E | 2 | 6 | 3 | 5 | 6 | 4 | 5 | 4 | 6 | 2 |

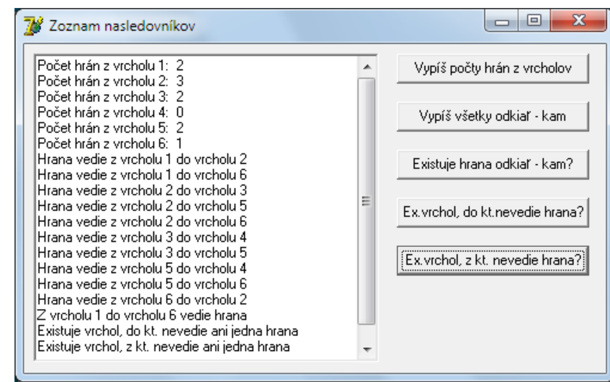



Napríklad vo V [1] je hodnota, udávajúca, kde začínajú vrcholy v poli E, do ktorých vedie hrana z vrcholu 1,
resp.
vo V [Vrchol] je hodnota, udávajúca, kde začínajú vrcholy v poli E, do ktorých vedie hrana z vrcholu Vrchol.
V E [1] až E [ V [2] – 1 ] sú všetky vrcholy, do ktorých vedie hrana z vrcholu 1,
resp.
v E [V [ Vrchol ] ] až E [ V [ Vrchol + 1 ] - 1 ] sú všetky vrcholy, do ktorých vedie hrana z vrcholu Vrchol.

### Príklad 20.3

Vytvorte program, ktorý pomocou zoznamu nasledovníkov grafu vyššie
a) vypíše počty hrán z jednotlivých vrcholov,
b) vypíše všetky dvojice vrcholov, medzi ktorými existujú hrany,
c) vypíše, po zadaní dvoch vrcholov, či existuje hrana vedúca z prvého do druhého vrcholu,
d) vypíše, či existuje vrchol, do ktorého nevedie ani jedna hrana,
e) vypíše, či existuje vrchol, z ktorého nevychádza ani jedna hrana.

*Analýza*
Na obrázku vpravo je možná realizácia úlohy.
Graf má 6 vrcholov (N = 6) a 10 hrán (M = 10). Hodnoty polí V a E je vhodné zadať už pri ich deklarácii.
Analýzu jednotlivých úloh uvádzame až pred procedúrami, ktoré ich riešia.




*Riešenie*

```
const   N = 6;                    //počet vrcholov
        M = 10;                   //počet hrán
var     V: array [1..N+1] of integer = (1,3,6,8,8,10,11);
        E: array [1..M] of integer = (2,6,3,5,6,4,5,4,6,2);
```

//a) Rozdiel susedných hodnôt poľa V zrejme udáva počty hrán z jednotlivých vrcholov

```
procedure TForm1.btPoctyHranClick(Sender: TObject);
var Vrchol: integer;
begin
for Vrchol:= 1 to N do
        Memo1.Lines.Add(Format( 'Počet hrán z vrcholu %d:  %d' , [ Vrchol , V[Vrchol+1] - V[Vrchol] ]));
end;
```

//b) vypíše všetky dvojice vrcholov, medzi ktorými existujú hrany
{
Pole E obsahuje vrcholy, do ktorých vedú hrany postupne z vrcholu 1, potom 2, 3 atď.;
príslušné vrcholy sú v poli E pre vrchol 1 na indexoch od V[1] až po V[2]-1, pre vrchol 2 na indexoch od V[2] po V[3]-1, všeobecne pre Vrchol na indexoch od V[Vrchol] až po V[Vrchol+1]-1
}

```
procedure TForm1.btOdkialKamClick(Sender: TObject);
var Vrchol, i: integer;
begin
for Vrchol:= 1 to N do
        for i:= V[Vrchol] to V[Vrchol+1]-1 do
                Memo1.Lines.Add(Format( 'Hrana vedie z vrcholu %d do vrcholu %d' , [ Vrchol , E[i] ] ));
end;
```

//c) vypíše, po zadaní dvoch vrcholov (Odkial a Kam), či existuje hrana vedúca z prvého do //druhého vrcholu
{
Vrcholy, do ktorých vedú hrany z vrcholu Odkial sú v poli E na indexoch od V[Odkial] až po V[Odkial+1]-1; preto, ak sa v poli E vyskytuje v rozsahu E[ V[Odkial] ] až po E[ V[Odkial+1]-1 ] hodnota Kam, spojnica existuje, inak nie
}

```
procedure TForm1.btExistujeOdKamClick(Sender: TObject);
var Odkial, Kam, i: integer;
    Exist: boolean;
begin
Odkial:= StrToInt(InputBox( 'Zoznam nasledovníkov' , 'Odkial' , '' ));
Kam:= StrToInt(InputBox( 'Zoznam nasledovníkov' , 'Kam' , '' ));
Exist:= False;
for i:= V[Odkial] to V[Odkial+1]-1 do
        if E[i] = Kam then Exist:= True;
if Exist
then Memo1.Lines.Add(Format( 'Z vrcholu %d do vrcholu %d vedie hrana' , [ Odkial , Kam ] ))
else Memo1.Lines.Add(Format( 'Z vrcholu %d do vrcholu %d nevedie hrana' , [ Odkial , Kam ] ))
end;
```

Príklad by sa dal riešiť aj efektívnejšie, aby prehľadávanie poľa E skončilo hneď, keď sa v ňom nájde hodnota Kam. Násilne možno akýkoľvek cyklus ukončiť použitím príkazu break, teda doplniť `if E[i] = Kam then begin Exist:= True; Break; end;` alebo použiť cyklus s podmienkou ukončenia.

//d) vypíše, či existuje vrchol, do ktorého nevedie ani jedna hrana
{
Napríklad do vrcholu 1 nevedie ani jedna hrana, t.j. číslo vrcholu sa ani raz nevyskytuje v poli E, preto stačí zistiť, či sa vrchol 1 vyskytuje v poli E, potom vrchol 2 atď.
}

```
procedure TForm1.btExVrcholNevedieDoClick(Sender: TObject);
var Vrchol, i: integer;
    Exist: boolean;
begin
Vrchol:= 0;
repeat
        inc(Vrchol);
```

```
        Exist:= True;
        for i:= 1 to M do if E[i] = Vrchol then Exist:= False;
until Exist or (Vrchol = N);
if Exist
then Memo1.Lines.Add( 'Existuje vrchol, do kt. nevedie ani jedna hrana' )
else Memo1.Lines.Add( 'Vrchol, do kt. by neviedla hrana neexistuje' );
end;
```

Aj v ostatnej procedúre môžeme „neefektívny“ cyklus for zastaviť príkazom break alebo použiť cyklus s podmienkou ukončenia, ako sme to spravili v ďalšej verzii riešenia úlohy d)

```
procedure TForm1.btVrcholNevedieDoClick(Sender: TObject);
var Vrchol, i: integer;
begin
Vrchol:= 0;
repeat
        inc(Vrchol);
        i:= 0;
        repeat
                i:= i + 1;
        until (E[i] = Vrchol) or (i = M) ;
until (E[i]<>Vrchol) or (Vrchol = N);
if E[i] <> Vrchol
then Memo1.Lines.Add( 'Existuje vrchol, do kt. nevedie ani jedna hrana' )
else Memo1.Lines.Add( 'Vrchol, do kt. by neviedla hrana neexistuje' );
end;
```

//e) vypíše, či existuje vrchol, z ktorého nevychádza ani jedna hrana

{

Z vrcholu 4 nevychádza ani jedna hrana, t.j. počet vrcholov, do ktorých sa dá dostať zo zadaného vrcholu je nula, čo sa v poli V prejaví tak, že sú vedľa seba dve rovnaké hodnoty (žiaden index v poli E pre daný vrchol).

}

```
procedure TForm1.btExVrcholNevedieZClick(Sender: TObject);
var i: integer;
begin
i:= 0;
repeat
        i:= i+1
until (V[i] = V[i+1]) or (i = N);
if V[i] = V[i+1]
then Memo1.Lines.Add( 'Existuje vrchol, z kt. nevedie ani jedna hrana' )
else Memo1.Lines.Add( 'Vrchol, z kt. by neviedla hrana neexistuje' );
end;
```

# Údajový typ záznam

Najmä pri hromadnom spracovaní údajov, pri informačných systémoch, často treba spojiť do jedného celku rôzne údajové typy (integer, string, boolean a pod.). Umožňuje to údajový typ záznam.

**Záznam** je nehomogénny štruktúrovaný údajový typ, ktorý sa skladá z pevného počtu položiek, vo všeobecnosti rôznych typov.

Definícia typu záznam má tvar:

```
type mt = record
      p1 : tp1;        kde mt je meno typu
      p2 : tp2;        p1 až pn sú identifikátory položiek záznamu a
      ...              tp1 až tpn sú typy jednotlivých položiek
      pn : tpn
    end;
```

Napríklad:

```
type tKarta = record
        OsCislo : integer;
        Meno,
        Priezv : string[25];
        DatNar : record
                Den : 1..31;
                Mes : 1..12;
                Rok : 1900..2100
              end;
        Adresa : record
                Obec,
                Ulica : string[25];
                PSC : string[5]
              end;
        Priemer : array [1..5] of real;
      end;
type tBod = record
      x,
      y : real;
      farba : 0..15;
      blik : boolean
    end;
```

Ak p je premenná typu záznam, k jej položke s názvom poi sa dostaneme zápisom: p**.**poi. Napríklad, ak K je premenná typu tKarta, možno použiť zápisy: K.OsCislo, K.Meno, K.DatNar.Den, K.DatNar.Rok, K.Adresa.Obec, K.Priemer[1] a pod.

### Príklad 21.1

Vytvorte program, ktorý po zadaní súradníc bodu a súradníc stredu a polomeru kružnice v rovine zistí ich vzájomnú polohu.

*Riešenie*

Rozlíšime tri situácie, bod môže ležať na kružnici (jeho vzdialenosť od stredu kružnice sa zrejme rovná polomeru), bod leží „zvonka“ kružnice (jeho vzdialenosť od stredu kružnice je väčšia ako polomer) alebo bod leží „vo vnútornej oblasti“ kružnice (jeho vzdialenosť od stredu kružnice je menšia ako polomer).

Pri definícii typov tBod a tKruznica využijeme údajový typ záznam. Program je pomerne jednoduchý, zaujímavý použitím recordu. Pre súradnice aj polomer sme použili typ integer, ak by sme chceli bod a kružnicu aj nakresliť. Úlohu môžete riešiť aj pre reálne súradnice a polomer.

```
type tBod = record
              x,
              y: integer;
            end;
    tKruznica = record
                   S: tBod;                    //stred kružnice je bod S
                   r: integer;                 //polomer
                end;
var B: tBod;
   k: tKruznica;

procedure ZadajBod(var A: tBod);
begin
A.x:= StrToInt( InputBox( 'Bod' , 'X-ová súradnica' , '' ) );
A.y:= StrToInt( InputBox( 'Bod' , 'Y-ová súradnica' , '' ) );
end;

procedure ZadajKruznicu(var k: tKruznica);
begin
ZadajBod(k.S);                                 //bod k.S má dve súradnice k.S.x a k.S.y
k.r:= StrToInt( InputBox( 'Kružnica' , 'Polomer' , '' ) );
end;

procedure TForm1.btBodClick(Sender: TObject);
begin
ZadajBod(B);
end;

procedure TForm1.btKruznicaClick(Sender: TObject);
begin
ZadajKruznicu(k);
end;

function VzdDvochBodov(A, B: tBod): real;
begin
Result:= sqrt( sqr(A.x - B.x) + sqr(A.y - B.y) );
end;

procedure TForm1.btZistiClick(Sender: TObject);
var Vzd: real;
begin
Vzd:= VzdDvochBodov(B,k.S);
if Vzd > k.r then ShowMessage( 'Bod leží "zvonka" kružnice' );
if Vzd = k.r then ShowMessage( 'Bod leží na kružnici' );
if Vzd < k.r then ShowMessage( 'Bod leží "vo vnútornej oblasti" kružnice' );
end;
end.
```

Použitie tlačidiel Bod a Kružnica má výhodu, že môžeme meniť súradnice len jedného objektu, nevýhodou je potreba dodržať, po spustení programu, postupnosť zadanie vstupov a až potom výpočet vzájomnej polohy.

## Príklad 21.2

Vytvorte program, ktorý po zadaní súradníc bodov v rovine vypočíta obvod obdĺžnika s minimálnymi stranami rovnobežnými s osami súradnicovej sústavy, obsahujúceho všetky body.

*Riešenie*

Na obrázku je možné riešenie formulára. Body vygeneruje počítač, kvôli jednoduchosti, sám. K dispozícii je aj tlačidlo Zisti krajné body, ktoré, pre kontrolu, vypíše najmenšiu a najväčšiu x-ovú a y-ovú súradnicu (sú potrebné pre výpočet strán obdĺžnika, ktorý obsahuje všetky zadané body a je rozmerovo minimálny).

V riešení sme zvolili generovanie súradníc z intervalu <-10,10>, celé čísla, spracovali sme ako real.

```
const MAX = 10;
type   tBod = record
                  x,
                  y: real;
               end;
var    Bod: array of tBod;

procedure TForm1.btVygenerujBodyClick(Sender: TObject);
var Pocet, i: integer;
begin
Pocet:= StrToInt(InputBox('Body','Počet bodov','10'));
SetLength(Bod, Pocet);
for i:= 0 to High(Bod) do
begin
        Bod[i].x:= MAX - random(2*MAX+1);
        Bod[i].y:= MAX - random(2*MAX+1);
end;
end;

procedure TForm1.btVypisBodyClick(Sender: TObject);
var i: integer;
begin
for i:= 0 to High(Bod) do Memo1.Lines.Add(Format( '[%6f, %6f]' , [ Bod[i].x , Bod[i].y ] ));
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
randomize;
Memo1.Font.Name:= 'Courier';                //aby bol výpis krajší
end;
```

```
procedure TForm1.btOhradaClick(Sender: TObject);
var MinX, MinY, MaxX,MaxY: real;
    i: integer;
begin
MinX:= MAX; MinY:= MAX;                    //nastavenie počiatočných hodnôt
MaxX:= -MAX; MaxY:= -MAX;
for i:= 1 to High(Bod) do                  //hľadanie miním a maxím
begin
        if Bod[i].x < MinX then MinX:= Bod[i].x;
        if Bod[i].y < MinY then MinY:= Bod[i].y;
        if Bod[i].x > MaxX then MaxX:= Bod[i].x;
        if Bod[i].y > MaxY then MaxY:= Bod[i].y;
end;
Memo1.Lines.Add(Format( 'Ohrada má obvod %4f' , [ 2*(MaxX-MinX + MaxY-MinY) ] ));
end;
```

Procedúra Krajné body sa líši od procedúry Ohrada len o príkazy výpisu

```
Memo1.Lines.Add(Format( 'Xmin = %6f  Ymin = %6f' , [ MinX , MinY ] ));
Memo1.Lines.Add(Format( 'Xmax = %6f  Ymax = %6f' , [ MaxX , MaxY] ));
```

**Príkaz with**

Ak v časti programu používame opakovane tú istú položku alebo používame viacej položiek tej istej premennej typu záznam, príkaz with umožňuje zjednodušiť resp. skrátiť zápis k prístupu k týmto položkám.

Príkaz with má tvar: `with z do p` kde z je premenná typu záznam a p je príkaz

V príkaze p je dovolené označovať položky premennej z len identifikátormi, t.j. zápis z.polozka skrátiť na zápis polozka.

Napríklad:

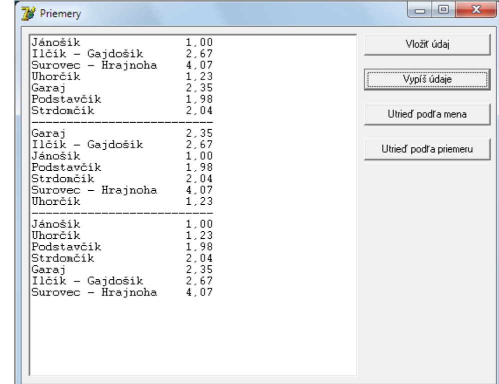
```
with Bod do begin
   X := StrToFloat( InputBox('Vstup' , 'X-ová súradnica bodu: ' , '') );
   Y := StrToFloat( InputBox('Vstup' , 'Y-ová súradnica bodu: ' , '') );
end;
with Z , DatNar do begin OsCislo:=99; Den:= 1; Mes:= 1; Rok:= 2012 end;
```

Pri zápise with z1,z2 do p možno v príkaze p skrátene označovať jednak položky záznamu z2, ale aj tie položky záznamu z1, ktoré sa nezhodujú v označení so žiadnou položkou záznamu z2.

## Príklad 21.3

Vytvorte program, ktorý umožní zadať meno a priemer žiaka, k jeho zápisu do poľa použije typ záznam a umožní utriediť údaje (záznamy) podľa mien alebo podľa priemerov.

*Riešenie*




```
type    tZiak = record
                Meno: string;
                Priemer: real;
            end;
        tTrieda = array of tZiak;
var     T: tTrieda;

procedure TForm1.btVlozitUdajClick(Sender: TObject);
begin
SetLength(T,length(T)+1);
with T[High(T)] do
```

```
begin
        Meno:= InputBox( 'Priemery' , 'Meno žiaka' , '' );
        Priemer:= StrToFloat(InputBox( 'Priemery' , 'Priemer žiaka' , '' ));
end;
end;

procedure TForm1.btVypisUdajeClick(Sender: TObject);
var i: integer;
begin
for i:=0 to High(T) do with T[i] do Memo1.Lines.Add(Format( '%-20s %5f' , [ Meno, Priemer ] ));
end;

procedure Vymen(var Z1,Z2: tZiak);
var Pom: tZiak;
begin
Pom:= Z1; Z1:= Z2; Z2:= Pom
end;

procedure TForm1.btUtriedMenaClick(Sender: TObject);
var PP, i: integer;
begin
for PP:= 1 to length(T)-1 do
        for i:= 0 to High(T)-PP do
                if T[i].Meno > T[i+1].Meno
                then Vymen(T[i] , T[i+1])
end;

procedure TForm1.btUtriedPriemeruClick(Sender: TObject);
var PP, i: integer;
begin
for PP:= 1 to length(T)-1 do
        for i:= 0 to High(T)-PP do
                if T[i].Priemer > T[i+1].Priemer
                then Vymen(T[i] , T[i+1])
end;
```

Vodorovné čiary - pomlčky vo výpise sme vložili z klávesnice, nie sú v procedúre Vypíš údaje naprogramované.

Vráťme sa ešte ku kapitole Grafy. Ďalšia reprezentácia grafov využíva údajový typ record a nazýva sa zoznam hrán.

**Zoznam hrán**
je pole alebo dynamický zoznam obsahujúci dvojice čísel vrcholov, medzi ktorými vedie hrana a prípadné ohodnotenie tejto hrany.
Príklad deklarácie

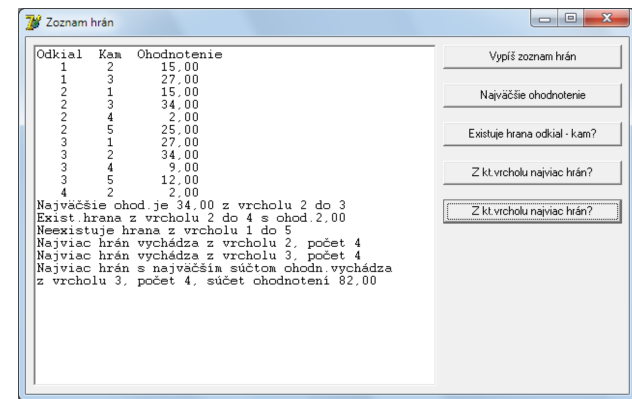
```
var Hrana: array [ 1 .. PocetHran ] of record
                                     Odkial, Kam: 1 .. PocetVrcholov;
                                     Ohodnotenie: real;
                                 end;
```

Zoznam hrán možno použiť pre grafy orientované aj neorientované, ohodnotené aj neohodnotené.

## Príklad 21.4

Vytvorte program, ktorý

a) vypíše zoznam hrán, t.j. odkiaľ, kam, ohodnotenie do troch stĺpcov (obrázok vpravo),
b) vypíše, odkiaľ – kam vedie hrana s najväčším ohodnotením,
c) vypíše, po zadaní dvoch vrcholov, či ich spája hrana,
d) vypíše, z ktorého vrcholu vychádza najviac hrán (ak ich je viacej, vypíše všetky),
e) *vypíše, z ktorého vrcholu ide najviac hrán, ak ich je viacej, vypíše ten, pre ktorý je súčet ohodnotení jeho hrán najväčší.




*Analýza*

Deklarácia zoznamu hrán má tvar

```
const   N = 6;
        M = 11;
var     Hrana: array [ 1 .. M ] of record
                                       Odkial, Kam: 1..N;
                                       Ohodnotenie: real;
                                  end;
```

Nech textový súbor zoznamhran.txt obsahuje po riadkoch tri údaje: odkiaľ – celé číslo, kam – celé číslo a ohodnotenie – reálne číslo. Nasledujúca procedúra uloží tieto údaje do poľa Hrana.

```
procedure TForm1.FormActivate(Sender: TObject);
var F: TextFile;
    i: integer;
begin
assignfile(F,'zoznamhran.txt');
reset(F);
for i:= 1 to M do with Hrana[i] do readln(F, Odkial, Kam, Ohodnotenie);
closefile(F);
Memo1.Font.Name:= 'Courier';                //pre krajší výpis, všetky znaky budú mať rovnakú šírku
end;
```

//a) vypíše zoznam hrán, t.j. odkiaľ, kam, ohodnotenie do troch stĺpcov

```
procedure TForm1.btVypisHranyClick(Sender: TObject);
var i: integer;
begin
Memo1.Lines.Add('Odkial  Kam  Ohodnotenie');                //hlavička budúcej tabuľky
for i:= 1 to M do
        with Hrana[i] do Memo1.Lines.Add(Format( '%4d %5d %10f' , [ Odkial , Kam , Ohodnotenie ] ));
end;
```

//b) vypíše, odkiaľ – kam vedie hrana s najväčším ohodnotením

{
Treba prejsť celé pole Hrana, t.j. pozrieť M hrán a nájsť maximum z hodnôt Hrana[i].Ohodnotenie.
}

```
procedure TForm1.btNajOhodnotenieClick(Sender: TObject);
var i, ZVrch, DoVrch: integer;
    Max: real;
begin
Max:= Hrana[1].Ohodnotenie; ZVrch:= Hrana[1].Odkial; DoVrch:= Hrana[1].Kam;
for i:= 2 to M do
        with Hrana[i] do if Ohodnotenie > Max
                          then begin
                                ZVrch:= Odkial;
                                DoVrch:= Kam;
                                Max:= Ohodnotenie;
                              end;
Memo1.Lines.Add(Format( 'Najväčšie ohod.je %f z vrcholu %d do %d' , [ Max , ZVrch , DoVrch ] ));
end;
```

//c) vypíše, po zadaní dvoch vrcholov, či ich spája hrana

{

Treba prechádzať poľom Hrana až kým Hrana[i].Odkial sa nebude rovnať prvej zadanej hodnote a zároveň hodnota Hrana[i].Kam sa nebude rovnať druhej zadanej hodnote alebo nebude koniec poľa Hrana.

}

```
procedure TForm1.btExistHranaClick(Sender: TObject);
var ZVrch, DoVrch, i: integer;
begin
ZVrch:= StrToInt(InputBox( 'Zoznam hrán' , 'Odkial' , '' ));
DoVrch:= StrToInt(InputBox( 'Zoznam hrán' , 'Kam' , '' ));
i:= 0;
repeat
        i:= i + 1;
until (Hrana[i].Odkial = ZVrch) and (Hrana[i].Kam = DoVrch) or (i = M);
if (Hrana[i].Odkial = ZVrch) and (Hrana[i].Kam = DoVrch)
then Memo1.Lines.Add(Format( 'Exist.hrana z vrcholu %d do %d s ohod.%f' , [ ZVrch , DoVrch ,
Hrana[i].Ohodnotenie ] ))
else Memo1.Lines.Add(Format( 'Neexistuje hrana z vrcholu %d do %d' , [ ZVrch , DoVrch ] ));
end;
```

//d) vypíše, z ktorého vrcholu vychádza najviac hrán (ak ich je viacej, vypíše všetky)

{

Do poľa Pocet budeme ukladať počty hrán vychádzajúce z prvého, druhého,..., N-tého vrcholu. Indexom poľa Pocet je vrchol, z ktorého hrana vychádza. Následne treba nájsť maximum v poli Pocet a príslušný index je hľadaným vrcholom. Na zistenie, či je viacej rovnakých maxím v poli Pocet, treba opäť prejsť pole Pocet (stačí od prvého indexu prislúchajúceho maximu +1).

}

```
procedure TForm1.btNajviacHran1Click(Sender: TObject);
var i, MaxPocet, MaxVrchol: integer;
    Pocet: array [1..N] of integer;
begin
for i:= 1 to N do Pocet[i]:= 0;
for i:= 1 to M do inc(Pocet[Hrana[i].Odkial]);
MaxPocet:= 0;
```

```
for i:= 1 to N do if Pocet[i] > MaxPocet
                    then begin
                           MaxPocet:= Pocet[i];
                           MaxVrchol:= i;
                         end;
Memo1.Lines.Add(Format( 'Najviac hrán vychádza z vrcholu %d, počet %d' , [MaxVrchol , MaxPocet]));
for i:= MaxVrchol+1 to N do
         if Pocet[i] = MaxPocet
         then Memo1.Lines.Add(Format('Najviac hrán vychádza z vrcholu %d, počet %d',[i,MaxPocet]));
end;
```

//e) *vypíše, z ktorého vrcholu ide najviac hrán, ak ich je viacej, vypíše ten, pre ktorý je súčet
// ohodnotení jeho hrán najväčší (úloha je „hviezdičková“, určená len pre fanatikov)

{
Ako úloha d), len je rozšírená o výber toho vrcholu, zo všetkých maxím, pre ktorý je súčet ohodnotení najväčší (súčet ohodnotení pre i-ty vrchol sa ukladá do Sucet[i]).
}

```
procedure TForm1.btNajviacHran2Click(Sender: TObject);
var i, MaxPocet, MaxVrchol: integer;
    Pocet: array [1..N] of integer;
    Sucet: array [1..N] of real;
    MaxSucet: real;
begin
for i:= 1 to N do begin Pocet[i]:= 0; Sucet[i]:= 0; end;
for i:= 1 to M do
begin
         inc(Pocet[Hrana[i].Odkial]);
         Sucet[Hrana[i].Odkial]:= Sucet[Hrana[i].Odkial] + Hrana[i].Ohodnotenie;
end;
MaxPocet:= 0; MaxSucet:= 0;
for i:= 1 to N do if Pocet[i] >= MaxPocet
                    then if Pocet[i] > MaxPocet
                          then begin
                                 MaxPocet:= Pocet[i];
                                 MaxVrchol:= i;
                                 MaxSucet:= Sucet[i];
                               end
                          else if Sucet[i] > MaxSucet
                                then begin
                                       MaxVrchol:= i;
                                       MaxSucet:= Sucet[i];
                                     end;
Memo1.Lines.Add( 'Najviac hrán s najväčším súčtom ohodn.vychádza' );
Memo1.Lines.Add(Format( 'z vrcholu %d, počet %d, súčet ohodnotení %f' , [ MaxVrchol , MaxPocet ,
MAxSucet ] ));
end;
```

# Správnosť algoritmu

**Algoritmus je vzhľadom na vstupné a výstupné podmienky správny**, ak:

1. pre všetky vstupné údaje spĺňajúce vstupnú podmienku sa proces predpísaný algoritmom zastaví a
2. výstupné údaje spĺňajú výstupnú podmienku.

Algoritmus je **čiastočne správny**, ak pre vstupné údaje, ak skončí, dáva správne výsledky.
Poznámka: Čiastočne správny je aj algoritmus, ktorého výpočet neskončí pre žiadne vstupné údaje.

Algoritmus je **správny**, ak je

1. čiastočne správny a
2. konečný, t.j. jeho výpočet pre všetky vstupné údaje skončí.

Poznámky:
Dokázať konečnosť algoritmu znamená dokázať najmä konečnosť v ňom použitých cyklov. Dokázať konečnosť cyklu znamená dokázať, že hodnoty premennej riadiacej počet prechodov cyklom tvoria klesajúcu, zdola ohraničenú postupnosť alebo rastúcu, zhora ohraničenú postupnosť.
Často je jednoduchšie dokázať, že algoritmus nie je správny. K dôkazu „nesprávnosti" algoritmu stačí nájsť aspoň jeden konkrétny príklad vstupných údajov, pre ktoré algoritmus zlyhá.

Príklad 1:
Dôkaz správnosti algoritmu na zistenie, či zadané prirodzené číslo ≥ 2 je prvočíslo.
(Prirodzené číslo väčšie ako 1 je prvočíslo, ak je deliteľné len 1 a sebou samým.)

Algoritmus:
```
ak Cislo ≥ 2 tak začiatok
        Delitel := 2;
        pokiaľ Cislo mod Delitel ≠ 0 opakuj Delitel := Delitel + 1;
        koniec;
ak Delitel = Cislo tak píš ('Číslo je prvočíslo.')
inak píš ('Číslo nie je prvočíslo.')
```

Algoritmus je čiastočne správny, pretože za prvočísla označí len tie z prirodzených čísel väčších ako 1, ktorých jediným deliteľom (okrem jednotky) je Delitel = Cislo (Cislo mod Cislo sa vždy rovná 0 pre vstupné hodnoty).
Algoritmus je konečný, pretože je konečný cyklus „pokiaľ", ktorý obsahuje premennú Deliteľ, ktorej hodnoty tvoria rastúcu postupnosť (Deliteľ = 2, 3, 4,...) zhora ohraničenú číslom Cislo (každé číslo je deliteľné bezo zvyšku sebou samým).
Algoritmus je správny, pretože sme dokázali, že je čiastočne správny a konečný.

Príklad 2:
Dôkaz správnosti Euklidovho algoritmu na nájdenie najväčšieho spoločného deliteľa dvoch celých čísel odčítaním:

1. algoritmus je čiastočne správny, t.j. každý spoločný deliteľ čísel pôvodných je spoločným deliteľom aj novej dvojice čísel a výsledok je najväčším deliteľom daných čísel:
   nech napr. A>B a A>0 a B>0
   ak D delí A aj B => ∃ k1, k2∈Z: A = k1*D a B = k2*D => A – B = k1*D – k2*D = (k1 – k2)*D => D delí aj A- B

   $$NSD(A,B) = \begin{cases} NSD(A-B,B) = \ldots \\ \text{alebo} \\ NSD(A,B-A) = \ldots \end{cases} \quad \ldots = NSD(D,D)$$

   Výpočet končí dvojicou čísel D, D, ktorej najväčším spoločným deliteľom je zrejme číslo D.

2. konečnosť algoritmu, t.j. že dvojice čísel A a B tvoria klesajúcu zdola ohraničenú postupnosť použijeme súčet čísel A a B, pre ktoré zrejme platí:

$$A + B > \begin{cases} A - B + B > \ldots \\ \text{alebo} \\ A + B - A > \ldots \end{cases} \ldots \geq 2$$

V dôkaze čiastočnej správnosti Euklidovho algoritmu sme použili tzv. invarianty.

**Invariant** je matematický vzťah, ktorý platí pri každom prechode daným bodom algoritmu.

Napr. do algoritmu výpočtu n-faktoriálu môžeme zapísať tieto invarianty:

začiatok Invarianty:
čítaj (N); ................................[ $N = n_0$
F := 1; ...................................[ $N = n_0$ a $F = 1$
I := 1; ....................................[ $N = n_0$ a $F = 1$ a $I = 1$
pokiaľ I <= N opakuj .............[ $N = n_0$ a $F = 1*1*2*...*(I-1)$ a $1 \leq I \leq n_0 + 1$
začiatok
  F := F * I; .........................[ $N = n_0$ a $F = 1*1*2*...*I$ a $1 \leq I \leq n_0$
  I := I + 1; ..........................[ $N = n_0$ a $F = 1*1*2*...*(I-1)$ a $1 \leq I \leq n_0 + 1$
koniec;
píš (F); ..................................[ $N = n_0$ a $F = 1*1*2*...*(I-1)$ a $I = n_0 + 1$ => **F = N!**
koniec

V príklade sme pomocou invariantov dokázali čiastočnú správnosť uvedeného algoritmu pre výpočet n-faktoriálu. Počet prechodov cyklom riadi premenná I, ktorej hodnoty tvoria rastúcu zhora ohraničenú postupnosť a preto cyklus skončí po konečnom počte krokov a algoritmus je teda konečný.

Na overenie správnosti algoritmu pre konkrétne vstupné údaje slúži testovanie vytvoreného algoritmu pomocou testovacej tabuľky (nie je dôkazom správnosti algoritmu!). Testovacia tabuľka obsahuje v hlavičke všetky premenné použité v algoritme a simuluje priebeh vykonávania algoritmu. Pokiaľ je počet vstupných údajov konečný, dala by sa testovaním dokázať správnosť algoritmu (otestovaním všetkých vstupných hodnôt). Algoritmy však väčšinou majú nekonečne veľa vstupných hodnôt. Vtedy sa snažíme otestovať aspoň všetky cesty algoritmom a hraničné hodnoty (pre n-faktoriál sú to hodnoty n = 0 a n = 1). Pri testovaní používame také vstupné hodnoty, pre ktoré poznáme správne výstupné hodnoty. Testovaním algoritmu môžeme získať vstupné hodnoty, pre ktoré algoritmus nedá správne výstupné hodnoty, čím dokážeme, že nie je správny.

Testovacia tabuľka pre n-faktoriál (vieme, že 3! = 6 a 0! = 1):

N = 3

| krok | N | I | F | podmienka |
|---|---|---|---|---|
| 1. | 3 | | | |
| 2. | | | 1 | |
| 3. | | 1 | | |
| | | | | $1 \leq 3$ áno |
| 4. | | | 1 | |
| 5. | | 2 | | |
| | | | | $2 \leq 3$ áno |
| 4. | | | 2 | |
| 5. | | 3 | | |
| | | | | $3 \leq 3$ áno |
| 4. | | | 6 | |
| 5. | | 4 | | |
| | | | | $4 \leq 3$ nie |
| 6. | | | (6) | |

N = 0 (hraničná hodnota)

| krok | N | I | F | podmienka |
|---|---|---|---|---|
| 1. | 0 | | | |
| 2. | | | 1 | |
| 3. | | 1 | | |
| | | | | $1 \leq 0$ nie |
| 6. | | | (1) | |

# Výpočtová zložitosť algoritmu a programu

**Rozsah problému** je prirodzené číslo N, ktoré vyjadruje veľkosť vstupných údajov.
Napr. počet čísel v poli, počet cifier v čísle a pod.

**Časová výpočtová zložitosť** $\mathcal{T}$(N) je funkcia, ktorá popisuje závislosť medzi rozsahom problému a potrebným počtom krokov na jeho vyriešenie.
Kroky napr.: počet operácií abstraktného procesora, počet „významných" operácií (aritmetické, logické, porovnania, presuny,...).

**Pamäťová výpočtová zložitosť** $\mathcal{S}$(N) je funkcia, ktorá popisuje závislosť medzi rozsahom problému a veľkosťou pamäte potrebnej na jeho vyriešenie.
Napr. počet bitov potrebných na uloženie všetkých údajov.

**Optimálny algoritmus** je taký, ktorý
- rieši daný problém a
- lepšie (efektívnejšie) sa už vyriešiť nedá.

**Najhorší prípad pri danom N** je konkrétna skupina údajov, ktorá vedie podľa voľby vstupných údajov k najhoršiemu prípadu (výpočet trvá najdlhšie).
Napr. pri bubblesorte je to prípad, keď sú vstupné údaje utriedené zostupne.

**Priemerný prípad** (štatistický pojem) je očakávaná zložitosť pri náhodne zvolenej skupine vstupných údajov rozsahu N.

**Asymptotická časová zložitosť** $\mathcal{O}(f)$, zjednodušene povedané, vyjadruje, ako rýchlo rastie časová výpočtová zložitosť s rastúcou hodnotou N (teoreticky s $N \to \infty$). $\mathcal{O}(f)$ je asymptotickým vyjadrením funkcie $\mathcal{T}$(N). V asymptotickej časovej zložitosti vynechávame multiplikačné konštanty a členy, ktoré sú menšie ako člen s najväčšou hodnotou.

Napr. pre bubblesort s cyklami for PP := 1 to N-1 do for I := 1 to N-PP do...
$\mathcal{T}(N) = (n - 1) * \frac{n}{2} = \frac{1}{2} n^2 - \frac{1}{2} n$, čo zapisujeme $\mathcal{O}(n^2)$.

Asymptotická časová výpočtová zložitosť môže byť:
- **polynomiálna** $\mathcal{O}(n^{exp})$, napr.:
  - **konštantná** $\mathcal{O}(1)$, napr. výpočet podľa vzorca, nájsť maximum v utriedenom poli
  - **lineárna** $\mathcal{O}(n)$, napr. hľadanie prvku s požadovanou vlastnosťou v neutriedenom poli (potrebných rádovo n porovnaní, kde n je počet prvkov poľa)
  - **kvadratická** $\mathcal{O}(n^2)$, napr. triedenia bubblesort, insertsort, selectsort,...
  - **kubická** $\mathcal{O}(n^3)$, napr. súčin dvoch matíc (tri vnorené cykly)
- **logaritmická**
  - $\mathcal{O}(\log_2 n)$, napr. hľadanie prvku s požadovanou vlastnosťou v utriedenom poli (potrebných rádovo $\log_2 n$ porovnaní, kde n je počet prvkov poľa)

    Počet delení (porovnaní): n, $\frac{n}{2}$, $\frac{n}{4}$,..., v najhoršom prípade $\frac{n}{2^p} = 1 \Rightarrow p = \log_2 n$
  - $\mathcal{O}(\log n)$, napr. pri operáciách s ciframi v desiatkovom čísle
- **exponenciálna** $\mathcal{O}(a^n)$ – napr. rekurzívny výpočet Fibonacciho čísla, prehľadávanie stromov do hĺbky - prakticky nerealizovateľná na počítači pre veľké n

- $\mathcal{O}$ (n!), napr. zo všetkých možností usporiadania n prvkov vybrať tie, pri ktorých sú prvky usporiadané vzostupne
- $\mathcal{O}$ ($n * \log_2 n$), napr. triediaci algoritmus Quicksort
- $\mathcal{O}$ ($\sqrt{n}$ ),…

Ukážka určenie $\mathcal{T}$ (N) napr. pri rekurzii – výpočte n-faktoriálu: rovnici T (n) ≈ T(n-1) + 1 (jedno násobenie) vyhovuje T (n) = n lebo n ≈ n + 1, čo je lineárna časová výpočtová zložitosť.

## Efektívnosť algoritmu a programu

Z dvoch algoritmov riešiacich tú istú algoritmickú úlohu je **časovo** efektívnejší ten, ktorého realizácia vyžaduje menej krokov (trvá kratšie). Efektívnejší je ten, ktorý je rýchlejší pri väčšom rozsahu problému (pri väčšom N). **Pamäťovo** efektívnejší je ten, ktorý potrebuje menej pamäťového miesta (menej premenných).
Úpravu algoritmu na efektívnejší tvar nazývame **optimalizáciou** algoritmu.

**Zásady optimalizácie:**

- výber najefektívnejšej metódy riešenia pre najpravdepodobnejší rozsah problému
- nerobiť v cykle opakovane to, čo sa dá spraviť jedenkrát pred ním
- optimalizovať predovšetkým telá cyklov
- využívať predchádzajúce výsledky (volania procedúr)
- zjednodušovať výrazy vybraním pred zátvorku

Napríklad

- výpočet hodnoty polynómu n-tého stupňa (sledujeme počet násobení):
  - umocňovaním $a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$: $\mathcal{T}$ (n) = n + (n-1) + ... + 1 – kvadratická zložitosť
  - Hornerovou schémou $(...((a_n x + a_{n-1}) * x + a_{n-2}) * x + ...) * x + a_0$: $\mathcal{T}$ (n) = n – lineárna zložitosť
- výpočet NSD (A, B)
  - odčítaním napr. NSD (5 000, 5) =NSD(4 995, 5)=NSD(4 990, 5)=...spolu 999 odčítaní ...= 5
  - funkciou mod napr. NSD (5 000, 5) = NSD (0, 5) = 5 (jedno delenie)
- použitie statického poľa je časovo efektívnejšie ako dynamického
- použitie parametrov nahradzovaných odkazom je pre štruktúrované údajové typy pamäťovo efektívnejšie, ako použitie parametrov nahradzovaných hodnotou

OBSAH

Algoritmus, algoritmické konštrukcie, riadiace príkazy ....3
Algoritmus....3
Testovanie a ladenie programu ....4
Sekvencia ....7
Vetvenie ....7
Úplné binárne vetvenie, podmienený príkaz if ....7
Neúplné binárne vetvenie, neúplný príkaz if ....8
N-árne vetvenie, podmienený príkaz case....8
Cyklus....13
Cyklus s pevným počtom opakovaní, príkaz for....13
Cyklus s podmienkou ukončenia na začiatku, príkaz while....14
Cyklus s podmienkou ukončenia na konci, príkaz repeat ....14
Kedy ktorý cyklus ....15
Údajové typy ....17
Štandardné údajové typy ....17
Štruktúrované údajové typy ....19
Typ interval ....19
Vymenovaný typ ....19
Údajový typ množina ....20
Procedúry, funkcie a unity ....21
Procedúry bez lokálnych objektov a bez parametrov....21
Procedúry s lokálnymi objektmi a bez parametrov ....23
Procedúry s parametrami....23
Funkcie ....28
Unity....38
Rekurzia ....39
Jednorozmerné pole....47
Dynamické pole....47
Funkcia Format ....50
Lineárne vyhľadávanie....55
Binárne vyhľadávanie ....59
Triedenie....63
Bublinkové triedenie (Bubble sort)....63
Triedenie priamym vkladaním (Insertion sort) ....64
Triedenie priamym výberom (Selection sort) ....66
Rýchle triedenie (Quick sort) ....71
Ďalšie úlohy na dynamické pole ....73
Statické pole ....77
Konštantné pole....80
Hodnoty premenných zadané pri ich deklarácii....83
Otvorené pole ako parameter procedúry ....83
Zásobník....85
Rad ....89
Textový súbor....97
Dvojrozmerné pole....121
Grafy....127
Údajový typ záznam....135
Správnosť algoritmu....143
Výpočtová zložitosť algoritmu a programu ....145

Použitá literatúra

Kalaš I. a kol.: Informatika pre stredné školy (učebnica), SPN
Blaho A.: Programovanie v Delphi (učebnica), SPN
Töpfer P.: Algoritmy a programovací techniky, Prometheus, 1995
Rychlík J.: Programovací techniky, KOOP, 1992
Drózd J., Kryl R.: Začínáme s programováním, GRADA, 1992
Wirth N.: Algoritmy a štruktúry údajov, ALFA, 1987