ZÁKLADY PROGRAMOVANIA

TURBO PASCAL

Jozef Piroško

Názov: Základy programovania - Turbo Pascal
Autor : RNDr. Jozef Piroško
Vydanie: prvé
Vydané: marec 2000

OBSAH

Úvod

Skriptá Základy programovania – Turbo Pascal sú sumarizáciou poznatkov z klasického kurzu programovania. Nekladú si za cieľ podrobne vyložiť všetky pojmy. Tiež výklad a poradie tém má klasické usporiadanie, od najjednoduchšieho k zložitejšiemu, od základných pojmov k zložitejším štruktúram. Výklad a príklady sa nevyhýbajú úlohám s matematickým základom, ako sa o to pokúšajú modernejšie učebnice programovania. Domnievam sa, že ak máte odmietavý vzťah k matematike, nemôže vás programovanie a algoritmizácia zaujať. Musíte byť trochu matematik, aby ste presne a jednoznačne vedeli formulovať a zapísať svoje myšlienky a trochu umelec, aby ste dokázali abstraktne myslieť a vytvoriť si svoje originálne riešenie daného problému. Základné „ťahy štetcom“ sa možno naučíte práve z týchto skrípt.

Obsahom výkladu sú základné algoritmické konštrukcie sekvencia, vetvenie a cyklus, a príkazy Turbo Pascalu, umožňujúce zrealizovať tieto konštrukcie v programe. Preto sa v príkladoch používajú len štandardné údajové typy integer, real, boolean a char a jediný štruktúrovaný typ string. Nekládol som si za cieľ o každom zavedenom pojme uviesť všetko, skôr ukázať jeho najtypickejšie použitie, preto sa miestami vedome dopúšťam nepresností alebo skôr neúplností, mali by však byť skutočne nepodstatné; možno sa dopúšťam aj chýb, ale tých určite nevedome. Tiež si nekladiem nároky na autorstvo použitých príkladov. Vybral som ich z veľkého množstva príkladov, s ktorými som sa stretol za viac ako desať rokov praxe. O originálnosti riešenia ťažko hovoriť, keďže ide o „triviálne“ príklady. Týmto spĺňam aj požiadavku študentky Maji S., ktorá ma raz poprosila o vydanie Triviálností pána procesora Pirošku (premenovanie je tiež dielom študenta, tentoraz Tomáša Z.).

Skriptá, okrem výkladu základných štruktúr a príkazov, obsahujú aj veľké množstvo riešených a neriešených úloh. Najprv by ste sa sami mali potrápiť nad úlohou, a až potom siahnuť po riešení v skriptách. Riešenia si treba ozrejmiť, tým sa naučíte minimálne „čítať“ programy a možno aj nové riešenie daného problému. Ideálne by bolo, keby ste si pri štúdiu programu kládli a najmä odpovedali na otázky charakteru: Počíta program správne aj pre takéto hodnoty?, Musí to byť zapísané práve takto?, Počiatočná hodnota premennej musí byť práve takáto?, Čo sa stane, keď to napíšem takto?, Čo keď zamením poradie týchto príkazov? A ak tento príkaz nahradím týmto? atď. Svoje tvrdenia si vždy overte priamo v TP odladením zmeneného programu.

Na záver vám prajem veľa originálnych riešení, najprv sa však treba naučiť „v pote tváre zopár základných ťahov štetcom“.

Algoritmus, vlastnosti algoritmu, formy zápisu algoritmu, algoritmické konštrukcie

Ak sa začnete zaujímať o počítače a najmä programovanie, určite narazíte na slovo algoritmus. Algoritmus je návod, postup, ako vyriešiť určitý problém. Tým problémom môže byť výmena pneumatiky pri defekte, zostavenie mlynčeka na mletie mäsa tak, aby to, čo vznikne zomletím, bolo ešte ďalej použiteľné, bezpečné prejdenie na druhú stranu cesty, „vzorné“ umytie zubov atď. Z týchto príkladov by malo byť zrejmé, že algoritmy nás učia od narodenia a na každom kroku („toto sprav takto a toto zas takto...“). Snáď v každej domácnosti sa nachádza zbierka algoritmov na prípravu jedál – kuchárska kniha. Každý návod na obsluhu, ktorý by ste mali dostať pri kúpe nového výrobku, je vlastne algoritmom. Vo všetkých týchto príkladoch je vykonávateľom algoritmu človek. Pri zápise algoritmu určeného pre človeka si môžeme dovoliť použiť spojenia „krátko povariť“, „pritiahnuť až na doraz“, „vyladiť stanicu“ a pod. Ak je však vykonávateľom algoritmu stroj, napr. počítač, sotva by porozumel uvedeným príkazom. Preto aj my prejdeme na odbornejšiu terminológiu (chceme sa predsa venovať počítačom a nie vareniu). **Algoritmus** je konečná postupnosť elementárnych príkazov, vykonanie ktorých, pre prípustné vstupné údaje, po konečnom počte krokov (akcií) vedie mechanicky k získaniu korektných výstupných údajov. Strašná veta, skúsme ju „rozpitvať“. Konečná postupnosť elementárnych príkazov znamená, že ide o príkazy, ktoré sú zapísané v presne stanovenom poradí, je ich konečný počet a pre vykonávateľa algoritmu sú elementárne, t.j. každý z príkazov pozná a vie ho vykonať. Vykonávanie algoritmu je proces, do ktorého vo všeobecnosti vstupujú na začiatku prípustné vstupné údaje a ich spracovaním získavame nové – výstupné údaje (na čo by sme ináč „trápili“ počítač). Celý proces samozrejme nesmie trvať nekonečne dlho (to by sme sa nedopracovali k žiadnym výsledkom) a, keďže počítač je síce usilovný ale hlupák[1], celý proces musí prebiehať mechanicky - bez rozmýšľania. Pod korektnými výstupnými údajmi rozumieme zmysluplné výsledky adekvátne vstupným údajom; ak napríklad opakovane použijeme rovnaké vstupné údaje, musíme dostať tie isté výsledky.

Z tejto analýzy vyplývajú aj **vlastnosti** algoritmu, z ktorých by sme uviedli len niekoľko:

1. hromadnosť – algoritmus slúži na riešenie celej skupiny úloh určitého typy[2]; aby sme dodržali túto vlastnosť, musí byť algoritmus napísaný všeobecne (univerzálne). Z matematiky vieme, že zovšeobecniť zápis nám umožňujú premenné. Už na základnej škole nás učili, že napríklad obsah obdĺžnika sa vypočíta S = a.b, kde a a b sú strany obdĺžnika a nie S = 5.7 pre nejaký konkrétny obdĺžnik. Preto prvý zápis môže byť príkazom algoritmu, druhý nie. Zaujímavé je, že algoritmus musí byť napísaný všeobecne ale jeho realizácia vždy prebieha s konkrétnymi údajmi; premenné nadobúdajú konkrétne hodnoty v závislosti od zadaných vstupných hodnôt. Ak budú vstupné údaje - strany obdĺžnika a = 5 a b = 7, tak konkrétny výpočet bude S = 5.7 a po ňom premenná S nadobudne hodnotu 35. Hromadnosť vlastne znamená, že algoritmus slúži na riešenie určitej skupiny konkrétnych úloh, ktoré sa líšia len vstupnými hodnotami.

---

[1] Jeden múdry pán raz povedal: „Počítač je usilovný hlupák na triedenie jednotiek a núl“. Dúfam, že s ním súhlasíte.

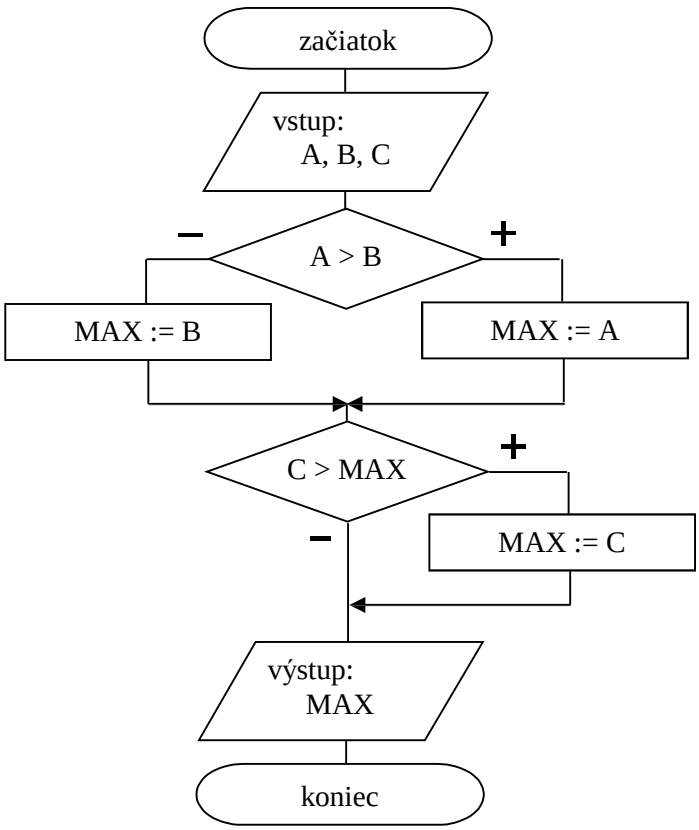
[2] My predsa nevieme vymeniť len tú konkrétnu pneumatiku na aute, na ktorom nás to učili, ale u daného typu auta kedykoľvek ktorúkoľvek pneumatiku; alebo prejsť cez cestu by sme mali vedieť bezpečne cez ktorúkoľvek cestu, podobnú tej, na ktorej nás to učili; zuby si tiež vieme umyť, či ich máme 32 alebo menej, atď.

2. deterministickosť (jednoznačnosť) – jednotlivé kroky (akcie) a ich poradie je presne a jednoznačne určené. Deterministickosť zaručuje, že pri opätovnom použití algoritmu pre tie isté vstupné údaje dostaneme rovnaký výsledok. Umožňuje tiež realizáciu algoritmu počítačom (počítač sa nemusí sám rozhodovať).

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; ide o staršiu formu zápisu algoritmu.
  Príklad: Algoritmus na nájdenie najväčšieho z troch čísel vo forme vývojového diagramu.



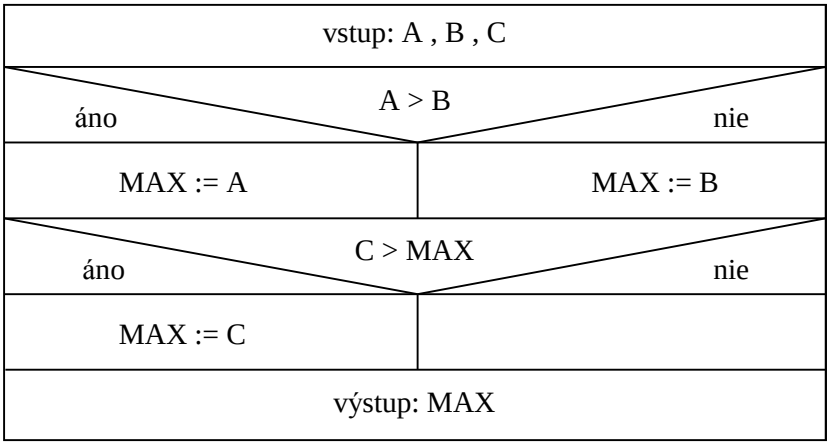
  Algoritmus sa vykonáva zhora nadol. Vetvou „+“ sa pôjde, keď bude podmienka splnená a vetvou „ -“, keď podmienka nebude splnená. Symbol „:=“ čítame „priraď“.

- slovný zápis používa tzv. kľúčové (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[1] – najnovšia grafická forma zápisu algoritmu.

  Príklad: Algoritmus na nájdenie najväčšieho z troch čísel zapísaný N-S diagramom.



Položme si otázku: Je každá úloha algoritmicky riešiteľná, t.j. existuje na každý problém algoritmus, pomocou ktorého ho vieme vyriešiť? Intuitívne cítime, že asi nie. Na algoritmicky neriešiteľné úlohy síce nenarazíme „na každom kroku“, ale predsa existujú. Podarilo sa napríklad dokázať tvrdenie: Neexistuje algoritmus, ktorý by pre akýkoľvek daný program rozhodol (na základe zdrojového tvaru programu a vstupných údajov), či tento program skončí.[2] Zatiaľ sme neobjasnili slovo *program*. Veľmi jednoducho: **program** je algoritmus úlohy zapísaný v jazyku, ktorému rozumie počítač, v tzv. programovacom jazyku. Onedlho sa jedným z programovacích jazykov začneme podrobnejšie zaoberať.

Ak je úloha algoritmicky riešiteľná, jej algoritmus možno vytvoriť kombináciou len troch algoritmických konštrukcií (Na prvý pohľad aké jednoduché!). **Algoritmické konštrukcie** sú: sekvencia, vetvenie a cyklus.

Teraz prerušíme výklad algoritmických konštrukcií a povieme si niečo o programovaní a programovacom jazyku Pascal.

Programovanie, programovací jazyk Turbo Pascal

Tvorbu algoritmov nazývame **algoritmizácia** a tvorbu programov **programovanie**. Keďže program je „len“ algoritmus zapísaný v programovacom jazyku, prioritnou činnosťou je algoritmizácia. Málokto však vydrží najprv „vyrobiť“ nedozerné množstvo algoritmov (naučiť sa algoritmizáciu) a až potom začať programovať. Je to pochopiteľné, pretože až počítač pomocou programu mu dokonale „zhmotní“ to, čo vymyslel na algoritmizácii. Preto aj my, aby sme si hneď overili, čo sme vymysleli, spojíme algoritmizáciu a programovanie. Pri preberaní algoritmických konštrukcií si hneď uvedieme aj príkazy programovacieho jazyka, ktoré nám umožňujú danú konštrukciu zrealizovať v programe. Budeme sa zaoberať základmi programovacieho jazyka Turbo Pascal, implementáciou jazyka Pascal, ktorý navrhol v roku 1971 profesor

[1] Tento názov sme zvolili pre podobné formy zápisu algoritmu, ako sú napríklad kopenogramy (autori Kofránek, Pecinovský a Novák) alebo N-S diagramy (autori Nassi a Schneiderman).

[2] Toto tvrdenie napríklad vedie k všeobecnejšiemu tvrdeniu: Neexistuje algoritmus, ktorý rozhodne, či v programe je alebo nie je chyba. Už vieme, prečo je „programátorský chlebík“ taký ťažký.

informatiky Niklaus Wirth[1]. Pascal bol vytvorený na vyučovanie programovania a má vlastnosti a kontrolné mechanizmy nútiace, alebo aspoň umožňujúce, tvoriť zrozumiteľné a prehľadné programy s čo najmenším počtom začiatočníckych chýb.

Program nie je len mechanickým prepisom algoritmu do programovacieho jazyka. Kým v algoritme sa snažíme vystihnúť podstatu riešenia problému a nezaoberáme sa podrobnosťami komunikácie s užívateľom, program sa snažíme spraviť užívateľsky „prítulný“ (budeme hovoriť o užívateľskom komforte). Na to slúžia ďalšie, rozširujúce príkazy. Tieto a ďalšie vlastnosti programovacieho jazyka Turbo Pascal si budeme postupne ozrejmovať pri konkrétnych programoch.

Pri práci v Turbo Pascale odporúčame:

- v Options – Environment – v položke Preferences... v skupine Auto save zapnúť voľbu Editor files a zapamätať ju položkou Save \...\ TURBO.TP; táto voľba spôsobí, že pred prekladom programu sa tento automaticky najprv uloží na disk (ak sme súbor s programom ešte nepomenovali, vyzve nás vložiť názov súboru), čo pri prípadnom použití tlačidla RESET neznamená stratu programu (najprv však skúste Ctrl+Break alebo v prostredí Windows 9x Ctrl+Alt+Del)
- v Options – Compiler... zapnúť v skupine Runtime errors všetky kontroly na chyby počas behu programu a zapamätať ich položkou Save \...\ TURBO.TP (jednotlivé kontroly preberieme podľa aktuálnosti)
- aj písanie nových programov začínať cez F3 Open a hneď vložiť názov súboru, do ktorého má byť nový program ukladaný
- čo najviac používať kontextový help: napísať kľúčové slovo, presunúť pod neho kurzor a stlačiť Ctrl+F1
- používať po označení textu pomocou myši alebo Shift a „kurzorových šípok“ na kopírovanie kombináciu Ctrl+C a K, premiestňovanie Ctrl+C a V a mazanie Ctrl+C a Y (alebo položky z ponuky Edit)
- programy spúšťať pomocou Ctrl+F9. Ak sme program práve napísali alebo v ňom spravili zmenu, dôjde najprv k prekladu programu, pred ktorým sa vykoná aj kontrola správnosti zápisu (odhalia sa skomolené vyhradené slová, chýbajúce bodkočiarky a pod.) a úplnosti programu (odhalia sa nedeklarované premenné, nedovolené priradenia a pod.). Táto kontrola neodhalí tzv. logické chyby (chyby v algoritme) alebo chyby, ktoré môžu nastať počas behu programu (zle zadaná vstupná hodnota, delenie nulou[2] a pod.)

Pri písaní programu si programátor volí názov programu, mená použitých premenných, konštánt, typov a iných objektov. Keďže norma Pascalu dovoľuje používať len písmená anglickej abecedy, vo zvolených názvoch nesmú byť použité diakritické znamienka (dĺžne, mäkčene a pod.). Tieto názvy sú tzv. identifikátory. **Identifikátor** je postupnosť písmen (anglickej abecedy) a číslic začínajúca písmenom; medzi písmená je dodefinovaný aj podčiarnik „_“. Identifikátory sú napríklad: CISLO, SUCET, obvod_obdlznika, ObsahStvorca, Premenna1, MENO, S, X, y, A, a, N5, z_13, ZNAK, text, OBSAHUJE, ... Identifikátory nie sú napríklad: 1A (nezačína písmenom), SUCET CISEL (použitá medzera), premenná1 (nedovolené písmeno á), suma$ (nedovolený znak $). Nerozlišujú sa malé a veľké písmená, zápisy sucet, Sucet, SuCeT,

[1] Narodil sa v roku 1934, prednášal na Vysokej škole technickej a univerzite v Zürichu.

[2] Ak spúšťate Turbo Pascal na najnovších počítačoch Pentium, pri použití unitu Crt môže byť po spustení programu zahlásená chyba Error 200: Division by zero. Odstrániť ju možno použitím unitu FDelay pred Crt (FDelay sa nenachádza v štandardnej knižnici UNITS).

SUCET pomenúvajú v Pascale tú istú premennú. Názov objektu je dobré voliť si tak, aby nám napovedal o jeho úlohe, veľmi dlhé názvy však nie sú praktické.

Iná situácia je pri znakových reťazcoch alebo skrátene len reťazcoch. **Reťazec** je text uzavretý v apostrofoch. Napríklad: 'Súčet čísel = ', 'Zadaj tri celé čísla: ', 'Najväčšie číslo je ', 'Vlož svoje meno'. V reťazcoch sa rozlišujú veľké a malé písmená, dovolené sú diakritické znamienka, medzery, interpunkčné znamienka. Príkazom write možno reťazce zobraziť na obrazovke v tvare, ako sú zapísané, uľahčujú najmä komunikáciu medzi užívateľom a programom.

S identifikátormi a reťazcami sa budeme v programoch „stretávať na každom kroku“. Identifikátory budeme, aspoň zo začiatku, písať veľkými písmenami, aby sa čitateľ ľahšie zorientoval. A teraz jednoduchý program na zoznámenie sa so štruktúrou, spúšťaním a úpravou programov.

Príklad S-1: Vytvorte algoritmus a program, ktorý pozdraví vypísaním slova Ahoj! na obrazovke.

```
algoritmus POZDRAV_1;          program POZDRAV_1;
začiatok                       begin
píš('Ahoj');                   write('Ahoj!');
koniec.                        end.
```

V ľavom stĺpci je slovný zápis algoritmu a v pravom program – prakticky preklad do angličtiny. Po spustení Turbo Pascalu (TP) pomocou súboru turbo.exe v adresári BIN odporúčame stlačiť F3 Open (alebo F10 – File – Open...) a hneď zadať meno súboru, do ktorého chceme program uložiť. Je dobré si uvedomiť rozdiel medzi menom programu a menom súboru, v ktorom je uložený daný program. Meno programu je identifikátor, ktorý môže mať aj viac ako osem znakov. Meno súboru môže mať najviac osem dovolených znakov (súbor s programom POZDRAV_1 môžete nazvať napríklad s-1 alebo pozdrav1; príponu .pas pridá Turbo Pascal sám). Po zadaní mena súboru a kliknutí na tlačidlo Open alebo stlačení klávesu Enter Turbo Pascal automaticky prejde do editora („modré“ okno), slúžiaceho na písanie a upravovanie programov (ak sa v editore zobrazil text, znamená to, že so zadaným názvom už súbor v adresári existuje a práve sa otvoril – pre nový program musíme zvoliť iné meno súboru). Po napísaní programu sa môžeme pokúsiť spustiť program stlačením kláves Ctrl+F9 alebo kliknutím na Run – Run (horné menu TP možno zaktívniť stlačením klávesu F10). Po odstránení prípadných chýb sa vykonajú príkazy programu a automaticky sa TP prepne do editora. Výstupy na obrazovku, v našom prípade pozdrav Ahoj!, je na užívateľskej obrazovke, do ktorej prepneme stlačením Alt+F5 alebo F10 - Debug - User screen. Po prezretí výsledku práce programu stlačením ľubovoľného klávesu sa vrátime do prostredia TP, presnejšie editora.

Ak chceme ukončiť prácu v prostredí TP, môžeme použiť položky horného menu F10 – File – Exit alebo klávesovú skratku Alt+X. Pri neuložených programoch sa nás TP pred ukončením opýta na uloženie. Uzavrieť súbor (program) bez ukončenia TP možno stlačením Alt+F3 alebo kliknutím na zeleno-žltý štvorček v ľavom hornom rohu okna (v editore).

Ďalšia verzia programu POZDRAV demonštruje použitie príkazu ClrScr (Clear Screen), slúžiaceho na zmazanie obrazovky. Príkaz clrscr nepatrí do štandardu TP, nachádza sa v jednotke (v unite) Crt, ktorá obsahuje rozširujúce príkazy pre prácu s obrazovkou, klávesnicou, kurzorom, zvukom atď. Jednotlivé príkazy a funkcie unitu crt si môžeme prezrieť po vyvolaní kontextového helpu (po stlačení Ctrl+F1 s kurzorom v slove crt) a výbere položky Crt – Procedures. Dôležité je vedieť, že pri použití príkazov z unitu musíme hneď za menom programu uviesť uses (použi) a meno unitu, v našom prípade crt. Do programu sme vložili aj poznámky, ktoré sa v TP vkladajú medzi množinové zátvorky – prekladač všetko medzi { a }

ignoruje[1]. Rovnakú funkciu plnia aj okrúhle zátvorky s hviezdičkami, t.j. zápisy (* a *).

```
program POZDRAV_2;
uses crt; { unit-jednotka obsahujuca prikazy rozsirujuce standard Pascalu }
begin
clrscr;   { prikaz na zmazanie obrazovky, nachadza sa v jednotke crt }
write('Ahoj!');
end.
```

Tretia verzia programu odstraňuje nepohodlné prepínanie medzi užívateľskou obrazovkou a editorom. Príkaz vstupu readln (podrobnejšie sa ním budeme zaoberať neskôr) zastaví beh programu, môžeme si pozrieť výstup na užívateľskej obrazovke, a až po stlačení klávesu Enter program pokračuje resp. skončí, keďže nasleduje end s bodkou (bodka signalizuje prekladaču: koniec programu).

```
program POZDRAV_3;
uses crt;
begin
clrscr;
write('Ahoj!');
readln    { pocitac caka na stlacenie klavesu Enter,
            pri verzii TP7 mozete pouzit aj prikaz readkey }
end.
```

V ďalšej verzii programu POZDRAV chceme, aby sa nás počítač opýtal na meno a slušne nás pozdravil: „Ahoj, naše meno !“. Nemôžeme zvoliť konkrétne meno, lebo sa všetci nevoláme rovnako. Preto musíme použiť premennú, ktorá bude nadobúdať hodnoty raz Jano, potom Miša, Peter, Alexandra,... Počítač musí mať v pamäti vyhradené miesto pre hodnotu každej premennej (ako by si ju ináč pamätal?). Aby vedel, že má rezervovať pamäťové miesto a aké veľké, musíme mu to povedať. Robí sa to tzv. deklaráciou premenných, ktorá začína slovkom var (variable – premenná). Cez meno premennej sa odvolávame na priradené pamäťové miesto a jeho veľkosť určujeme typom premennej. Ak je hodnotou premennej celé číslo, hovoríme o type integer, ak reálne číslo, hovoríme o type real. O údajových typoch si povieme neskôr. Hodnotou našej premennej MENO je reťazec znakov, anglicky string. Pri použití typu string sa v pamäti vyhradí miesto až pre 255 znakov (má niekto dlhšie krstné meno, asi nie).

```
program POZDRAV_4;
uses crt;
var  MENO:string;
begin
clrscr;
writeln('Ako sa volas ?');
readln(MENO);
writeln('Ahoj, ',MENO,'!');
readln
end.
```

Posledná verzia programu POZDRAV je určená pre tých, ktorí si chcú vyskúšať príkaz GotoXY z unitu Crt. Potrebujete vedieť, že obrazovka má štandardne 80 znakov v riadku (stĺpcov v smere x) a 25 riadkov na strane (v smere -y).

```
program POZDRAV_5;
uses crt;
var  MENO:string;
```

[1] Dopustili sme sa malej nepresnosti, lebo keby prvým znakom za zátvorkou bol $, prekladač by očakával tzv. direktívu.

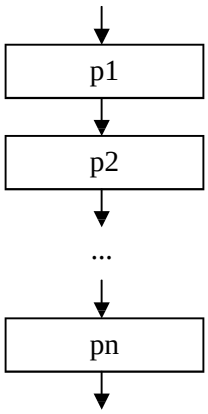
```
begin
clrscr;
writeln('Ako sa volas ?');
readln(MENO);
clrscr;
gotoxy(35,12);
writeln('Ahoj, ',MENO,'!');
readln
end.
```

To bola odbočka do prostredia Turbo Pascalu a k prvému programu a teraz späť k algoritmickým konštrukciám.

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 TP:

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.

Každá z verzií programu POZDRAV (príklad S-1) obsahuje príkazy, ktoré sa po spustení programu vykonajú postupne za sebou, ako sú zapísané. Príklad S-1 je príkladom na sekvenciu.

Príkaz priradenia

Slúži na priradenie hodnoty premennej.

Má tvar: **p := v** kde p je premenná a v výraz; symbol „:=“ sa číta „priraď“

Vykonanie: vyhodnotí sa výraz na pravej strane a jeho hodnota sa priradí premennej na ľavej strane príkazu priradenia.

Príklady: A := 7; MENO := 'Peter'; I := 5; I := I + 1; OBSAH := A*A (hviezdička je symbol pre násobenie)

Je dobré predstaviť si, čo sa v skutočnosti v počítači udeje. Napríklad pre príkaz I := I + 1. Premenná I zaberá v pamäti určité pamäťové miesto, znázornime si ho takto: I [ ] Tu je zapamätaná hodnota premennej I. Po spustení programu tam môže byť náhodné číslo, v Turbo Pascale vo verzii 7 tam je automaticky vložená 0. V našom prípade nech tam je hodnota 5, t.j. I [5]

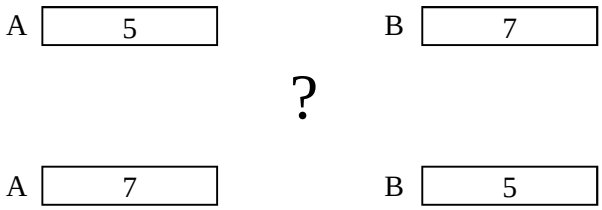
Príkaz I := I + 1 zoberie hodnotu premennej I, t.j. 5, zväčší ju o 1 na 6 a túto hodnotu zapíše do premennej na ľavej strane príkazu priradenia, teda do I [6] . Predchádzajúca hodnota 5 je nenávratne stratená!

Aj keď to je asi najjednoduchší a najčastejší príkaz, dobre si ho premyslite. Zapamätajte si, že sa vykonáva sprava doľava a predchádzajúca hodnota premennej na ľavej strane sa prepíše novou hodnotou.

Príkaz priradenia má rovnaký tvar v algoritmizácii aj v Pascale, tu sa však vyžaduje, aby premenná p a výraz v boli rovnakého typu, čo je pochopiteľné, lebo ťažko by sa napríklad výraz typu string (reťazec znakov) priraďoval premennej typu integer (celé čísla); dovolená je len jedna výnimka: celé číslo (integer) môže byť priradené premennej pre reálne čísla (typu real).

Príklad: Napíšte postupnosť (sekvenciu) príkazov priradenia, ktoré vymenia hodnoty dvoch premenných. Napríklad premenná A nech má hodnotu 5 a premenná B hodnotu 7. Po vykonaní sekvencie nech je v A hodnota 7 a v B hodnota 5.

Analýza:



Aj keď nechceme čitateľa podceňovať, sekvencia

```
A := B;
B := A;
```

nie je správna (odsimulujte si, čo sa udeje v počítači). Ak si uvedomíte prečo, správne riešenie ľahko zistíte. Vykonaním príkazu A := B sa prepíše pôvodná hodnota premennej A (5), ktorú však ešte potrebujeme. Riešením je odložiť si ju skôr, než dôjde k jej prepísaniu, niekde „bokom“, do pomocnej premennej napr. POM. Správna sekvencia: POM := A; A :=B; B := POM;

Poznámky:

1. Úloha má ešte jedno analogické riešenie (POM := B;...).
2. Ak hodnoty premenných A a B sú čísla, nemusíme použiť ani tretiu premennú (napr.: A := A-B; B := B+A; A := B-A;).
3. Úloha výmeny hodnôt dvoch premenných je analogická s úlohou výmeny tekutín (napr. voda a mlieko) v dvoch pohároch navzájom.

Príklad S-2: Vytvorte algoritmus a program na výpočet obsahu a obvodu obdĺžnika.

Analýza: Algoritmus často rieši určitý problém z nejakej vednej disciplíny, v tomto prípade z matematiky. Preto treba mať znalosti z daného vedného odboru, v našom prípade poznať vzorce na výpočet obsahu a obvodu obdĺžnika. Ďalej je dobré si uvedomiť, čo musíme poznať pre daný výpočet (rozmery obdĺžnika - vstupné údaje) a čo očakávame, že vykonaním algoritmu získame (obsah a obvod - výstupné údaje). Keďže základnou vlastnosťou algoritmu je hromadnosť, vstupné aj výstupné údaje zadávame cez premenné. Pre rozmery obdĺžnika sú to bežne písmená A a B, pre obsah a obvod si zvolíme názvy premenných OBSAH a OBVOD. Pri zostavovaní algoritmu uvažujte, ako by ste daný problém riešili bez počítača („manuálne“) a tento postup zapíšte vo forme algoritmu.

manuálne	algoritmus	program

Začínam
potrebujem vedieť rozmery obdĺžnika
použijem vzorec na výpočet obsahu S = a.b
použijem vzorec na výpočet obvodu
O = 2.(a+b)
oznámim výsledok
skončil som.
	začiatok
	čítaj (A, B);
	OBSAH := A*B;
	
	OBVOD := 2*(A+B);
	píš (OBSAH, OBVOD);
	koniec.
		begin
		readln(A, B);
		OBSAH := A*B;
		
		OBVOD := 2*(A+B);
		write(OBSAH, OBVOD);
		end.

Program treba doplniť o deklaráciu premenných. V prvej alternatíve budeme počítať len s celými číslami (typ integer), v druhej s reálnymi (typ real). Programy sú doplnené o užívateľsky komfort, pod ktorým rozumieme predovšetkým pred každý príkaz vstupu read dať príkaz výstupu write, ktorý vypíše na obrazovke, čo treba zadať a tiež jednoznačný výstup výsledných hodnôt.

Program očakáva vloženie dvoch celých čísel, ktoré môžeme vložiť naraz, oddelené medzerou (nie čiarkou!) alebo napíšeme jedno, stlačíme Enter a potom druhé a Enter.

```
program OBDLZNIK_CELE;
uses crt;
var A, B, OBSAH, OBVOD: integer;
begin
clrscr;
write('Zadaj strany obdlznika (cele cisla): ');
readln(A,B);
OBSAH:=A*B;
OBVOD:=2*(A+B);
writeln('Obsah obdlznika = ',OBSAH);
writeln('Obvod obdlznika = ',OBVOD);
readln
end.
```

Typickou chybou začiatočníkov je, že po odladení programu (odstránení chýb, ktoré nájde prekladač TP) už ďalej „slepo veria počítaču“ a program ďalej netestujú. Musíte sa prinútiť po napísaní programu ho otestovať, t.j. zadať čo najviac takých vstupných hodnôt, pre ktoré poznáte výsledky a tiež vložiť tzv. hraničné hodnoty, ktoré sú na hranici dovolených vstupných hodnôt alebo pri ktorých sa mení charakter výpočtu. Program OBDLZNIK by sme mohli otestovať pre A a B rovné 0, 1 a napríklad 2 a 4 (2.4=8 a 2.(2+4)=12) alebo iné čísla, pre ktoré si ľahko spamäti zistíme správne výsledky. Skúste aj A = 1000 a B = 1000. Ak máte zapnuté kontroly chýb počas behu programu (ako sme to odporučili na strane 6), počítač ukončí výpočet chybovým hlásením Error 215: Arithmetic overflow – preplnenie, výsledkom výpočtu je číslo väčšie ako preň vyhradené pamäťové miesto. Prečo je tomu tak, si povieme pri údajových typoch na nasledujúcej strane. Ak kontrolu nemáte zapnutú, možno si zlý výsledok 16 960 (pre obsah) ani nevšimnete! Takže dôveruj ale preveruj!

```
program OBDLZNIK_REALNE;
uses crt;
var A, B, OBSAH, OBVOD: real;
begin
clrscr;
write('Zadaj strany obdlznika: ');
readln(A,B);
OBSAH:=A*B;
OBVOD:=2*(A+B);
writeln('Obsah obdlznika s presnostou na stotiny = ',OBSAH:4:2);
writeln('Obvod obdlznika s presnostou na stotiny = ',OBVOD:4:2);
readln
end.
```

Reálne čísla vkladáme podobne ako celé, len ich zápis môže obsahovať aj desatinnú bodku. Znova všetkému nerozumiete, ale naraz sa všetko nedá. Tie tajomné čísla :4:2 objasníme pri príkaze write.

Údajové typy

Jednoduché štandardné údajové typy:

Meno typu: **Integer** (ordinálny typ)

Množina hodnôt: celé čísla z intervalu –32 768 po 32 767 (konštanta MaxInt)

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:

V TP sú pre hodnoty typu integer vyhradené v pamäti 2 B, t.j. 16 bitov. V jednom bite sa zapamätáva znamienko (0+/1-) a do zvyšných pätnástich bitov sa ukladá hodnota bez znamienka pomocou 0 a 1. Všetkých možností je 2^{15} = 32 768. Preto do daného pamäťového miesta možno uložiť všetky nezáporné celé čísla od 0 po 32 767 alebo záporné celé čísla od –1 po – 32 768. Ak je výsledkom operácie s typom integer číslo mimo tento rozsah, nastane v príklade S-2 opísaná chyba, tzv. preplnenie (v Options – Compiler... prepínač Overflow Checking – kontrola preplnenia). Ak nám uvedený rozsah nevyhovuje, môžeme použiť ďalšie celočíselné typy: Shortint, Longint, Byte a Word, ktorých rozsahy si môžete pozrieť pomocou kontextového helpu (Ctrl+F1 s kurzorom pod napr. integer). Najväčší rozsah má typ longint, kde konštanta MaxLongInt má hodnotu 2 147 483 647 (= 2^{31} –1).

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 = C a A mod B = D potom B.C + D = A a 0<=D<B. Pre záporné celé čísla funkcie div a mod v TP pracujú ináč ako v matematike!

Výsledkom: 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 zaokrúhlené reálne číslo x, napr. round(8.9) = 9, round(-5.6) = -6, round(0.5) = 1
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 približne $-1{,}7.10^{38}$ po $1{,}7.10^{38}$

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.

Ďalšie údajové typy pre reálne čísla sú single, double, extended a comp, ich použitie je však podmienené prítomnosťou alebo emuláciou matematického koprocesora (funkcie v F10 – Options - Compiler... – Numeric processing).

Niektoré funkcie sú popísané vyššie, u typu integer, a málo používané funkcie nepopíšeme.

Výsledkom: 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

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, 0 a True – pravda, nie 0; false < true.

pred(true) = false, succ(false) = true; ord(false) = 0, ord(true) = 1

Meno typu: **Char** (ordinálny typ)

Množina hodnôt: znaky tabuľky ASCII

Funkcie: upcase(z)

succ(z), pred(z), ord(z), chr(x)

Relačné operátory: <, <=, =, <>, >=, >, in

Vysvetlivky:

Znaky tabuľky ASCII aj s poradovými číslami možno zobraziť pomocou programu:

```
program ZNAKY_ASCII;
uses crt;
var I:byte;
begin
clrscr;
for I:=32 to 255 do
  write(I:6,chr(I):2);              { vypis v tabulke }
```

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

```
readln;
clrscr;
for I:=1 to 255 do
  begin
  writeln(I:6,chr(I):4);           { vypis pod sebou }
  if I mod 23 = 0 then readln      { zastavi vypis vzdy po 23 riadkoch }
  end
end.
```

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.

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 TP 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.

Štruktúrovaný údajový typ:

Meno typu: **string** alebo **string[n]** kde n je celé číslo od 1 po 255 – udáva maximálnu dĺžku reťazca

Množina hodnôt: reťazce zo znakov ASCII

Dovolené operácie: + (spája reťazce)

Funkcie: length, copy, delete, concat, insert, pos

Relačné operátory: <, <=, =, <>, >=, >

Vysvetlivky:

String vyhradí v pamäti miesto pre 255 znakový reťazec (pre najviac 255 znakov), napríklad string[5] vyhradí v pamäti miesto pre päť znakový reťazec (pre najviac 5 znakov, ostatné nebudú zapamätané).

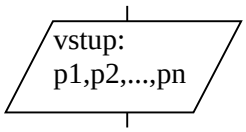
Výsledkom funkcie length(r) je celé číslo - dĺžka reťazca r. Prázdny reťazec '' má dĺžku 0.

Pri relačných operáciách sa najprv porovnajú prvé znaky v porovnávaných reťazcoch, ak sú zhodné, druhé atď.

Príkazy vstupu, príkaz read

Príkazy vstupu slúžia na vstup údajov z klávesnice.

Majú tvar:



čítaj (p1, p2, ... , pn)

kde p1 až pn sú premenné

V TP sú to modifikácie príkazu **read**:

read (p1, p2, ... , pn)
readln (p1, p2, ... , pn) kde p1 až pn sú premenné
readln

Vykonanie: zastaví sa beh programu a počítač čaká na vloženie príslušného počtu hodnôt v stanovenom poradí a deklarovaných typov.

Napríklad: readln(A,B,C); readln(TEXT); readln(X); readln(MENO); readln(ODPOVED); readln;

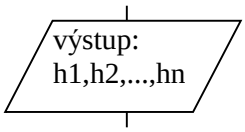
Poznámky:

- príkazy ukončené ln (line) možno interpretovať: načítaj hodnoty premenných p1 až pn a nastav čítanie na nový riadok
- jednotlivé číselné hodnoty môžu byť oddelené medzerami alebo vkladané v samostatných riadkoch zadaním hodnoty a stlačením klávesu Enter
- „prázdny“ príkaz readln neočakáva hodnotu, len stlačenie klávesu Enter
- príkaz read(p1, ... , pn) nebudeme predbežne používať
- po zadaní väčšieho čísla, ako je dovolený rozsah daného typu, sa zobrazí chybové hlásenie Error 201: Range check error - prekročenie rozsahu (v Options – Compiler... prepínač Range Checking – kontrola rozsahu hodnôt). Napríklad u typu integer po zadaní čísla väčšieho ako je maxint (32 767). Ak nedošlo k zobrazeniu chybového hlásenia, pozri str. 6
- ak pri zadávaní hodnôt zadaná hodnota nezodpovedá očakávanému typu, napr. u číselného typu vložíme miesto číslice písmeno, zobrazí sa chybové hlásenie Error 106: Invalid numeric format – zlý číselný formát (v Options – Compiler... prepínač I/O Checking – kontrola vstupno/výstupných operácií)

Príkazy výstupu, príkaz write

Príkazy výstupu slúžia na zobrazenie výstupných hodnôt na obrazovke monitora.

Majú tvary:



píš (h1, h2, ... , hn)

kde h1 až hn sú výstupné hodnoty

V TP sú to modifikácie príkazu write:

write (f1, f2, ... , fn)
writeln (f1, f2, ... , fn) kde f1 až fn sú formátovacie výrazy
writeln

Vykonanie: na obrazovke monitora sa zobrazia výstupné hodnoty v predpísaných formátoch (buď hodnoty premenných alebo text pôvodne uvedený v apostrofoch).

Napríklad: writeln('Dĺžka čiary je ',DLZKA); write('Zadaj kladné číslo: '); writeln(A,' ',B); writeln;

Poznámky:

- príkazy ukončené ln (line) možno interpretovať: zobraz výstupné hodnoty a nastav výstup (kurzor) na začiatok nového riadku
- „prázdny“ príkaz writeln nastaví výstup na nový riadok

Formátovacie výrazy:

1) h — h je výraz reprezentujúci výstupnú hodnotu
2) h : pzv — h ako vyššie, pzv je výraz reprezentujúci počet znakov výstupu
3) h : pzv : pdm — h a pzv ako vyššie, pdm je výraz reprezentujúci počet desatinných miest výstupu u typu real

Napríklad:

a) nech A je premenná typu integer

hodnota A	príkaz	na obrazovke	poznámka
12345	write(A)	12345	znamienko plus sa nezobrazuje
-12345	write(A)	-12345	znamienko mínus zaberá jeden znak
12345	write(A:10)	12345	zľava sa doplní päť medzier
-12345	write(A:7)	-12345	zľava sa doplní jedna medzera
12345	write(A:3)	12345	ignoruje počet znakov výstupu

b) nech X je premenná typu real (matematický koprocesor vypnutý)

hodnota X	príkaz	na obrazovke	poznámka
123,456789	write(X)	1.2345678900E+02	tzv. semilogaritmický tvar, E – exponent sa číta „krát desať na“, pre „+“ vynechaný znak
123,456789	write(X:10)	1.235E+02	neprehliadnite zaokrúhlenie mantisy
123,456789	write(X:2)	1.2E+02	minimálny výpis s exponentom (8 znakov)
123,456789	write(X:6:2)	123.46	6 znakov výstupu, 2 za des. bodkou
123,456789	write(X:4:2)	123.46	ignoruje počet znakov výstupu
123,456789	write(X:10:3)	123.457	zľava sa doplnia tri medzery
123,456789	write(X:12:10)	123.4567890000	maximálna presnosť zobrazenia
123,456789	write(X:5:0)	123	zľava sa doplnia dve medzery
0,0000987654321	write(X)	9.8765432100E-05	semilogaritmický tvar
0	write(X)	0.0000000000E+00	semilogaritmický tvar

c) nech S je premenná typu string

hodnota S	príkaz	na obrazovke	poznámka
Janosik	write(S)	Janosik	
Janosik	write(S:3)	Janosik	ignoruje počet znakov výstupu
Janosik	write(S:10)	Janosik	zľava sa doplnia tri medzery

d) nech Q je premenná typu boolean

hodnota Q	príkaz	na obrazovke	poznámka
true	write(Q)	TRUE	
false	write(Q:10)	FALSE	zľava sa doplní päť medzier
true	write(Q:1)	TRUE	ignoruje počet znakov výstupu

Poznámky:

- neprehliadnite zaokrúhľovanie reálnych čísel na zadaný počet desatinných miest
- bez použitia formátovacích výrazov napríklad v príkaze write(A,B), kde A a B sú typu integer, neodlíšite od seba cifry kladných čísel! Okrem formátovania je riešením aj vloženie medzery, t.j. write(A,' ',B).
- formátovacie výrazy sú výrazy, t.j. môžu mať aj tvar napríklad writeln(A*B), write(X : PRESNOST+2 : PRESNOST), kde premenná PRESNOST typu integer môže nadobúdať hodnoty 0 (jednotky), 1 (desatiny), 2 (stotiny), ... , 10
- ujasnite si vykonanie príkazu writeln('Obvod = ':40,2*(A+B):4:2)

Vráťme sa však k sekvencii.

Príklad S-3: Vytvorte program simulujúci palubný počítač auta. Známy je objem palivovej nádrže a priemerná spotreba paliva počas jazdy. Vždy pri natankovaní plnej nádrže sa vynuluje tzv. denné počítadlo kilometrov. Po zadaní prejdených kilometrov (z denného počítadla kilometrov) počítač vypočíta aktuálny dosah auta. Po zadaní vzdialenosti do cieľa vypočíta priemernú dobu jazdy.

Analýza: Dosah auta pri plnej nádrži sa vypočíta ako podiel: objem palivovej nádrže / priemerná spotreba paliva na 100 km, krát 100 km. Keďže tento podiel nemusí byť celé číslo, zaokrúhlime ho na celé km, samozrejme nadol. Aktuálny dosah sa zrejme rovná: dosah pri plnej nádrži – prejdené km. Priemerná doba jazdy sa vypočíta ako podiel: vzdialenosť do cieľa / priemerná rýchlosť za hodinu (konštanta). Výsledok je v hodinách a zobrazíme ho s presnosťou na desatiny hodiny.

```
program POCITAC;
uses  crt;
const ObjemPalNadrze = 46;    { litrov }
      PriemSpotPaliva = 6.5;  { litra na 100 km }
      PriemRychlost = 90;     { km/hod. }
var   PrejdeneKM, AktDosah, Dosah, VzdDoCiela : integer;
                             PriemDobaJazdy : real;
begin
clrscr;
write('Prejdene kilometre (z denneho pocitadla km): ');
readln(PrejdeneKM);
Dosah:=trunc(ObjemPalNadrze/PriemSpotPaliva*100);
AktDosah:=Dosah-PrejdeneKM;
writeln('Aktualny dosah: ',AktDosah,'km');
writeln;
write('Vzdialenost do ciela: ');
readln(VzdDoCiela);
PriemDobaJazdy:=VzdDoCiela/PriemRychlost;
writeln('Priemerna doba jazdy: ',PriemDobaJazdy:3:1,' hod.');
readln
end.
```

Príklad S-4: Vytvorte program na „uhádnutie“ celého čísla z intervalu 0 až 100, ak viete, že po zadaní zvyškov po delení 3 (zvysok3), 5 (zvysok5) a 7 (zvysok7) mysleného čísla sa myslene číslo rovná zvyšku po celočíselnom delení výrazu (70*zvysok3+21*zvysok5+15*zvysok7) 105-timi. Strašné, ani len zadaniu sa nedá rozumieť!

Analýza: Treba zadať zvyšky po delení 3, 5 a 7 mysleného čísla – to sú vstupné údaje a my pre ne zvolíme vstupné premenné označené zvysok3, zvysok5 a zvysok7. Myslené číslo sa rovná zvyšku po delení výrazu...,

preto si tento výraz vypočítajme. VYRAZ = 70*zvysok3+21*zvysok5+15*zvysok7. Myslené číslo sa rovná zvyšku po celočíselnom delení VYRAZu 105-timi. Zvyšok po celočíselnom delení nám dáva funkcia mod. A tu je program:

```
program UHADNEM_CISLO;
uses crt;
var zv3,zv5,zv7,myslene_cislo:byte;
begin
clrscr;
writeln('U H A D N E M   M Y S L E N E   C I S L O  D O  100 !':60);
write('Zvysok po deleni 3: '); readln(zv3);
write('Zvysok po deleni 5: '); readln(zv5);
write('Zvysok po deleni 7: '); readln(zv7);
myslene_cislo:=(70*zv3+21*zv5+15*zv7) mod 105;
writeln;
writeln('Myslel si si cislo ',myslene_cislo);
readln
end.
```

Príklad S-5: Vytvorte program, ktorý prepočíta čas v sekundách na hodiny : minúty : sekundy.

Analýza: Vieme, že hodina má 3 600 sekúnd. Koľkokrát sa 3 600 nachádza v sekundách (bezo zvyšku), je hľadaný počet hodín – to vie vypočítať funkcia div. Funkcia mod určí zvyšok sekúnd, čo zostal na minúty a sekundy. Počet minút zistíme opäť celočíselným delením zvyšných sekúnd číslom 60. No a to čo zostane, sú už len sekundy (< 60). Výpočet funkciou mod možno nahradiť aj príkazom CAS := CAS – 3600*H resp. CAS := CAS – 60*M.

```
program CAS_V_SEKUNDACH;
uses crt;
var  H, M, CAS : longint;
begin
clrscr;
write('Zadaj cas v sekundach: ');
readln(CAS);
H:=CAS div 3600; CAS:=CAS mod 3600;
M:=CAS div 60;   CAS:=CAS mod 60;
writeln(H,' : ',M,' : ',CAS);
readln
end.
```

Úloha: Dokážete upraviť predchádzajúci program tak, aby počítal iba s jednou premennou CAS?

```
program CAS_V_SEKUNDACH_2;
uses crt;
var  CAS:longint;
begin
clrscr;
write('Zadaj cas v sekundach: '); readln(CAS);
gotoxy(34,12);
writeln(CAS div 3600,' : ',CAS mod 3600 div 60,' : ',CAS mod 3600 mod 60);
readln
end.
```

Neriešené úlohy na sekvenciu:

1. Vytvorte program na výpočet obsahu a obvodu štvorca s presnosťou na desatiny.
   Návod: Vzorec pre výpočet obsahu $S = a^2$ a obvodu $O = 4a$.
2. Vytvorte program na výpočet obsahu a obvodu kruhu s presnosťou na tisíciny.
   Návod: Vzorec pre výpočet obsahu $S = \pi.r^2$ a obvodu $O = 2\pi.r$. TP pozná konštantu PI = 3,1415...
3. Vytvorte program na výpočet objemu, povrchu a telesovej uhlopriečky kvádra s presnosťou na stotiny.
   Návod: Vzorec pre výpočet objemu $V = a.b.c$, povrchu $S = 2(a.b+a.c+b.c)$, uhlopriečky $u = \sqrt{a^2+b^2+c^2}$.
4. Vytvorte program na výpočet objemu a povrchu gule s najväčšou možnou presnosťou.
   Návod: Vzorec pre výpočet objemu $V = 4/3\pi.r^3$, povrchu $S = 4\pi.r^2$.
5. Vytvorte program na výpočet počtu 10, 5, 2 a 1 Sk mincí na vyplatenie zadanej sumy (použite funkcie div a mod).
   Poznámka: Pozri program CAS_V_SEKUNDACH.
6. Vytvorte program na nákup valút za určitú sumu korún. Zmenáreň si účtuje manipulačný poplatok vo výške 1% z vloženej sumy. Skutočne platená čiastka sa zaokrúhľuje na celé koruny. Názov a kurz meny si zvoľte sami.
7. Priemerná denná teplota sa počíta ako aritmetický priemer teplôt nameraných o 6-tej ráno, o 12-tej a o 18-tej hodine, pričom údaj nameraný o 18-tej hodine sa započítava dvakrát. Vytvorte program, ktorý zo zadaných údajov vypočíta priemernú dennú teplotu.
8. Sú zadané odpory dvoch rezistorov v ohmoch. Určte hodnotu výsledného odporu pri ich sériovom a paralelnom zapojení.
   Návod: Pri sériovom zapojení $R = R_1+R_2$, pri paralelnom $R = R_1.R_2 / (R_1+R_2)$.
9. Vytvorte program na riešenie lineárnej rovnice $ax + b = 0$ a, b reálne, $a \neq 0$.
   Návod: Koreň $x = -b/a$.
10. Vytvorte program na prevod veľkosti uhla v radiánoch na stupne a minúty.
    Návod: π radiánov = 180^0, 1^0=60'. TP pozná konštantu PI.

VETVENIE, ROZHODOVANIE

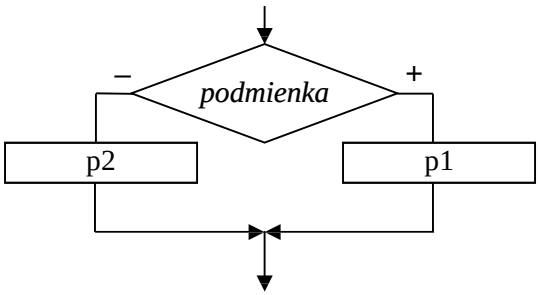
Vetvenie použijeme, ak vykonanie príkazu (alebo skupiny príkazov) je podmienené splnením alebo nesplnením určitej podmienky.

S rozhodovaním sa stretávame v bežnom živote na každom kroku. Ak bude teplo, pôjdem na kúpalisko, inak budem sedieť za počítačom. Vykonanie akcie „pôjdem na kúpalisko“ je podmienené splnením podmienky „bude teplo“. Vykonanie akcie „budem sedieť za počítačom“ je podmienené nesplnením podmienky!

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

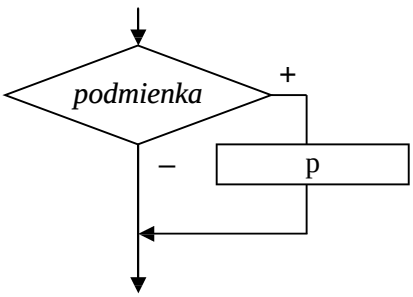
if b
then p1
else p2 kde b je výraz typu boolean, p1 a p2 sú príkazy

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

Napríklad:
```
if X < 0 then writeln('Číslo je záporné') else writeln('Číslo je nezáporné')
if PRIESTUPNY_ROK then DNI := 29 else DNI := 28
if (ZNAK >= 'A') and (ZNAK <= 'Z') then writeln('veľké písmeno') else writeln('neviem')
if odd(CISLO) then writeln('nepárne') else writeln('párne')
if true then writeln('vždy toto') else writeln('toto nikdy')
if A mod B = 0 then writeln(A,' je deliteľné ',B) else writeln(A,' nie je deliteľné ',B)
```

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 TP 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.

Napríklad: if MENO = 'Peter' then writeln('Ďalší Peter')

if CISLO > NAJVACSIE then NAJVACSIE := CISLO

if (CISLO > MaxInt) or (CISLO <= -MaxInt) then writeln('mimo rozsah integer')

if ord(ZNAK) < 32 then writeln('riadiaci znak')

if X = HLADANE_CISLO then POCET := POCET + 1

Príklad VIF-1: Vytvorte program na nájdenie najväčšieho z troch celých čísel.

Analýza: Znova odporúčame algoritmus tohto problému zostaviť obyčajným „sedliackym“ rozumom. Čo spravíme prvé, keď sa dozvieme čísla, z ktorých treba nájsť najväčšie? Asi porovnáme prvé dve čísla a z nich vyberieme väčšie (ak sú rovnaké, je jedno, ktoré vyberieme) a to následne porovnáme s tretím číslom. A máme najväčšie – maximum. Algoritmus je na strane 4.

```
program MAXIMUM_1;
uses crt;
var A, B, C, MAX : integer;
begin
clrscr;
write('Zadaj tri cele cisla: ');
readln(A,B,C);
if A>B
  then MAX:=A
  else MAX:=B;
if C>MAX
  then MAX:=C;
writeln('Z cisel ',A,', ',B,', ',C,' je najvacsie ',MAX,'.');
readln
end.
```

Aby sme demonštrovali variabilnosť riešenia už tak jednoduchého problému, uvádzame ďalšie programy riešiace rovnakú úlohu.

Iná možnosť (jej výhodou je jednoduchosť a použiteľnosť pre ľubovoľný počet čísel, treba len opakovať príkaz if ? > MAX then MAX := ?):

```
program MAXIMUM_2;
uses crt;
var A, B, C, MAX : integer;
begin
clrscr;
write('Zadaj tri cele cisla: ');
readln(A,B,C);
MAX:=A;      { priradenie pociatocnej hodnoty premennej MAX }
if B>MAX
  then MAX:=B;
if C>MAX
  then MAX:=C;
writeln('Z cisel ',A,', ',B,', ',C,' je najvacsie ',MAX,'.');
readln
end.
```

V nasledujúcom programe je logická chyba, viete ju nájsť?

```
program MAXIMUM_3;
uses crt;
var A, B, C : integer;
begin
clrscr;
write('Zadaj tri cele cisla: ');
readln(A,B,C);
write('Z cisel ',A,', ',B,', ',C,' je najvacsie ');
if (A>B) and (A>C)        { preco sme presunuli vystup (v nom nie je chyba!)? }
  then writeln(A);
if (B>A) and (B>C)
  then writeln(B);
if (C>A) and (C>B)
  then writeln(C);
readln
end.
```

Takto to je už dobre?

```
program MAXIMUM_4;
uses crt;
var A, B, C : integer;
begin
clrscr;
write('Zadaj tri cele cisla: ');
readln(A,B,C);
write('Z cisel ',A,', ',B,', ',C,' je najvacsie ');
if (A>B) and (A>C)
  then writeln(A)
  else if (B>A) and (B>C)
         then writeln(B)
         else writeln(C);
readln
end.
```

Programátorský štýl

Ak sa pozriete na zápisy príkazov if aj celých programov, vidíte v nich určitú štruktúru, ktorou sa snažíme vyjadriť podriadenosť niektorých častí nad inými. Rovnocenné príkazy sú na jednej úrovni, podriadené časti sa snažíme posunúť napríklad o tri medzery vpravo. Zároveň takýto zápis sprehľadňuje celý algoritmus alebo program. Veľmi vám odporúčame používať určité logické schémy pri zápisoch programov a algoritmov. Budú pre všetkých prehľadnejšie, zrozumiteľnejšie, pri tvorbe spravíte menej chýb a ľahšie sa v nich budú hľadať chyby. Odporúčame dve schémy:

1. zápis if podmienka
   then príkaz1
   else príkaz2

   dobre graficky znázorňuje, že ide o úplné binárne vetvenie s oboma vetvami na samostatných riadkoch; riadky s then a else sú podriadené riadku if, kde príkaz začína, ďalší príkaz by bol už na úrovni if;

2. zápis if podmienka then príkaz1
   else príkaz2

   zdôrazňuje prítomnosť vetvy else, t.j. úplného binárneho vetvenia, vetva then je „schovaná“, keďže sa musí vyskytovať v každom príkaze if.

Pri písaní programov v editore TP si všimnite, že podporuje štruktúrovaný zápis.

Prakticky prekladaču TP je jedno, ako je daný program napísaný. Pre neho je rozhodujúca len prítomnosť bodkočiarok medzi jednotlivými príkazmi. Celý program by mohol byť napísaný v jednom riadku (ak by sa tam zmestil). Súčasťou dobrého programátora je aj dobrý programátorský štýl, aj čo sa týka zápisu programov, ktorý sa určite neprezentuje programom v jednom stĺpci alebo riadku!

A propos bodkočiarky. Platí: príkazy sa od seba oddeľujú bodkočiarkami. Preto za begin, pred end a inými vyhradenými slovami bodkočiarky nemusia byť, neoddeľovali by príkazy. V niektorých prípadoch sú dokonca chybou (uvidíme pri cykloch). My v programoch na miestach, kde nemusia byť, nebudeme dávať bodkočiarky, aj keď väčšinou by tam mohli byť.

Zložený príkaz

Ak si dobre všimnete príkaz if, za then aj else je dovolený len jeden (!) príkaz. Ak potrebujeme na týchto miestach použiť viacej príkazov, musíme vytvoriť tzv. zložený príkaz.

Zložený príkaz použijeme na mieste, kde je dovolený len jeden príkaz a my potrebujeme, aby došlo k vykonaniu viacerých príkazov. Zložený príkaz vytvára zo skupiny príkazov jeden príkaz.

Má tvar:	začiatok	v TP:	**begin**	
	p1;		**p1;**	
	p2;		**p2;**	
	...		**...**	
	pn;		**pn;**	
	koniec		**end**	kde p1 až pn sú príkazy

Vykonanie: Z príkazov p1, p2 až pn je vytvorený jeden (zložený) príkaz.

```
Napríklad: if A > B                    if  X<=0
             then begin                  then writeln('Pre X<=0 neviem vypočítať!')
                  POM := A;              else  begin
                  A := B;                      XnaN := exp( N*ln(X) );
                  B := POM;                    writeln( X, ' na ' , N , ' = ' , XnaN )
                  end                         end
           { po skončení A ≤ B }       { výpočet mocniny x^n }
```

Poradie vyhodnocovania operácií (priorita operátorov)

Všimnite si, že v programoch MAXIMUM 3 a 4 sme v podmienkach príkazov if použili zátvorky. Otázka znie: Kedy musíme použiť zátvorky v zápise výrazov a kedy nie? O tom rozhoduje priorita jednotlivých operátorov. Podobne ako v matematike má aj v TP násobenie a delenie „prednosť“ pred sčítaním a odčítaním a pod. Všeobecne platí, že najprv sa vyhodnotia funkcie (abs, sqr, sqrt, odd,...).

Priorita operátorov je nasledovná:

1. unárne mínus (zmena znamienka), not
2. *, / , div, mod, and
3. +, - , or, xor
4. <, <=, =, <>, >=, >, in

pri rovnosti operátorov sa postupuje zľava doprava; prvé sa vyhodnocujú výrazy v zátvorkách, od vnútorných k vonkajším.

Príklady: (I = N) or (ODP = 'K')
boolean boolean

Bez zátvoriek by došlo ku chybe, lebo or má vyššiu prioritu ako = a preto by sa najprv vykonalo N or ODP pričom N je číselný typ a ODP je typu char, čo nie sú typy boolean očakávané „okolo“ or.

Výraz $\frac{a+b}{c}$ musí byť zapísaný pomocou zátvoriek (A+B)/C, inak by došlo najprv k deleniu B/C (delenie má vyššiu prioritu ako sčítanie) a až potom k sčítaniu, čo nezodpovedá pôvodnému výrazu.

Výraz $\frac{a}{b.c}$ musí byť zapísaný A/(B*C), ináč by nezodpovedal pôvodnému výrazu, pretože zápis A/B*C (pravidlo „zľava doprava“) zodpovedá výrazu $\frac{a}{b}.c$ (čo nie je pôvodný výraz). Pôvodnému výrazu však zodpovedá aj zápis A/B/C!

Výraz $\frac{a.b}{c}$ môže byť zapísaný bez zátvoriek A*B/C (pravidlo „zľava doprava“; najprv sa vykoná násobenie a potom delenie).

Znova sme trochu múdrejší v teórii, vráťme sa však k príkladom využívajúcim vetvenie a príkaz if.

Príklad VIF-2: Vytvorte program na usporiadanie troch celých čísel (zostupne).

Analýza: Podľa potreby budeme vymieňať hodnoty premenných A, B, C, kým nebude A<=B<=C.

```
program USPORIADAJ;
uses crt;
var A, B, C, POM : integer;
begin
clrscr;
write('Zadaj tri cele cisla: ');
readln(A,B,C);
if A>B
  then begin POM:=A; A:=B; B:=POM end;
if B>C
  then begin POM:=B; B:=C; C:=POM end;
if A>B
  then begin POM:=A; A:=B; B:=POM end;
writeln(A,' <= ',B,' <= ',C);
readln
end.
```

Venujme sa ešte jednej „teoretickej“ otázke. V príkaze if nič nebráni tomu, aby na mieste príkazu p1 alebo p2 mohol byť znova, podľa potreby, použitý príkaz if, ako to vidieť v programe MAXIMUM_4.

Pri štruktúrach

```
if b1
  then if b2
         then p1
else p2
```

```
if b1
  then  if b2
          then p1
else p2
else p3
```

vzniká otázka, ku ktorému if patria vetvy else?

Platí: Vetva else patrí vždy k najbližšiemu predchádzajúcemu then.

Vyjadrené štruktúrovane:

```
if b1
  then if b2
         then p1
         else  p2
```

```
if b1
  then  if b2
          then p1
          else  p2
  else p3
```

Ak chceme, aby else p2 v prvej štruktúre patrilo prvému then musíme to prekladaču povedať zápisom:

```
if b1
  then begin
         if b2
           then p1
         end
  else  p2
```

Príklad VIF-3: Vytvorte program na výpočet výsledného odporu dvoch rezistorov zapojených sériovo alebo paralelne s presnosťou na stotiny.

Analýza: Z fyziky treba vedieť, že pri sériovom zapojení rezistorov výsledný odpor $R=R_1+R_2$, pri paralelnom zapojení $R = \frac{R_1 . R_2}{R_1 + R_2}$.

Doriešiť treba ešte zadanie spôsobu zapojenia. Rozhodli sme sa pre vloženie prvého písmena (typ char), presnejšie, pri vložení znaku p alebo P je zapojenie paralelné, inak sériové.

```
program VYSLEDNY_ODPOR;
uses crt;
var R1, R2, R : real;
        ZAPOJ : char;
begin
clrscr;
write('Zadaj hodnoty odporov R1 a R2: '); readln(R1,R2);
write('Sposob zapojenia (P - paralelne, iny znak - seriovo): ');
readln(ZAPOJ);
if (ZAPOJ='P') or (ZAPOJ='p')
  then R:=R1*R2/(R1+R2)
  else R:=R1+R2;
writeln('Vysledny odpor R = ',R:4:2);
readln
end.
```

Využitie funkcie upcase a upravenie výstupu:

```
program VYSLEDNY_ODPOR_2;
uses crt;
var R1, R2, R : real;
        ZAPOJ : char;
begin
clrscr;
write('Zadaj hodnoty odporov R1 a R2 v ohmoch: '); readln(R1,R2);
write('Sposob zapojenia (P - paralelne, iny znak - seriovo): ');
readln(ZAPOJ);
ZAPOJ:=upcase(ZAPOJ);
if ZAPOJ='P'
  then R:=R1*R2/(R1+R2)
  else R:=R1+R2;
writeln;
write('Vysledny odpor pri ');
if ZAPOJ='P'
  then write('paralelnom')
  else write('seriovom');
writeln(' zapojeni R = ',R:4:2,' ohmov.');
readln
end.
```

Ďalšou možnosťou je ošetrenie zadania spôsobu zapojenia tak, aby program „zobral“ len S alebo P.

```
program VYSLEDNY_ODPOR_3;
uses crt;
var R1, R2 : real;
     ZAPOJ : char;
begin
clrscr;
write('Zadaj hodnoty odporov R1 a R2 v ohmoch: '); readln(R1,R2);
write('Sposob zapojenia (S - seriovo, P - paralelne): '); readln(ZAPOJ);
if (ZAPOJ='S') or (ZAPOJ='s')
  then writeln('Vysledny odpor R = ',R1+R2:3:1,' ohmov')
  else if (ZAPOJ='P') or (ZAPOJ='p')
         then writeln('Vysledny odpor R = ',R1*R2/(R1+R2):3:1,' ohmov')
         else writeln('Nebolo stlacene S alebo P!');
readln
end.
```

Úloha: Program VYSLEDNY_ODPOR môžete upraviť aj tak, aby reagoval na vloženie celého slova seriovo alebo paralelne (zrejme premenná ZAPOJ bude typu string[9]).

Príklad VIF-4: Vytvorte program, ktorý po zadaní pH oznámi: prostredie kyslé, neutrálne alebo zásadité.

Analýza: Trochu chémie nezaškodí. Tuším nás učili: ak je pH < 7, tak je prostredie kyslé; ak je pH = 7, tak je neutrálne a ak je pH > 7, tak je prostredie zásadité. Použitie podmienených príkazov je zrejmé.

```
program PH_PROSTREDIA;
uses crt;
var PH:real;
begin
clrscr;
write('Zadaj pH: '); readln(PH);
writeln;
write('Prostredie je ');
if PH<7 then writeln('kysle.');
if PH=7 then writeln('neutralne.');
if PH>7 then writeln('zasadite.');
readln
end.
```

Príklad VIF-5: Vytvorte program analogický programu S-5 (prepočíta čas v sekundách na hodiny : minúty : sekundy s výstupom HH : MM : SS (dvojčísla).

Analýza: Rozdiel od príkladu S-5 je v tom, že ak je číselný údaj (hodiny, minúty alebo sekundy) jednociferný, tak treba „predložiť“ nulu.

```
program CAS_V_SEKUNDACH;
uses crt;
var  H,M,CAS:longint;
begin
clrscr;
write('Zadaj cas v sekundach: '); readln(CAS);
H:=CAS div 3600; CAS:=CAS mod 3600;
M:=CAS div 60;   CAS:=CAS mod 60;
if H<10 then write('0');
write(H,' : ');
if M<10 then write('0');
write(M,' : ');
if CAS<10 then write('0');
```

```
writeln(CAS);
readln
end.
```

Príklad VIF-6: Vytvorte program na určenie, či zadaný rok je priestupný.

Analýza: Platí, že rok je priestupný, ak je deliteľný 4 a nie je deliteľný 100, okrem rokov deliteľných 400. Napríklad rok 1600 je priestupný ale rok 1700 nie. Ešte sa „s priestupným rokom“ stretneme.

```
program PRIESTUPNY_ROK;
uses crt;
var ROK:integer;
begin
clrscr;
write('Zadaj rok: '); readln(ROK);
if (ROK mod 4=0) and ((ROK mod 100<>0) or (ROK mod 400=0))
  then writeln('Rok ',ROK,' - priestupny')
  else writeln('Rok ',ROK,' - nepriestupny');
readln
end.
```

Ešte jedno riešenie (toto riešenie ešte použijeme neskôr):

```
program PRIESTUPNY_ROK_2;
uses crt;
var ROK:integer;
    PRIESTUPNY:boolean;
begin
clrscr;
write('Zadaj rok: '); readln(ROK);
PRIESTUPNY:=(ROK mod 100<>0) and (ROK mod 4=0) or (ROK mod 400=0);
if PRIESTUPNY
  then writeln('Rok ',ROK,' - priestupny')
  else writeln('Rok ',ROK,' - nepriestupny');
readln
end.
```

Upravte program tak, aby oznamoval aktuálne: Rok ... bol / je / bude ... Použite príkaz GetDate z unitu Dos (pozri pomocou Ctrl+F1), ktorého výsledkom je aktuálny rok, mesiac, deň a poradové číslo dňa v týždni, pričom poradové číslo nedele je 0, pondelka 1 atď. Použité premenné sú typu word (pozri aj príklad VCS-1: program POCET_DNI).

Príklad VIF-7: Vytvorte program na výpočet doby splácania - počtu rokov a mesiacov, bezúročnej pôžičky po zadaní požičanej sumy a výšky mesačnej splátky.

Analýza: Zovšeobecnite situáciu, keď máte požičaných 12 000 Sk a mesačne splácate po 100 Sk. Ako dlho budete platiť? Najprv nech nás nezaujímajú „zvyšky“ (rozumej mesiace), len celé roky, ktoré treba splácať. Tie určí delenie bezo zvyšku, t.j. funkcia div. Ak ste našli odpoveď, môžeme ísť ďalej. Ak by ste mali požičaných 13 000 Sk, koľko mesiacov by ste museli ešte po desiatich rokoch platiť? Zovšeobecnite!

Myslite aj na možnosť, že splácať nebude treba ani jeden celý rok, potom položka *počet rokov splácania* by sa vôbec nemala zobraziť, podobne s mesiacmi, ak sa všetko splatí za celý násobok roka.

```
program SPLATKY;
uses crt;
var SUMA, MESACNE, ROKOV, MESIACOV, ZOSTAVASK : longint;
begin
clrscr;
```

```
write('Pozicana suma: '); readln(SUMA);
write('Vyska mesacnej splatky: '); readln(MESACNE);
writeln;
writeln('Treba splacat: ');
ROKOV:=SUMA div (12*MESACNE);
if ROKOV>0
  then writeln('- rokov: ',ROKOV);
ZOSTAVASK:=SUMA-ROKOV*12*MESACNE;
MESIACOV:=ZOSTAVASK div MESACNE;
if MESIACOV>0
  then writeln('- mesiacov: ',MESIACOV);
ZOSTAVASK:=SUMA-(ROKOV*12*MESACNE+MESIACOV*MESACNE);
if ZOSTAVASK>0
  then writeln('- dalsi mesiac este splatit: ',ZOSTAVASK,' korun.');
readln
end.
```

Doplňujúca úloha: Výstup s koncovkami v správnom tvare:

Treba splácať:

1 rok alebo 2, 3, 4 roky alebo 5, 6, ... rokov

1 mesiac alebo 2, 3, 4 mesiace alebo 5, 6, ... mesiacov

... splatiť 1 korunu alebo 2, 3, 4 koruny alebo 5,6, ...korún.

Príklad VIF-8: Na pobavenie a doplnenie vôbec prvého programu v skriptách. Program, ktorý po zadaní mena pozdraví, opýta sa "Ako sa máš?" a primerane zareaguje.

```
program POZDRAV;
uses Crt;
var  MENO, ODPOVED:string;
begin
clrscr;
write('Ako sa volas ? ');
readln(MENO);
{ nasledujuce zmeni prve pismeno v mene na velke - pozri Help pomocou Ctrl+F1 }
MENO:=upcase(MENO[1])+copy(MENO,2,length(MENO));
writeln;
writeln('Ahoj, ',MENO,'!');
writeln;
write('Ako sa mas? ');
readln(ODPOVED);
writeln;
if (ODPOVED='dobre') or (ODPOVED='vyborne') or (ODPOVED='OK')
  then writeln('Som rad, ze sa mas ',ODPOVED,'!')
  else if (ODPOVED='zle') or (ODPOVED='nanic') or (ODPOVED='na...')
         then writeln('Mrzi ma, ze sa mas ',ODPOVED,'!')
         else writeln('"',ODPOVED,'" nepoznam!');
writeln;
gotoxy(25,12);
writeln('A h o j , musim koncit!');
delay(3500);     { delay(CISLO) zastavi beh programu na CISLO milisekund }
clrscr
end.
```

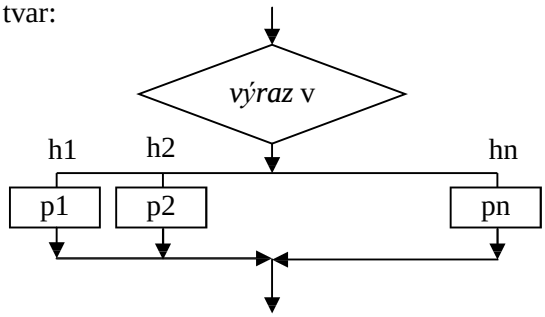
Príklad VIF-9: Už sme mali fyziku, chémiu, bankovníctvo, slušné správanie, tak znova trocha matematiky. Vytvorte program, ktorý zistí, či zadané číslo je deliteľné druhým zadaným číslom.

Analýza: Napríklad číslo 21 je deliteľné 7, lebo zvyšok po delení čísla 21 siedmimi je nula! Ale zvyšok po celočíselnom delení dáva funkcia mod, preto ak...

```
program DELI;
uses crt;
var DELENEC, DELITEL : integer;
begin
clrscr;
write('Zadaj delenca a delitela: ');
readln(DELENEC, DELITEL);
if DELENEC mod DELITEL=0
  then writeln('Cislo ',DELENEC,' je delitelne ',DELITEL)
  else writeln('Cislo ',DELENEC,' nie je delitelne ',DELITEL);
readln
end.
```

N-árne vetvenie, podmienený príkaz case

Má tvar:



h1, h2 až hn sú hodnoty, ktoré môže nadobudnúť výraz v

v slovnom zápise nemá ekvivalent,

realizuje sa zápisom:

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

kde p1 až pn sú príkazy

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

Má tvar:	**case v of**	kde v je tzv. výberový výraz ordinálneho typu,
	h1 : p1;	h1, h2 až hn sú hodnoty rovnakého typu ako výberový výraz
	h2 : p2;	a p, p1, p2 až pn sú príkazy
	...	
	hn : pn	
	[else p]	do hranatých zátvoriek sa umiestňuje nepovinná časť
	end	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.

Príklad:
```
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
end          { do premennej DNI priradí počet dní v mesiaci }
```

Dovolené sú aj zápisy s intervalmi (medzi minimálnou a maximálnou hodnotou sú dve bodky):

```
case ZNAK of
  'A'..'V' , 'a'..'v' : write( succ(ZNAK) );
             'Z' : write( 'A' ) ;
              'z' : write( 'a' )
   else write(ZNAK)
end      { príkaz zakóduje písmeno na nasledujúce  (Z, z na A, a) a zobrazí ho }
         { iné znaky nemení, zobrazí pôvodné }
```

Príklad VCS-1: Program na určenie aktuálneho dňa v týždni pomocou príkazu GetDate (unit Dos), na určenie počtu dní do konca mesiaca a do konca roka.

Analýza: Výsledkom príkazu GetDate (pozri pomocou Ctrl+F1) je aktuálny rok, mesiac, deň a poradové číslo dňa v týždni, pričom poradové číslo nedele je 0, pondelka 1 atď. Využili sme aj tvrdenie, že rok je priestupný (február má 29 dní), ak je deliteľný 4 a pritom nie je deliteľný 100 alebo je deliteľný 400. Rok 2000 by mal byť priestupný, lebo je deliteľný 4, nemal by byť priestupný, lebo je deliteľný 100, ale je priestupný, pretože je deliteľný 400. Zaujímavé. Ak rok nie je deliteľný 4, určite je nepriestupný.

Dali sme si záležať aj na koncovke slova deň, dni, dní (slovenská diakritika sa do TP dá dostať).

Riešenie výpočtu dní do konca roka nie je veľmi elegantné, pokúste sa o krajšie riešenie .

```
program POCET_DNI;
uses crt,dos;
var
  d, m, r, por : word;
  pdm, dkm: byte;    { pdm - pocet dni v mesiaci, dkm - do konca mesiaca }
  dkr:integer;       { dkr - do konca roka }
  priestupny:boolean;{ vysledkom je true, ak je rok priestupny, inak false }
begin
clrscr;
GetDate(r,m,d,por);
write('Dnes je ');
case por of
  0: write('nedela');
  1: write('pondelok');
  2: write('utorok');
  3: write('streda');
  4: write('stvrtok');
  5: write('piatok');
  6: write('sobota');
end;
writeln(' ', d, '.', m, '.', r, '.');
writeln;
priestupny:=(r mod 4=0) and ((r mod 100<>0) or (r mod 400=0));
case m of
  1,3,5,7,8,10,12: pdm:=31;
                2: if priestupny
                     then pdm:=29
                     else pdm:=28;
         4,6,9,11: pdm:=30;
end;
dkm:=pdm-d;
write('Do konca mesiaca ');
case dkm of
      1: writeln('zostáva 1 deň.');
  2,3,4: writeln('zostávajú ', dkm,' dni.');
  else writeln('zostáva ', dkm,' dní.')
  end;
writeln;
case m of
  1: if priestupny
       then dkr:=366-d
       else dkr:=365-d;
  2: if priestupny
       then dkr:=366-31-d
       else dkr:=365-31-d;
  3: dkr:=dkm+30+31+30+31+31+30+31+30+31;
  4: dkr:=dkm+31+30+31+31+30+31+30+31;
  5: dkr:=dkm+30+31+31+30+31+30+31;
  6: dkr:=dkm+31+31+30+31+30+31;
  7: dkr:=dkm+31+30+31+30+31;
```

```
  8: dkr:=dkm+30+31+30+31;
  9: dkr:=dkm+31+30+31;
 10: dkr:=dkm+30+31;
 11: dkr:=dkm+31;
 12: dkr:=dkm;
end;
write('Do konca roka ');
case dkr of
      1: writeln('zostáva 1 deň.');
  2,3,4: writeln('zostávajú ', dkr,' dni.');
  else writeln('zostáva ', dkr,' dní.')
end;
readln
end.
```

Neriešené úlohy na vetvenie:

1. Vytvorte program na riešenie rovnice ax + b = 0, a, b reálne.
2. Vytvorte program na riešenie kvadratickej rovnice $ax^2 + bx + c = 0$, a, b, c reálne, $a \neq 0$.

   Návod: $D = b^2 - 4.a.c$; pre $D \geq 0$ platí $x_{1,2} = (-b \pm \sqrt{D}) / (2.a)$.
3. Vytvorte program na riešenie rovnice $ax^2 + bx + c = 0$, a, b, c reálne.

   Návod: Pozri úlohy č. 1 a 2.
4. Vytvorte program na výpočet neznámej strany pravouhlého trojuholníka po zadaní usporiadanej trojice [a, b, c] (a, b odvesny, c prepona), kde neznáma strana má pri zadaní veľkosť 0.

   Návod: Použi Pythagorovu vetu, napríklad pre trojicu [3,0,5] sa počíta $b = (5^2 - 3^2) = 4$.
5. Vytvorte program na výpočet výšky poštovného pri podaní peňažnej poukážky (tarifná tabuľka je na druhej strane poukážky).
6. Vytvorte program na výpočet povrchu a obvodu zvoleného rovinného útvaru. Ponuka napr.: štvorec, obdĺžnik, trojuholník, lichobežník, kruh a pod. (selekcia príkazom case!).
7. Vytvorte program, ktorý zistí, či zadané tri čísla môžu byť veľkosti strán trojuholníka.

   Návod: Použi trojuholníkovú nerovnosť: súčet dvoch strán v trojuholníku je vždy väčší ako tretia strana.
8. Ukážte, že tri podmienky umožňujú rozlíšiť 8 situácií.
9. Osoby majú hmotnosť x , y a z. Nosnosť výťahu je w. Vytvorte program na zistenie, koľko jázd je treba, aby sa všetci dostali výťahom z prízemia na 13. poschodie.
10. Je daný počet dní v mesiaci a informácia, na ktorý deň v týždni pripadá prvý deň v mesiaci (pondelok - 1, utorok - 2, atď.). Zistite, koľko je v danom mesiaci piatkov.
11. Je daný počet dní v mesiaci a informácia, na ktorý deň v týždni pripadá prvý deň v mesiaci (pondelok - 1, utorok - 2, atď.). Zistite, koľko je v danom mesiaci pracovných dní.
12. Vytvorte program, ktorý po zadaní postupnosti celých čísel oznámi, či je postupnosť rastúca, neklesajúca, klesajúca, nerastúca, konštantná alebo nemá žiadnu z týchto vlastností.

CYKLUS

Použijeme, ak nejaký príkaz alebo skupina príkazov sa má opakovane vykonávať, pokiaľ je splnená alebo nesplnená daná podmienka.

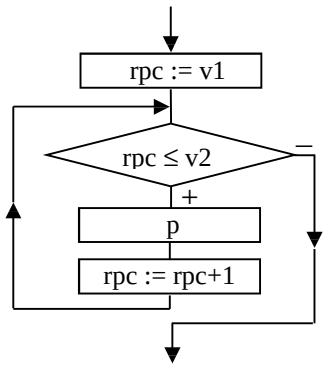
Cykly rozdeľujeme:

1. podľa umiestnenia podmienky na:
    a) cyklus s podmienkou na začiatku
    b) cyklus s podmienkou na konci
    c) úplný cyklus (s podmienkou v strede)
2. podľa toho, či je známy alebo neznámy počet opakovaní v cykle na:
    a) cyklus s explicitne daným počtom opakovaní
    b) cyklus s implicitne daným počtom opakovaní.

Cyklus s explicitne daným počtom opakovaní, príkaz for

Cyklus s explicitne („zvonka“) daným 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

V TP cyklu so známym 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 succ(rpc) (pred(rpc)).

Poznámky:

- Riadiaca premenná cyklu je ordinálneho typu, t.j. môže byť len typu integer, boolean alebo char. Z toho vyplýva, že nemôže nadobúdať hodnoty meniace sa o ľubovoľný krok, napr. 0,5; 0,1 a pod. Programátor s dobrým programátorským štýlom zásadne nemení hodnotu rpc ani o krok napr. 2, 10 a pod., len na nasledovníka resp. predchodcu. Ak potrebujeme zrealizovať cyklus s „nevhodným“ krokom pre príkaz for, použijeme cyklus s príkazom while alebo repeat.

- Začiatočná a koncová hodnota riadiacej premennej cyklu sa získa vyhodnotením výrazov v1 a v2 na začiatku vykonávania príkazu for a počas cyklu ich nemožno meniť.
- Z historických dôvodov (z čias programovacieho jazyka fortran) sa zaužívalo najčastejšie označiť riadiacu premennú cyklu písmenom I, nie je to však pravidlo.

Príklady: for I := 1 to 100 do write(I : 4)

príkaz zobrazí čísla 1 2 3 4 ... 99 100 v piatich riadkoch po 20 čísel (80 znakov v riadku delené 4 znaky na číslo = 20 čísel v riadku)

for I := 100 downto 1 do write (I : 4)

príkaz zobrazí čísla 100 99 98 ... 2 1, t.j. opačne ako v predchádzajúcom príkaze for

SUCET := 0;

for CISLO := 1 to POCET do SUCET := SUCET + CISLO

príkaz sčíta čísla 0 + 1 + 2 + 3 + ... + POCET, 0 je počiatočná hodnota premennej SUCET

MOCNINA := 1;

for I := 1 to N do MOCNINA := MOCNINA * X

príkaz vykoná 1.X.X. ... .X = X^N, 1 je počiatočná hodnota premennej MOCNINA

for Z := 'A' to 'Z' do writeln(Z : 5, ord(Z) : 3)

príkaz zobrazí v riadkoch znaky A až Z a ich poradové čísla z ASCII tabuľky

for J := -10 to 10 do p

príkaz for vykoná príkaz p 21-krát (J bude postupne -10, -9, ... , 0, ... , 10)

for K := 0 to (downto) 0 do p

príkaz for vykoná príkaz p jedenkrát

for NIC := 100 to –100 do p

príkaz for nevykoná príkaz p ani raz (začiatočná hodnota rpc nie je ani raz menšia alebo rovná koncovej hodnote)

for NIC := 0 downto 10 do p

príkaz for nevykoná príkaz p ani raz (začiatočná hodnota rpc nie je ani raz väčšia alebo rovná koncovej hodnote)

for NIC := 'z' to 'A' do p

príkaz for nevykoná príkaz p ani raz, pretože 'z' > 'A' (znak z má v ASCII tabuľke väčšie poradové číslo)

for I := - trunc(sqrt(N)) to trunc(sqrt(N)) do p

Výrazy v1 a v2 nemusia mať pri zápise len konkrétne hodnoty, môžu to byť aj výrazy.

Príklad CFR-1: Vytvorte program na zobrazenie hodnôt riadiacej premennej cyklu zo zadaného intervalu pre typ integer aj char. Uvedomte si, že ak variant „to“ zobrazí viac ako jednu hodnotu, variant „downto“ nemôže zobraziť ani jednu hodnotu a naopak.

```
program HODNOTY_RPC;
uses crt;
var  ZAC, KON, I : integer;
     ZACZ, KONZ, Z : char;
begin
```

```
clrscr;
write('Zadaj zaciatocnu a koncovu hodnotu typu integer: ');
readln(ZAC,KON);
writeln('Hodnoty rpc pre "to":');
for I:=ZAC to KON do
  write(I:8);
writeln;
writeln('Hodnoty rpc pre "downto":');
for I:=ZAC downto KON do
  write(I:8);
writeln; writeln;
write('Zadaj zaciatocnu hodnotu typu char: ');
readln(ZACZ);
write('Zadaj  koncovu  hodnotu  typu char: ');
readln(KONZ);
writeln('Hodnoty rpc pre "to":');
for Z:=ZACZ to KONZ do
  write(Z:4);
writeln;
writeln('Hodnoty rpc pre "downto":');
for Z:=ZACZ downto KONZ do
  write(Z:4);
writeln;
readln
end.
```

Neuviedli sme analýzu, radšej upozorníme na „problém“ s načítaním viacerých hodnôt typu char v jednom príkaze readln. Keby sme, tak ako pri type integer, umiestnili premenné ZACZ a KONZ do jedného príkazu readln a hodnoty vložili oddelené medzerou, premennej KONZ by vždy priradilo medzeru (druhý znak je predsa medzera)! Preto nečíselné hodnoty načítavame z klávesnice väčšinou v samostatných príkazoch readln. Ak chceme načítať obe hodnoty v jednom príkaze readln, musíme ich vložiť za sebou bez oddelenia.

Príklad CFR-2: Vytvorte program na výpočet súčtu všetkých prirodzených čísel od 1 po zadané N pomocou cyklu.

Analýza:

1. Akú algoritmickú konštrukciu treba použiť?

   Napríklad pre N = 100 treba sčítať 1 + 2 + 3 + ... + 99 + 100 = 5050. Vidíme, že sa opakuje sčítavanie. Keďže opakovane treba vykonávať nejaký príkaz, použijeme cyklus. Poznáme aj počet opakovaní, N-krát (aj 1 treba pričítať k pôvodnej hodnote v pamäti), preto to je cyklus so známym počtom opakovaní a ten sa realizuje príkazom for: for CISLO := 1 to N do

2. Aký príkaz sa má v cykle opakovať?

   Vyzerá to tak, že vždy k čiastočnému súčtu, t.j. k súčtu, kým nie je všetko sčítané, treba pripočítať vhodné číslo. Vhodné číslo je pri prvom prechode cyklom 1, pri druhom 2, pri treťom 3 atď., až pri n-tom N. Ak sme pozorní, uvedomíme si, že práve tieto hodnoty nadobúda riadiaca premenná cyklu for. Preto: nový čiastočný súčet = predchádzajúci čiastočný súčet + CISLO

   Ak dobre poznáme vykonanie príkazu priradenia, uvedomíme si, že ten presne pracuje s „novým a predchádzajúcim“, ak na jeho ľavej aj pravej strane je tá istá premenná (to isté pamäťové miesto)! Preto v cykle treba opakovať príkaz: SUCET := SUCET + CISLO

   V príkaze takéhoto typu musíte neustále vidieť: nový SUCET = starý SUCET + ...

   alebo všeobecne: nová HODNOTA = stará HODNOTA +,*,-,/...

3. Teraz je už všetko v poriadku?

   Ešte nie. Ak začneme simulovať výpočet v cykle, zistíme, že nepoznáme hodnotu premennej SUCET na pravej strane pri prvom prechode cyklom. Nepoznáme počiatočný súčet resp. nenastavili sme ešte počiatočnú hodnotu premennej SUCET. Správne vykonanie príkazu priradenia, v ktorom je rovnaká premenná na ľavej aj pravej strane, si vyžaduje pred príkazom cyklu nastavenie počiatočnej hodnoty takejto premennej. Pre sčítanie to býva (ale vo všeobecnosti nemusí byť) 0, pre násobenie 1. Preto pred príkaz for vložíme ešte príkaz: SUCET := 0;

   Už počujeme hlasy, že váš program počíta správne aj bez tohto priradenia (spomínali sme, že TP 7.0 vloží do všetkých číselných premenných počiatočnú hodnotu 0). Čo však, ak program nespustíte v TP 7.0 alebo ho vložíte do cyklu, aby vám viackrát umožnil zopakovať výpočet? Potom určite nebude počítať správne (počiatočnou hodnotou bude náhodné číslo alebo hodnota predchádzajúceho súčtu).

   Teraz je už všetko v poriadku a tu je program:

```
program SucetPrvychN;
uses crt;
var  N, CISLO, SUCET : integer; { pripadne typ word alebo longint }
begin
clrscr;
write('Scitat prirodzene cisla po ');
readln(N);
SUCET:=0;
for CISLO:=1 to N do
  SUCET:=SUCET+CISLO;
writeln('Sucet prvych ',N,' cisel je ',SUCET);
readln
end.
```

Poznámka: Na výpočet súčtu prvých n prirodzených čísel možno použiť aj vzorec SUCET = $\frac{N}{2}$.(1+N), preto sme v zadaní úlohy uviedli, že chceme výpočet pomocou cyklu. Zároveň si uvedomte, že výrazne sa efektivitou výpočtu líšiacich algoritmov môže byť (a väčšinou aj býva) viacej!

Program upravte tak, aby:

a) počítal súčet prvých N prirodzených čísel od N po 1,
b) počítal súčet všetkých prirodzených čísel od zadaného A po zadané B vrátane,
c) počítal súčet všetkých celých čísel od zadaného A po zadané B vrátane.

Príklad CFR-3: Vytvorte program na výpočet mocniny x na n-tú, x reálne, n prirodzene číslo.

Analýza: Začneme „sedliackym“ rozumom. Ak máme vypočítať napríklad 5^3, násobíme 5.5.5 = 125. Ďalej by sme mohli vedieť, že 0^0 nie je definované (asi preto, že 0 na „čokoľvek“ je 0 a „čokoľvek“ na 0-tú je 1, preto 0^0 by malo byť 0 a zároveň 1 a to nejde).

1. Opakovane sa vykonáva násobenie, preto treba použiť cyklus; násobenie treba vykonať n-krát, preto cyklus so známym počtom opakovaní, teda príkaz for.
2. Vždy nový súčin sa rovná predchádzajúci súčin krát x, preto sa v cykle bude opakovať príkaz SUCIN := SUCIN * X.
3. Nesmieme zabudnúť na počiatočnú hodnotu premennej SUCIN, ktorá môže byť 1. Keby sme zvolili nulu?

4. Je tu však ešte jedno obmedzenie, ak zadáme N aj X nula, tak má program oznámiť: „0 na 0-tú nedefinovane!“, inak má „normálne“ počítať – teda vetvenie.

```
program MOCNINA;
uses crt;
var X, MOCNINA : real;
     N, I, PDM : integer;
begin
clrscr;
write('Zadaj zaklad mocniny a exponent: ');
readln(X,N);
write('Zadaj pocet desatinnych miest vystupu: ');
readln(PDM);
if (X=0) and (N=0)
  then writeln('0 ma 0-tu nedefinovane!')
  else begin
       MOCNINA:=1;
       for I:=1 to N do
         MOCNINA:=MOCNINA*X;
       writeln(X:PDM+2:PDM,' na ',N,' = ',MOCNINA:PDM+2:PDM)
       end;
readln
end.
```

Príklad CFR-4: Vytvorte program na výpočet aritmetického priemeru n reálnych čísel.

Analýza: Snáď všetci vieme, že aritmetický priemer sa vypočíta ako súčet daných n čísel delený počtom všetkých čísel, t.j. n. Program si má vypýtať n reálnych čísel a sčítať ich, preto cyklus s „príkazmi“: ***zadaj číslo*** a ***k čiastočnému súčtu pripočítaj nové číslo***. Toto sa má zopakovať pre n čísel, teda n-krát. Takže cyklus so známym počtom opakovaní – príkaz for.

V programe sme namiesto premennej n použili premennú POCET a ošetrili sme aj prípad, keď bude zadaný počet čísel 0 - program by bez ošetrenia skončil chybovým hlásením: Delenie nulou!

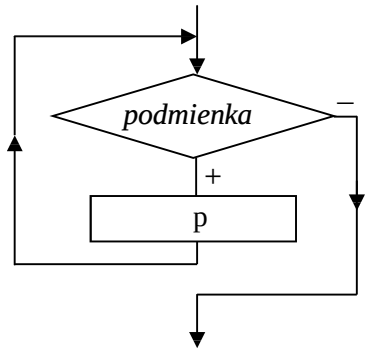
```
program PRIEMER;
uses crt;
var  I, POCET : integer;
     X, SUCET : real;
begin
clrscr;
write('Zadaj pocet cisel: ');
readln(POCET);
SUCET:=0;
for I:=1 to POCET do
  begin
  write(I:2,'. cislo: ');
  readln(X);
  SUCET:=SUCET+X
  end;
if POCET>0
  then writeln('Priemer zaokruhleny na stotiny: ',SUCET/POCET:4:2);
readln
end.
```

Ďalšie úlohy aj na príkaz for sú uvedené v kapitole Ďalšie zaujímavé úlohy.

Cyklus s podmienkou na začiatku, príkaz while

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

Má tvar:



v slovnom zápise:

pokiaľ *podmienka* opakuj
p

kde p je príkaz

V TP 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.

Príklady:

```
I := 1;
while I <= 10 do
   I := I + 1;                    príkaz v cykle sa vykoná 10-krát

SUCET := 0;
while N > 0 do
   begin
   SUCET := SUCET + N;
   N := N - 1
   end                            sčíta všetky prirodzené čísla od N po 1

DELITEL := 2;
while CISLO mod DELITEL <> 0  do
    inc ( DELITEL )               nájde deliteľa zadaného čísla ( CISLO > 1 )
```

Príklad CWH-1: Príkaz for je viac-menej špeciálnym prípadom príkazu while. Demonštrujeme to v nasledujúcom programe.

```
program AKO_FOR;
uses crt;
var  N, I : integer;
begin
clrscr;
write('Zadaj pocet opakovani: ');
readln(N);
writeln('Ako prikaz for s "to" od 1 po ',N);
I:=1;
while I<=N do
  begin
  writeln(I:6,'. opakovanie');
  I:=I+1
  end;
readln;
```

```
writeln('Ako prikaz for s "downto" od ',N,' po 1');
while N>0 do
  begin
  writeln(N:6,'. opakovanie');
  N:=N-1
  end;
readln
end.
```

Príklad CWH-2: Vytvorte program na výpočet ciferného súčtu zadaného prirodzeného čísla.

Analýza: Ciferný súčet napríklad čísla 123 je 6, lebo 1 + 2 + 3 = 6. K prvej cifre (3) sa dostaneme predelením čísla 123 desiatimi: 123 : 10 = 12, zvyšok 3. Pokúsme sa vystačiť s celými číslami. Ak spravíme 123 mod 10, dostaneme rovno číslo 3. Ak spravíme 123 div 10 dostaneme 12, čo sa nám tiež hodí, lebo ďalej potrebujeme to isté (mod a div) zopakovať s 12-kou (12 mod 10 = 2 a 12 div 10 = 1) a nakoniec s 1-kou (1 mod 10 = 1 a 1 div 10 = 0). Tým sme prešli všetky cifry čísla. Opakovane spracúvam výsledky po funkciách mod a div s deliteľom 10, preto treba použiť cyklus. Zadané číslo môže byť jedno ale aj troj (to naše) prípadne desaťciferné a na začiatku cyklu nevieme, kedy spracujeme poslednú cifru. Preto cyklus s neznámym počtom opakovaní príkazov v cykle – príkaz while. Podmienkou ukončenia cyklu je, aby sme nemali už čo „spracovať“, t.j. aby aktuálnym číslom bola nula (CISLO div 10 = 0).

```
program CIFERNY_SUCET;
uses crt;
var  CISLO    : longint;
     CIFSUCET : byte;
begin
clrscr;
write('Zadaj prirodzene cislo: ');
readln(CISLO);
write('Sucet cifier cisla ',CISLO,' je ');
CIFSUCET:=0;
while CISLO>0 do
  begin
  CIFSUCET:=CIFSUCET+CISLO mod 10;
  CISLO:=CISLO div 10
  end;
writeln(CIFSUCET);
readln
end.
```

Prečo sme výstup museli posunúť hore? Ak sa vám funkcie div a mod nepáčia, tu je riešenie bez nich:

```
program CIFERNY_SUCET_2;
uses crt;
var  CISLO    : longint;
     CIFSUCET : byte;
begin
clrscr;
write('Zadaj prirodzene cislo: ');
readln(CISLO);
write('Sucet cifier cisla ',CISLO,' je ');
CIFSUCET:=0;
while CISLO>0 do
  begin
  CIFSUCET:=CIFSUCET+trunc(frac(CISLO/10)*10);
  CISLO:=trunc(CISLO/10)
  end;
writeln(CIFSUCET);
readln
end.
```

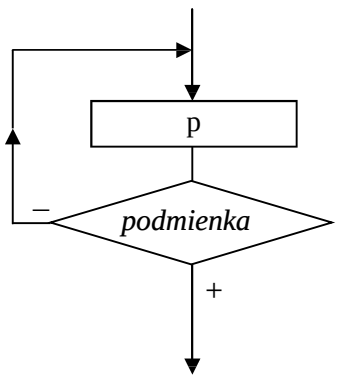
Príklad CWH-3: Vytvorte program na zobrazenie prevodnej tabuľky medzi slov. korunami a inou menou. Kurz je zadaný v konštante. Tabuľka bude zobrazovať hodnoty od 0,- Sk po zadanú sumu (premenná KoncovaSuma) so zadaným prírastkom (premenná Krok). Napríklad pre koncovú sumu 100 s prírastkom 10 má vypočítať a zobraziť hodnoty pre 0, 10, 20, 30, 40, ... , 100 Sk. Výpočet hodnoty v cudzej mene je jednoduchý: suma v Sk krát kurz. Keďže sa opakovane počíta hodnota v cudzej mene, treba použiť cyklus. Riadiaca premenná cyklu sa však môže zväčšovať o teoreticky ľubovoľné kladné reálne číslo, preto použijeme príkaz while.

Program sme doplnili o jednoduchú hlavičku tabuľky a hodnoty sú zobrazované s presnosťou na desatiny, na miestach stotín sa zobrazujú nuly. Doplňte tabuľku o zvislé „čiary".

```
program TABULKA;
uses  crt;
const KURZ=1.24;
var   KoncovaSuma, Krok, RPC : real;
begin
clrscr;
write('Zadaj koncovu sumu a prirastok: ');
readln(KoncovaSuma, Krok);
clrscr;
writeln('----------------------':48);
writeln('      SK         MENA ':48);
writeln('----------------------':48);
RPC:=0;
while RPC<=KoncovaSuma do
  begin
  writeln(RPC:33:1,'0',KURZ*RPC:12:1,'0');
  RPC:=RPC+KROK
  end;
writeln('----------------------':48);
readln
end.
```

Cyklus s podmienkou 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 TP 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.

Príklady:

```
I := 0;
repeat
  I := I + 1
until I = 10
```
príkaz v cykle sa vykoná 10-krát

```
repeat
  write('Zadaj číslo: ');
  readln(X)
until X = 0
```
príkazy v cykle sa budú opakovať, kým nevložíme 0

```
repeat
  p1; ...
  pn
until false
```
nekonečný cyklus ukončí sa Ctrl+Break

Príklad CRP-1: Vytvorte programovú schému na zabezpečenie opakovania behu programu ľubovoľný počet krát.

Analýza: Už vás nudí neustále spúšťať ten istý program, najmä keď ho testujete, či pracuje správne? Riešením je príkaz repeat. Chceme dosiahnuť, aby sa po spustení program vykonal a na konci sa opýtal: „Opakovať?“. Ak odpovieme N = nie, program skončí, inak sa vykoná znova.

Máme viacej možností, najprv tá najjednoduchšia:

```
...
begin
repeat
  { príkazy programu }
  write('Opakovať (nie = N) ?');
  readln(ODPOVED)
until (ODPOVED = 'N') or (ODPOVED = 'n')
end.
```

Elegantnejším riešením je použiť funkciu unitu crt readkey. Výsledkom funkcie readkey je stlačený kláves, ušetríme premennú ODPOVED aj stlačenie klávesu Enter. Readkey v kombinácii s upcase:

```
begin
repeat
  { príkazy programu }
  writeln('Opakovať (nie = N) ?')
until  upcase(readkey) = 'N'
end.
```

Na ukončenie programov sa používa aj kláves Esc. Pomocou programu na nasledujúcej strane zistíme, že jeho poradové číslo je 27. A tu je schéma:

```
begin
repeat
  { príkazy programu }
  writeln('Koniec – stlač Esc!':45)
until  readkey = chr(27)
end.
```

```
const Esc = #27;
...
begin
{ príkazy programu }
writeln('Koniec – stlač Esc!':45)
until  readkey = Esc
end.
```

Zápis chr(X) je rovnocenný so zápisom #X.

Aj na zistenie poradového čísla stlačeného znaku môže poslúžiť malý program s cyklom repeat:

```
program KOD_STLACENEHO_KLAVESU;
uses crt;
var  stlaceny_klaves:char;
begin
clrscr;
writeln('Koniec – stlac C':45);
repeat
  stlaceny_klaves:=readkey;
  writeln(ord(stlaceny_klaves));
until upcase(stlaceny_klaves)='C';
end.
```

Príklad CRP-2: Vytvorte programovú schému na ošetrenie správnosti vstupných hodnôt. Ak vstupné hodnoty nespĺňajú vstupné podmienky, majú sa vyžiadať nové vstupné hodnoty.

Analýza: Príkaz repeat je vhodný aj na ošetrenie vstupných hodnôt, či spĺňajú vstupné podmienky. Najprv predsa musíme zadať vstupné hodnoty a až potom môže podmienka rozhodnúť, či treba vstup opakovať alebo môže program pokračovať. Ako ukážku ošetríme výpočet mocniny x^n pre x = 0 a n = 0 (pozri príklad CFR-3):

```
repeat
  write('Zadaj zaklad mocniny a exponent: ');
  readln(X,N)
until (X<>0) or (N<>0)
```

Správnejšie je nie len „zacykliť“ vstup ale aj doplniť výpisom, prečo sa príkazy vstupu opakujú. Teda:

```
repeat
  write('Zadaj zaklad mocniny a exponent: ');
  readln(X,N);
  if (X=0) and (N=0) then writeln('0 na 0-tú nedefinovanú!')
until (X<>0) or (N<>0);
```

Pod chybou vstupu môžeme rozumieť aj „preklep“, napríklad vložením písmena namiesto číslice (Error 106: Invalid numeric format.). K vyriešeniu tohto problému treba poznať funkciu IOResult, ktorá po každej vstupno/výstupnej operácii nadobúda buď hodnotu 0, ak nedošlo ku chybe, alebo číslo chyby. Keďže sme zapli v Options – Compiler... prepínač I/O Checking – kontrolu všetkých vstupno/výstupných operácií, program by hneď skončil s chybovým hlásením. Túto kontrolu však možno vypnúť umiestnením direktívy {$I - } pred príkaz vstupu a následne ju zapnúť príkazom {$I +}. Schéma má tvar:

```
...
var CHYBA : boolean;
            X : integer;   { X akýkoľvek číselný typ }
...
begin
repeat
  write('Vlož číslo: ');
  {$I-} readln(X); {$I+}
  CHYBA := IOResult <> 0;
  if CHYBA then writeln('Chyba vstupu!')
until not CHYBA;
napr. writeln(X)
```

Ak dôjde ku chybe, IOResult bude rôzny od nuly a preto premenná CHYBA nadobudne hodnotu TRUE. Na obrazovke sa zobrazí: Chyba vstupu! a znova Vlož číslo:, pretože CHYBA má hodnotu true a negácia (not) true je false – vtedy sa príkazy v repeat opakujú. Otestujte program pre dobré aj zlé vstupné hodnoty

A teraz už „normálny“ príklad:

Príklad CRP-3: Vytvorte program – hru na uhádnutie zadaného čísla, keď po vložení čísla program oznámi „Veľa, uber!“, „Málo, pridaj!“ alebo „Výborne, uhádol si!“. Nech treba uhádnuť celé číslo z intervalu od 0 po 100.

Analýza: Pre získanie čísla, ktoré bude treba uhádnuť sú dve možnosti:

- buď máme protihráča a ten zadá číslo tak, aby to súper nevidel (napríklad čiernym písmom – nastavuje sa príkazom textcolor na čiernom podklade – nastavuje sa príkazom textbackground), alebo
- číslo pomocou funkcie random vyberie počítač (random(x) - výsledkom je náhodné celé číslo z intervalu <0,x); pred príkazom random musí byť použitý na začiatku programu príkaz randomize).

Druhá časť programu je vlastné hádanie. „Sedliacky“ rozum hovorí: vložíme číslo, počítač ho porovná s číslom, ktoré treba uhádnuť a oznámi „Veľa, uber!“ alebo „Málo, pridaj!“ a znova sa celé opakuje, až kým neuhádneme, vtedy hádanie končí. Keďže nevieme, na koľký pokus sa nám podarí uhádnuť hľadané číslo, je to cyklus s neznámym počtom opakovaní. Najprv musíme vložiť číslo a porovnať a až potom vieme rozhodnúť, či cyklus končí – preto cyklus s podmienkou na konci. A tu je hotový program:

```
program HADAJ;
uses  crt;
const MAX=100;
var   CISLO,UHCISLO:integer;
      ODP:char;
begin
randomize;
repeat
  clrscr;
  gotoxy(1,12); writeln('Cislo vyberie pocitac ....... P':50);
  writeln;      writeln('Cislo zada protihrac ........ H':50);
  ODP:=upcase(readkey);
  clrscr;
  case ODP of
    'P': UHCISLO:=random(MAX+1);
    'H': begin
         gotoxy(15,12);
         write('Zadaj cislo na uhadnutie < ',MAX,': ');
         textcolor(black);
         readln(UHCISLO);
         textcolor(white);
         clrscr;
         end;
  end;
  writeln('H A D A J  C I S L O !');
  repeat
    readln(CISLO);
    if CISLO <> UHCISLO
      then if CISLO<UHCISLO then writeln('Malo, pridaj!')
                            else writeln('Vela, uber!');
  until CISLO=UHCISLO;
  clrscr; textcolor(yellow+blink);
  gotoxy(30,12); write('B  I  N  G  O  ! ! ! ');
  delay(2500);
  normvideo;
  gotoxy(30,24); write('Este raz? (nie = N) ');
until upcase(readkey)='N';
end.
```

Príkaz normvideo by mal nastaviť štandardné atribúty pre farby textu aj pozadia. Program môžete doplniť aj o zvukové efekty pomocou príkazov sound(*frekvencia*) a nosound – vypína zvuk, tento nesmiete zabudnúť uviesť po príkaze sound. Dĺžka trvania tónu sa realizuje príkazom delay(*milisekúnd*).

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 ASCII), 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 situácii, 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.

Sem-tam sa u začiatočníkov stáva, v snahe nezabudnúť bodkočiarku, že ju dajú aj za slovo „do“ v príkaze for alebo while. Tým vznikol tzv. **prázdny príkaz** a cyklus zrejme pracuje celkom ináč, ako to bolo pôvodne plánované.

Organizácia programu

Poslednou teoretickou časťou je organizácia programu v Turbo Pascale.

Program sa skladá:

program *meno programu;*	hlavička programu (môže chýbať)		
uses *unity oddelené čiarkami***;**	deklarácia unitov		
const *meno konštanty* **=** *hodnota***;**	definícia konštánt		
type *meno typu* = ...;	definícia typov	úsek definícií	
var *premenné oddelené čiarkami* **:** *typ premenných*;	deklarácia premenných	a deklarácií	
procedure ...	deklarácia		
function ...	procedúr a funkcií		blok
begin			
príkaz1;	príkazová časť		
príkaz2;	začína slovom begin		
...	a končí slovom end		
príkazn			
end.	bodka – koniec programu (nepatrí do bloku)		

Poznámky:

- Niektoré objekty z predchádzajúcej schémy ešte nepoznáme.
- Príkazy sa v príkazovej časti oddeľujú bodkočiarkou.
- Všetky mená objektov sú identifikátory.

EXE verzie programov

Turbo Pascal umožňuje vytvárať aj od prostredia TP nezávislé verzie programov, priamo programy s príponou exe. Keď chceme získať exe verziu programu, musíme jeho kompiláciu (preklad) presmerovať z pamäte na disk. Robí sa to prepnutím položky Destination v Compile z hodnoty Memory na hodnotu Disk. Program s rovnakým názvom ako bol pôvodný súbor, len s príponou exe, sa uloží do aktuálneho adresára.

Ďalšie riešené úlohy

Príklad CWH-4: Vytvorte program na krátenie zlomkov.

Analýza: Napr. v zlomku 15/12 sa dá čitateľ aj menovateľ deliť číslom 3 a zlomok sa dá upraviť do základného tvaru 5/4. Číslo 3, ktorým sme krátili zlomok, je najväčší spoločný deliteľ (nsd) čitateľa a menovateľa zlomku. Nsd(15,12) sa vypočíta takto: nsd(15,12) = nsd(15-12,12) = nsd(3,12) = nsd(3,12-3) = nsd(3,9) = nsd(3,9-6) = nsd(3,3) = 3. Opakujeme odčítanie menšieho čísla od väčšieho dovtedy, kým sa nerovnajú. Rovnaké číslo je najväčší spoločný deliteľ pôvodných čísel.

V slovnom zápise:
```
pokiaľ je A<>B opakuj
    ak A>B
        tak A := A-B
        inak B := B-A;
NSD := A  { alebo B, keďže sú rovnaké }
```

Výstup programu nech je v tvare: 15 / 12 = 5 / 4, ak je podiel celé číslo, zlomková čiara a menovateľ 1 nech sa nezobrazia, napr. 12 / 6 = 2.

```
program ZLOMOK;
uses crt;
var CIT,MEN,A,B:integer;
begin
clrscr;
repeat
  write('Zadaj citatela a menovatela zlomku: ' );
  readln(CIT,MEN);
  write(CIT,' / ',MEN,' = ');
  A:=CIT;
  B:=MEN;
  while A<>B do
    if A>B
      then A:=A-B
      else B:=B-A;
  CIT:=CIT div A;
  MEN:=MEN div B;
  if MEN=1
    then writeln(CIT)
    else writeln(CIT,' / ',MEN);
  writeln('Koniec - stlac Esc!')
until readkey=chr(27)
end.
```

Príklad CRP-4: Vytvorte program na výpočet aritmetického priemeru vopred neznámeho počtu kladných reálnych čísel.

Analýza: Úloha je podobná úlohe CFR-4, v nej však bol známy počet čísel a preto sme mohli použiť príkaz for. Pri neznámom počte čísel musíme použiť cyklus s neznámym počtom opakovaní. Ukončenie cyklu je v takomto prípade na tzv. koncovú hodnotu. V našom príklade sú všetky zadávané reálne čísla kladné a preto môžeme zvoliť za koncovú hodnotu nulu. Znamená to, že príkazy v cykle sa budú opakovať, pokiaľ nevložíme 0, vtedy sa cyklus ukončí. Najprv vložíme číslo a potom môžeme rozhodnúť, či cyklus končí, teda podmienka je na konci, preto príkaz repeat. Ošetrite aj prípad, keď hneď prvým vloženým číslom bude 0.

```
program PRIEMER2;
uses crt;
var     POCET : integer;
     X, SUCET : real;
begin
clrscr;
POCET:=0;
SUCET:=0;
repeat
  write('Vloz cislo: ');
  readln(X);
  if X>0
    then begin
         inc(POCET);
         SUCET:=SUCET+X
         end
until X=0;
if POCET>0
  then writeln('Priemer zaokruhleny na stotiny: ',SUCET/POCET:4:2)
  else writeln('Bola zadana len koncova hodnota!');
readln
end.
```

Program CFR-5: Program na prevod malých písmen textu na veľké.

Analýza: Prakticky si treba len osvojiť zápis napr. TEXT[1] – prvý znak reťazca v premennej TEXT, TEXT[5] – piaty znak reťazca, TEXT[I] – I-ty znak reťazca v premennej TEXT. Ak teda chceme zmeniť všetky písmená v reťazci na veľké, musíme ísť od prvého znaku až po posledný. Počet znakov v reťazci nám zistí funkcia length (vracia číslo – dĺžku reťazca). Funkcia upcase zmení malé písmeno na veľké, ostatné znaky nemení.

```
program PREVOD;
uses crt;
var TEXT : string;
    DLZKA, I : integer;
begin
clrscr;
write('Text: ');
readln(TEXT);
DLZKA:=length(TEXT);
for I:=1 to DLZKA do
  TEXT[I]:=upcase(TEXT[I]);
writeln(TEXT);
readln
end.
```

Program CFR-6: Vytvorte program na kódovanie textu posunutím o jeden znak vpravo (Zz -> Aa) a dekódovanie textu posunutím o jeden znak vľavo (Aa -> Zz).

Analýza: Podstatou algoritmu je brať znak po znaku text a pokiaľ je znakom písmeno, zobraziť jeho nasledovníka – pri kódovaní, resp. predchodcu – pri dekódovaní. Situáciu trochu komplikujú okrajové písmená, pretože veľké Z a malé z sa majú posunúť na veľké A a malé a resp. Aa na Zz pri dekódovaní. Keďže je viacej možností, a na každú sa má ináč kódovať, vhodné je použiť príkaz case s časťou else (každý iný znak ako písmeno sa nemá zmeniť). Prejsť treba celý text, od prvého po posledný znak, čo nám zaručí príkaz for (v2 = length(TEXT)). Program musí umožňovať vybrať si z dvoch možností: Kódovať alebo Dekódovať.

```
program KODOVANIE_a_DEKODOVANIE_posunutim_o_znak;
uses crt;
var TEXT:string;
    I:byte;
    ODPOVED:char;
begin
clrscr;
write('Text: ');
readln(TEXT);
writeln('[K]odovat alebo [D]ekodovat?');
ODPOVED:=upcase(readkey);
if ODPOVED='K'
  then begin
       for I:=1 to length(TEXT) do
         case TEXT[i] of
           'A'..'V','a'..'v': write(succ(TEXT[I]));
                        'Z': write('A');
                        'z': write('a');
           else write(TEXT[I])
         end;
       writeln
       end
   else if ODPOVED='D'
          then begin
               for I:=1 to length(TEXT) do
                 case TEXT[i] of
                   'B'..'Z','b'..'z': write(pred(TEXT[I]));
                                'A': write('Z');
                                'a': write('z');
                   else write(TEXT[I])
                 end;
               writeln
               end
          else writeln('Nebolo stlacene K alebo D!');
readln
end.
```

Príklad CFR-7*: Príklad je určený pre zdatnejších alebo húževnatejších programátorov. Je principiálne odlišný od predchádzajúceho. Program má kódovať a dekódovať písmená textu posunutím o POSUN písmen vpravo (vľavo).

Analýza: Programy s hviezdičkou (nie sú zakázané) sú určené na prácu s hotovým programom. Ujasnite si jednotlivé časti programu a vyskúšajte, ako pracujú. Program si pamätá predchádzajúci text (pri prvom spustení samozrejme nie), aby ste ho mohli po zakódovaní hneď dekódovať a opačne. Dekódovanie je v skutočnosti kódovanie o 26-POSUN znakov (neustále vpravo). Číslo 26 je počet písmen (A až Z alebo malé a až z), pričom poradové číslo A je 65 a malého a 97. Pri kódovaní sa pohybujeme v uzavretom kruhu.

```
program KODOVANIE_a_DEKODOVANIE_posunutim_o_POSUN_znakov;
uses crt;
var TEXT:string;
    POSUN:integer;
    DLZKA,i,ZacAbc,PozVAbc,RelPozKodu:byte;
    ODPOVED:char;
begin
clrscr;
repeat
  writeln('Novy text (ano - A)?');
  if upcase(readkey)='A'
    then begin
         writeln('Text:');
         readln(TEXT);
         DLZKA:=length(TEXT);
         end;
  repeat
    write('[K]odovat alebo [D]ekodovat? ');
    readln(ODPOVED);
    ODPOVED:=upcase(ODPOVED);
  until (ODPOVED='K') or (ODPOVED='D');
  write('Posun o znakov: ');
  readln(POSUN);
  POSUN:=POSUN mod 26;
  if POSUN<0 then POSUN:=POSUN+26;
  if ODPOVED='D' then POSUN:=26-POSUN;
  for i:=1 to DLZKA do
    if TEXT[i] in ['A'..'Z','a'..'z']
      then begin
           case TEXT[i] of
             'A'..'Z':ZacAbc:=65;
             'a'..'z':ZacAbc:=97;
           end;
           PozVAbc:=ord(TEXT[i])-ZacAbc;
           RelPozKodu:=(PozVAbc+POSUN) mod 26;
           TEXT[i]:=chr(ZacAbc+RelPozKodu)
           end;
  writeln(TEXT);
  writeln('KONIEC - stlac Esc':45)
until readkey=#27
end.
```

Ďalšou skupinou úloh sú „grafické“ úlohy v textovom režime obrazovky.

Príklad CFR-8: Program ROZSVECOVANIE_V_OKNACH náhodným generovaním polohy postupne vypĺňa plochu obrazovky „rozsvietenými“ obdĺžnikmi (znak s poradovým číslo 219) .

Analýza: Treba opakovane „rozsvecovať okná“, preto cyklus. My sme ho postavili na známom počte opakovaní a preto sa program opýta, koľko okien má rozsvietiť (to je počet opakovaní príkazov v cykle). No a potom už stačí len použiť funkciu random a nechať náhodne vygenerovať číslo od 1 do 80 – stĺpec a číslo od 1 do 24 – riadok, v ktorom sa „rozsvieti“. Požadovaný stĺpec a riadok sa nastaví príkazom gotoxy.

```
program ROZSVECOVANIE_V_OKNACH;
uses crt;
var  I, POCET : word;
begin
randomize;
clrscr;
write('Pocet rozsvietenych okien do 65 tis.: ');
readln(POCET);
for I:=1 to POCET do
  begin
```

```
  gotoxy(random(80)+1,random(24)+1);
  write(chr(219));
  end;
readln
end.
```

Predchádzajúca úloha je ešte zaujímavejšia, ak chceme, aby program generoval „rozsvecovanie“, až kým nestlačíme ľubovoľný kláves. Funkcia keypressed unitu crt má hodnotu true, ak bol stlačený kláves, inak má hodnotu false. Celý problém je vo vytvorení tentoraz cyklu s neznámym počtom opakovaní, ktorý skončí na stlačenie klávesu. Ak chceme „rozsvecovanie“ spomaliť, stačí vložiť do cyklu príkaz delay(*milisekúnd*).

```
program ROZSVECOVANIE_V_OKNACH2;
uses crt;
begin
randomize;
clrscr;
repeat
  gotoxy(random(80)+1,random(24)+1);
  write(chr(219));
until keypressed;
readln
end.
```

V ďalšom príklade sa pokúsime usmerniť pohyb.

Príklad CRP-5: Vytvorte program na pád „vajíčka“ (písmena O) z hora nadol v strede riadkov.

Analýza: Kto si poctivo ujasnil predchádzajúce dva programy, nemôže mať s nasledujúcimi dvoma problémy. Mali by sme použiť príkaz for RIA:=1 to 25 do ... ale chceme od vás, aby ste použili príkaz repeat. Program sme doplnili aj o zvukový efekt. Ďalšie zvukové efekty môžete doprogramovať pre let vajíčka. Ak sa chcete pohrať, môžete pridať aj „gravitáciu“, aby vajíčko pri svojom páde zrýchľovalo.

```
program pad_vajca;
uses crt;
var  ria:byte;
begin
clrscr;
ria:=1;
repeat
  gotoxy(40,ria); write('.');
  gotoxy(40,ria+1); write('O');
  delay(150); inc(ria)
until ria=25;
sound(60);
gotoxy(36,12); write('B U M !!!');
gotoxy(37,ria); write('---^---');
delay(200);
nosound;
readln
end.
```

V ďalšej verzii programu PAD_VAJCA sa ho pokúsime zachrániť pred rozbitím. Nech program vygeneruje náhodne veľké písmeno (veľké písmená začínajú „na“ 65 a končia na 90) a zobrazí ho v ľavom hornom rohu obrazovky. Ak stihneme do dopadnutia vajíčka „do 25 riadku“ nájsť na klávesnici a stlačiť hľadaný kláves, pád sa zastaví.

```
program zachran_vajce;
uses crt;
var  RIA:byte;
     PISMENO, STLACIL:char;
begin
randomize;
clrscr;
PISMENO:=chr(random(26)+65);
gotoxy(1,5); write('Najdi ',PISMENO);
RIA:=1;
repeat
  gotoxy(40,RIA); write('.');
  gotoxy(40,RIA+1); write('O');
  if keypressed then STLACIL:=upcase(readkey);
  delay(100); inc(RIA)
until (STLACIL=PISMENO) or (RIA=25);
if RIA<25
  then begin gotoxy(33,12); write('V Y B O R N E !') end
  else begin
       sound(60);
       gotoxy(36,12); write('B U M !!!');
       gotoxy(37,RIA); write('---^---');
       delay(200); nosound
       end;
readln
end.
```

Ak vás pohyb na obrazovke zaujal, môžeme skúšať ďalej.

Príklad CRP-6: Vytvorte program na pohyb znaku po obrazovke. Na ovládanie pohybu použite šípky na ovládanie pohybu kurzora. Testujte aj „vybehnutie“ znaku z obrazovky.

Analýza: Tento program uvedieme bez hlbšej analýzy. Necháme ho na podrobné štúdium a experimentovanie. Ku kódom šípok sme sa dostali pomocou programu uvedeného nad príkladom CRP-2 (0 si nevšímajte, sú tam preto, že sa nejedná o obyčajné klávesy). Všimnite si cyklus repeat until keypressed, ktorý zamedzuje neustálemu prepisovaniu znaku na mieste a jeho blikaniu (prakticky zacyklí program na danom mieste až do stlačenia ľubovoľného klávesu). Program možno násilne kedykoľvek ukončiť stlačením klávesu Esc (príkaz halt – je to len ukážka, nenavykajte si naň!).

Programy začínajú byť zložitejšie a preto sme sa rozhodli začať nenápadne používať podprogramy (procedúry). Podprogram rieši nejaký čiastkový problém a je veľmi podobný programu (namiesto slova program používa slovo procedure). V príkazovej časti hlavného programu stačí uviesť meno procedúry a automaticky sa vykoná všetko, čo v nej je naprogramované.

```
program pohyb_znaku_pri_pridrzani_sipky_bez_blikania;
uses  crt;

const Esc=chr(27);
      sipkaVPRAVO=chr(77);     {M}
      sipkaVLAVO =chr(75);     {K}
      sipkaHORE  =chr(72);     {H}
      sipkaDOLE  =chr(80);     {P}

var   stl,ria:byte;

procedure inicializacia;
  begin
  clrscr;
  stl:=40; ria:=12; gotoxy(stl,ria); write('*');
  end;
```

```
procedure pohyb;
  var stlznak:char;
  begin
  repeat
    repeat until keypressed;
    stlznak:=upcase(readkey);
    gotoxy(stl,ria); write(' ');
    case stlznak of
      sipkaVPRAVO : inc(stl);
      sipkaVLAVO  : dec(stl);
      sipkaHORE   : dec(ria);
      sipkaDOLE   : inc(ria);
      Esc         : halt;
      end;
    gotoxy(stl,ria); write('*');
  until (stl=1)or(stl=80)or(ria=1)or(ria=25);
  gotoxy(25,13); writeln('B U M ! ! !  O H R A D A ! ! !');
  end;
BEGIN
inicializacia;
pohyb;
readln
END.
```

Pohyb je zaujímavejší, ak sa znak pohybuje určeným smerom, až kým nezvolíme iný smer. Čiže, kým nezvolíme iný smer (nestlačíme inú šípku), znak sa má uberať „starým“ smerom.

```
program pohyb_znaku_bez_podrzania_sipky;
uses  crt;
const Esc=chr(27);
      sipkaVPRAVO=chr(77);     {M}
      sipkaVLAVO =chr(75);     {K}
      sipkaHORE  =chr(72);     {H}
      sipkaDOLE  =chr(80);     {P}
var   stl,ria:byte;

procedure inicializacia;
  begin
  clrscr;
  stl:=40; ria:=12; gotoxy(stl,ria); write('*');
  end;

procedure pohyb;
  var stlznak:char;
  begin
  repeat
    if keypressed then stlznak:=upcase(readkey);
    gotoxy(stl,ria); write(' ');
    case stlznak of
      sipkaVPRAVO : inc(stl);
      sipkaVLAVO  : dec(stl);
      sipkaHORE   : dec(ria);
      sipkaDOLE   : inc(ria);
      Esc         : halt;
      end;
    gotoxy(stl,ria); write('*'); delay(70);
  until (stl=1)or(stl=80)or(ria=1)or(ria=25);
  gotoxy(25,13); writeln('B U M ! ! !  O H R A D A ! ! !');
  end;
BEGIN
inicializacia;
pohyb;
readln
END.
```

Spravme v programe ešte zopár úprav krásy. Po prvé, vyrobme okolo obrazovky ohradu (v dolnej časti po riadok 24). Potrebné kódy znakov sme si našli po zobrazení ASCII tabuľky. Môžete postupovať aj tak, že v konštantách budete mať priamo zobrazené príslušné semigrafické znaky v apostrofoch. druhá úprava krásy spočíva vo vypnutí zobrazovania kurzora, aby nám „nekazil celkový dojem“ (príkazy a potrebné premenné – registre pozná unit dos).

```
program pohyb_znaku_bez_podrzania_sipky2;
uses  crt,dos;
const ESC=chr(27);
      sipkaVPRAVO=chr(77);     {M}
      sipkaVLAVO =chr(75);     {K}
      sipkaHORE  =chr(72);     {H}
      sipkaDOLE  =chr(80);     {P}
var   stl,ria:byte;

procedure ohrada;
  var i:integer;
  begin
  write(chr(201)); for i:=2 to 79 do write(chr(205)); write(chr(187));
  for i:=2 to 23 do begin write(chr(186)); write(chr(186):79) end;
  write(chr(200)); for i:=2 to 79 do write(chr(205)); write(chr(188));
  end;

procedure skry_kurzor;
  var reg:registers;
  begin                {ukaz}
  reg.ah:=1;           {   1}
  reg.ch:=32;          {  13}
  reg.cl:=0;           {  40}
  intr($10,reg)
  end;

procedure inicializacia;
  begin
  clrscr;
  ohrada;
  skry_kurzor;
  stl:=40; ria:=12; gotoxy(stl,ria); write('*');
  end;

procedure pohyb;
  var stlznak:char;
  begin
  repeat
    if keypressed then stlznak:=readkey;
    gotoxy(stl,ria); write(' ');
    case stlznak of
      sipkaVPRAVO : inc(stl);
      sipkaVLAVO  : dec(stl);
      sipkaHORE   : dec(ria);
      sipkaDOLE   : inc(ria);
      Esc         : halt;
      end;
    gotoxy(stl,ria); write('*'); delay(70);
  until (stl=1)or(stl=80)or(ria=1)or(ria=24);
  gotoxy(25,13); write('B U M ! ! !  O H R A D A ! ! !');
  end;
BEGIN
inicializacia;
pohyb;
readln
END.
```

Predchádzajúci pohyb doplníme do hry. Nech program vygeneruje na náhodnom, ale vhodnom(!) mieste napríklad O, ktoré treba pohybujúcim sa znakom trafiť. Princíp je jednoduchý, pribudne testovanie, či sa nenachádza pohybujúci sa znak na pozícii znaku O.

```
program traf_znak;
uses  crt,dos;

const Esc=chr(27);
      sipkaVPRAVO=chr(77);     {M}
      sipkaVLAVO =chr(75);     {K}
      sipkaHORE  =chr(72);     {H}
      sipkaDOLE  =chr(80);     {P}

var   stl,ria,x,y:byte;

procedure skry_kurzor;
  var reg:registers;
  begin
  reg.ah:=1;
  reg.ch:=32;
  reg.cl:=0;
  intr($10,reg)
  end;

procedure ohrada;
  var i:integer;
  begin
  textcolor(white);
  write(chr(201)); for i:=2 to 79 do write(chr(205)); write(chr(187));
  for i:=2 to 23 do begin write(chr(186)); write(chr(186):79) end;
  write(chr(200)); for i:=2 to 79 do write(chr(205)); write(chr(188));
  end;

procedure inicializacia;
  begin
  randomize;
  skry_kurzor;
  clrscr;
  ohrada;
  x:=random(78)+2; y:=random(22)+2; gotoxy(x,y); write('O');
  stl:=40; ria:=12; gotoxy(stl,ria); write('*');
  end;

procedure pohyb;
  var stlznak:char;
  begin
  repeat
    if keypressed then stlznak:=readkey;
    gotoxy(stl,ria); write(' ');
    case stlznak of
      sipkaVPRAVO : inc(stl);
      sipkaVLAVO  : dec(stl);
      sipkaHORE   : dec(ria);
      sipkaDOLE   : inc(ria);
      Esc         : halt;
      end;
    gotoxy(stl,ria); write('*'); delay(100);
  until (stl=1)or(stl=80)or(ria=1)or(ria=24)or((stl=x)and(ria=y));
  if (stl=x)and(ria=y)
    then begin gotoxy(32,12); write('Z A S A H ! ! !') end
    else begin gotoxy(25,12); write('B U M ! ! !  O H R A D A ! ! !') end;
  end;
```

```
BEGIN
inicializacia;
pohyb;
readln
END.
```

Program môžete ďalej vylepšovať, napríklad nech sa náhodne pohybuje aj znak O. Ale to už nechávame na vašu šikovnosť.

Okrem dynamických (pohybujúcich sa) obrazcov je zaujímavé programovať aj statické obrazce. Väčšinou ide o požitie príkazov for, pričom

- ak obrazec ne je plný, cykly bývajú susedné
- ak je obrazec plný, cykly bývajú vnorené - vonkajší cyklus nastavuje riadok a vnútorný cyklus zabezpečuje pohyb v danom riadku.

Nasledujúci program demonštruje uvedené tvrdenia. Susedné cykly sú v procedúrach obdlznik1 a 2, stvorec a trojuholnik1 a 2; vnorené cykly sú v procedúrach obdlznik3 a trojuholnik3.

```
program OBRAZCE;
uses crt;
var n,i:integer;

procedure obdlznik1;
  begin
  clrscr;
  write('Strana obdlznika: '); readln(n);
  for i:=1 to n do write('*');
  writeln;
  for i:=2 to n-1 do writeln('*','*':n-1);
  for i:=1 to n do write('*');
  writeln;
  readkey
  end;

procedure obdlznik2;
  begin
  clrscr;
  write('Strana obdlznika: '); readln(n);
  for i:=1 to n do write('*');
  writeln;
  for i:=2 to n-1 do writeln('*','*':i-1,'*':n-i);
  for i:=1 to n do write('*');
  writeln;
  readkey
  end;

procedure stvorec;
  begin
  clrscr;
  write('Strana stvorca: '); readln(n);
  for i:=1 to n do write('* ');
  writeln;
  for i:=2 to n-1 do writeln('*','*':2*(n-1));
  for i:=1 to n do write('* ');
  writeln;
  readkey
  end;

procedure trojuholnik1;
  begin
  clrscr;
  write('Strana trojuholnika: '); readln(n);
  writeln('*');
```

```
  for i:=2 to n-1 do writeln('*','*':i-1);
  for i:=1 to n do write('*');
  writeln;
  readkey
  end;

procedure trojuholnik2;
  begin
  clrscr;
  write('Strana trojuholnika: '); readln(n);
  for i:=1 to n do write('*');
  writeln;
  for i:=2 to n-1 do writeln('*','*':n-i);
  writeln('*');
  readkey
  end;

procedure obdlznik3;
  var j:integer;
  begin
  clrscr;
  write('Strana obdlznika: '); readln(n);
  for i:=1 to n do
    begin
    for j:=1 to n do
      if i<>j then write('*') else write(' ');
    writeln
    end;
  readkey
  end;

procedure trojuholnik3;
  var j:integer;
  begin
  clrscr;
  write('Strana trojuholnika: '); readln(n);
  for i:=1 to n do
    begin
    for j:=1 to i do write('*');
    writeln
    end;
  readkey
  end;

begin
obdlznik1;
obdlznik2;
stvorec;
trojuholnik1;
trojuholnik2;
obdlznik3;
trojuholnik3;
end.
```

Pokúste sa vytvoriť program ktorý nakreslí rovnoramenný trojuholník s podstavou šírky n znakov, obrátenú pyramídu, kosoštvorec, štyri štvorce vedľa seba so stranou n znakov, štvorec s uhlopriečkami, plný obdĺžnik bez znakov na opačnej uhlopriečke, ako to je v procedúre obdlznik3.

Obrazce sú nekonečnou témou. Môžete sa pokúsiť naprogramovať aj pohyb napr. obdĺžnika (medzi jednotlivými polohami netreba mazať celú obrazovku).

Teraz trochu na oddýchnutie, aby sme však dlho neoddychovali, pobežia nám hodiny.

Príklad CRP-7: Vytvorte program simulujúci beh digitálnych hodín v strede obrazovky v tvare HH : MM : SS. Využite príkaz GetTime z unitu Dos.

Analýza: Program má vlastne do nekonečna (kým nestlačíme ľubovoľný kláves) čítať hodnoty systémového času pomocou príkazu gettime unitu dos a zobrazovať ich v strede obrazovky v predpísanom formáte. Problémom môže byť prechod z dvojciferného údaja na jednociferný, keď cifra z desiatok nebude premazaná – na to nesmieme zabudnúť.

```
program HODINY;
uses crt,dos;
var h,m,s,s100:word;
           reg:registers;
begin
{skry kurzor}
reg.ah:=1;
reg.ch:=32;
reg.cl:=0;
intr($10,reg);
clrscr;
repeat
  gettime(h,m,s,s100);
  gotoxy(33,12);
  if h<10 then write('0');
  write(h,' : ');
  if m<10 then write('0');
  write(m,' : ');
  if s<10 then write('0');
  write(s);
until keypressed;
end.
```

Doplňte program tak, aby hodiny pracovali ako budík („zazvonia“ v zadanom čase) prípadne časovač („cinknú“ po uplynutí zadaného času).

Na záver si ešte zopakujeme prácu s textom.

Príklad CWH-5: Vytvorte program, ktorý zistí, či zadané slovo je symetrické, t.j. či sa rovnako číta zľava aj sprava. Symetrické sú napríklad slová RADAR a ABBA.

Analýza: Zistiť, či zadané slovo je symetrické, znamená porovnať jeho prvý znak s posledným, ak sa rovnajú, porovnať jeho druhý znak s predposledným, ak sa rovnajú, porovnať..., až kým nenarazíme na dva rôzne znaky, alebo nie sme v strede slova (ľavá polovica slova sa rovná pravej polovici). Opakovane porovnávame vhodné znaky, preto treba použiť príkaz cyklu. Nevieme, kedy sa znaky už prestanú rovnať, je to cyklus s neznámym počtom opakovaní. Úloha je riešiteľná s príkazom while aj repeat, my sme sa rozhodli pre while. Keď sa po skončení cyklu s dvoma podmienkami ukončenia rozhodujete, pomocou ktorej testovať, prečo vlastne cyklus skončil, vyberte si vždy tú dôležitejšiu. V našom prípade to znamená testovať, či po skončení cyklu sa posledne testované písmená rovnajú alebo nie (if SLOVO[i]=SLOVO[DLZKA-i+1]), či sme už v strede slova, nie je natoľko dôležité.

```
program SYMETRICKE_SLOVO;
uses crt;
var SLOVO:string[80];
    i,DLZKA:byte;
```

```
begin
clrscr;
repeat
  write('Vloz slovo: '); readln(SLOVO);
  DLZKA:=length(SLOVO);
  i:=1;
  while (SLOVO[i]=SLOVO[DLZKA-i+1]) and (i<DLZKA div 2) do
    inc(i);
  if SLOVO[i]=SLOVO[DLZKA-i+1]
    then writeln('Text je symetricky')
    else writeln('Text nie je symetricky');
  writeln('KONIEC - stlac Esc')
until readkey=#27;
end.
```

Uvedený program možno zdokonaliť tým, že nebude rozlišovať medzi veľkými a malými písmenami a všetky ostatné znaky bude „ignorovať“. Potom aj text: „Jelenovi pivo nelej“ bude symetrický!

Analýza: Prvú požiadavku, nerozlišovať medzi veľkými a malými písmenami, možno vyriešiť pomocou funkcie upcase (známa vec). Druhú požiadavku, aby všetky ostatné znaky boli ignorované, možno dosiahnuť vytvorením nového textu, ktorý bude obsahovať len písmená. Využili sme množinový operátor in – je prvkom množiny (prvky množín sa uvádzajú v hranatých zátvorkách).

```
program SYMETRICKY_TEXT_2;
uses crt;
var TEXT,novyTEXT:string[80];
    i,j,DLZKA:byte;
begin
clrscr;
repeat
  write('Zadaj text: '); readln(TEXT);
  DLZKA:=length(TEXT);
  j:=0;
  for i:=1 to DLZKA do
    if TEXT[i] in ['A'..'Z','a'..'z']
      then begin
           inc(j);
           novyTEXT[j]:=upcase(TEXT[i])
           end;
  for i:=1 to j do
    TEXT[i]:=novyTEXT[i];
  DLZKA:=j;
  i:=1;
  while (TEXT[i]=TEXT[DLZKA-i+1]) and (i<DLZKA div 2) do
    inc(i);
  if TEXT[i]=TEXT[DLZKA-i+1]
    then writeln('Text je symetricky')
    else writeln('Text nie je symetricky');
  writeln('KONIEC - stlac Esc':45)
until readkey=#27;
end.
```

Neriešené úlohy na cykly:

1. Vytvorte program na výpočet súčinu A.B dvoch prirodzených čísel pomocou súčtu A + A + ...+ A (A ≥ B).
2. Vytvorte program výpočet mocniny s krokom x^2 (napr. $x^7 = x^2 . x^2 . x^2 . x$).
3. Vytvorte program na zistenie, či zadané číslo je prvočíslo.
4. Vytvorte program na vypísanie všetkých prvočísiel po zadané prirodzené číslo väčšie ako 1.
5. Vytvorte program na zistenie, či prirodzené číslo N < 341 je prvočíslo, ak viete, že platí veta:

   „N je prvočíslom práve vtedy , ak je deliteľom čísla 2^N-2 a N < 341“.
6. Vytvorte program na nájdenie všetkých prirodzených čísel po zadanú hranicu, ktorých ciferný súčet sa rovná zadanému číslu.
7. Vytvorte program, ktorý ako učebnú pomôcku vygeneruje tabuľku násobilky zadaného násobiteľa. Obmedzte hodnotu násobiteľa do 100 a násobenec nech má hodnoty z množiny {1,2,...,10 }.
8. Vytvorte program na prevod čísla z desiatkovej do dvojkovej sústavy.
9. Vytvorte program na prevod čísla z dvojkovej do desiatkovej sústavy.
10. Vytvorte program na prevod čísla z desiatkovej do šestnástkovej sústavy.
11. Vytvorte program na prevod čísla zo šestnástkovej do desiatkovej sústavy.
12. *Vytvorte program na zobrazenie podielu dvoch prirodzených čísel v tvare: {celá časť},{predperióda}{(perióda)}.

    Napr.: 3 / 2 = 1,5(0); 7 / 11 = 0,(63); 11 / 7 = 1,(571428) 31 / 60 = 0,51(6)

    (Platí veta: Predperióda je tvorená toľkými ciframi, koľkokrát je deliteľ deliteľný číslami 10, 5 a 2; napríklad deliteľ 60 je deliteľný 10 a 2).
13. Vytvorte program, ktorý zobrazí prevodnú tabuľku stupňov Celzia na Kelviny. Tabuľka začína 0^0C, končí hodnotou 100^0C s krokom po 10^0C. Tabuľka má mať hlavičku s nadpismi a pokúste sa o rámček zo semigrafických znakov (write(chr(...))).
14. Analogicky ako v predchádzajúcom príklade vytvorte prevodnú tabuľku uhlové stupne - radiány.
15. Vytvorte program , ktorý vypočíta dĺžku lomenej čiary určenej bodmi so súradnicami [x,y]. Na výpočet dĺžky úsečky (vzdialenosti dvoch bodov v rovine) použite Pythagorovu vetu.
16. Vytvorte program na nájdenie najväčšieho spoločného deliteľa troch prirodzených čísel.
17. Vytvorte program na vykreslenie grafu napr. funkcie sínus s osou x zvisle.
18. Vytvorte program na zistenie , či zadané prirodzené číslo je dokonalé. Číslo N je dokonalé, ak súčet všetkých jeho deliteľov menších ako N sa rovná N. Dokonalé je napríklad číslo 6.
19. Vytvorte obmenu programu CRP-5, keď budú postupne padať náhodne generované veľké písmená od prvého po 80. stĺpec. Na konci hry nech sa zobrazí úspešnosť v percentách (počet „zachytených“ písmen). Ďalšie varianty hry:
    - padanie sa bude postupne zrýchľovať, napr. po „zachytení“ 5 písmen za sebou
    - pád doplňte zvukovými efektami
    - hru doplňte ponukou: začiatočník – pokročilý – expert – koniec

- hru nech možno kedykoľvek ukončiť stlačením klávesu Esc
- vytvorte „nekonečný“ cyklus, keď po 80. stlpci pokračuje hra automaticky v 1. stĺpci
- písmená nech padajú v náhodne generovaných stĺpcoch
- padajúce písmená nech menia farbu
- atď.

Procedúry

Každý problém (okrem najjednoduchších) možno rozložiť na čiastkové podproblémy. Ich vyriešením a vykonaním v správnom poradí, dostávame riešenie daného problému. Programovacie jazyky na riešenie „podúloh“ ponúkajú podprogramy. 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 (každý poriadny program sa „rozpadá“ minimálne na zadanie vstupných hodnôt, výpočet a výstup získaných výsledkov) alebo
- ak potrebujeme vykonať rovnaký „výpočet“[1] viackrát s rôznymi vstupnými hodnotami (toto je praktický dôvod, aby sme neopisovali alebo nekopírovali tie isté príkazy viackrát v programe všade tam, kde potrebujeme vykonať rovnakú postupnosť príkazov).

Programovací jazyk TP má dva druhy podprogramov, procedúry a funkcie. V tejto kapitole sa budeme venovať procedúram.

Procedúry môžu byť:

- bez lokálnych objektov a bez parametrov,
- s lokálnymi objektami 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
   write(´Zadaj dve čísla: ´);
   readln(A,B);
   end;

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

procedure VYSTUP;
   begin
   writeln(A,' ',B)
   end;
```

[1] V súčasnosti sa už počítače na klasické výpočty používajú minimálne, častejšie na vyhľadávanie alebo triedenie dát.

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 hlavného programu:

```
BEGIN
clrscr;
VSTUP;
VYMENA;
VYSTUP;
readln
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 hlavného programu. Ako v celom TP, aj pri podprogramoch platí, že každý objekt musí byť najprv definovaný alebo deklarovaný a až potom ho môžeme použiť.

Procedúry s lokálnymi objektami 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 objektami má tvar:

procedure mp;
úsek definícií a deklarácií
príkazová časť
} blok

kde mp je meno procedúry.

Pamäťovo efektívnejší zápis procedúry VYMENA (s lokálnou premennou POM):

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

Volanie procedúry s lokálnymi objektami 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.

Procedúry s parametrami

Ak by sme procedúru POM 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 NAJVACSIE (A , B , C : integer; var MAX : integer);
procedure ZASIFRUJ (TEXT : string; POSUN : integer; var KOD : string);

Ak potrebujeme hodnoty do procedúry len doviesť, špecifikujeme 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, TEXT a POSUN.

Ak potrebujeme hodnoty z procedúry aj vyviesť, špecifikujeme 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, MAX a KOD.

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);
NAJVACSIE (5 , -3 , 7 , MAXIMUM); NAJVACSIE (X , Y , Z , Q);
ZASIFRUJ (´CEZAR´, 1 , POSLI); ZASIFRUJ (SPRAVA , N , KOD);

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.

Praktické použitie procedúr si ukážeme po prebratí jednorozmerného poľa.

Funkcie

Funkcia je špeciálnym prípadom procedúry. Procedúru môžeme zapísať ako funkciu, keď jej výsledkom je jedna jednoduchá hodnota alebo typ 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 MOCNINA (X : real; N : integer) : real;
function NACHADZA_SA : boolean;
function MAXIMUM (A , B , C : integer) : integer;

Príklad: Vytvorte podprogram na výpočet mocniny x^n, kde x je reálne číslo a n prirodzené číslo.

Analýza: Keďže výsledkom výpočtu je jedna jednoduchá hodnota, môžeme použiť aj funkciu. Na výpočet mocniny použijeme cyklus so známym počtom opakovaní: x^n = x.x.x. ... .x.

```
function MOCNINA (X : real; N : integer ) : real;
  var MOC : real;
      I : integer;
  begin
  MOC := 1;
  for I:=1 to N do
      MOC := MOC * I;
  MOCNINA := MOC
  end;
```

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: writeln (´Výsledok výpočtu mocniny: ´, MOCNINA (ZAKLAD , EXPONENT) : 12 : 10);
VYSLEDOK := MOCNINA (X , 3);

Príklad FNK-1: Vytvorte program na výpočet počtu kombinácií C(k,n) k-tej triedy z n prvkov $0 \le k \le n$.

Analýza: Platí vzorec $C(k,n) = \frac{n!}{(n-k)!.k!}$. Vidíme, že vo vzorci sa trikrát opakuje ten istý výpočet – výpočet faktoriálu (n! = 1.2.3. ... .n), len s rôznymi vstupnými hodnotami (n, n-k a k). Preto je výhodné použiť podprogram na výpočet faktoriálu a keďže výsledkom je jedna jednoduchá hodnota, konkrétne funkciu.

```
program KOMBINACIE;
uses  crt;
const Esc=chr(27);
var   N,K:integer;

function KOMB(N,K:integer):integer;

  function FAKT(N:integer):longint;       {lokalna funkcia}
    var I:integer;
        F:longint;
```

```
    begin
    F:=1;
    for I:=1 to N do
      F:=F*I;
    FAKT:=F
    end;

  begin     {príkazová časť funkcie KOMB }
  KOMB:=FAKT(N) div FAKT(N-K) div FAKT(K)
  end;

BEGIN
clrscr;
writeln('KOMBINACIE, KONIEC - Esc':50);
repeat
  write('Zadaj n a k ( n>=k>=0 ): ');readln(N,K);
  writeln('C (',N,',',K,') = ',KOMB(N,K));
until readkey=Esc;
END.
```

Program je skôr akademický, pretože hodnota n! už pre n = 13 prekročí maximálnu hodnotu typu longint (13! = 6 227 020 800).

Tak ako procedúry, aj funkcie si precvičíme až v ďalších kapitolách.

Rekurzia

Niektoré programovacie jazyky, vrátane TP, umožňujú používať veľmi elegantnú programovaciu techniku, tzv. rekurziu. Ak procedúra alebo funkcia vo svojom tele volá samu seba, hovoríme o priamej rekurzii; ak napríklad procedúra A volá procedúru B a naopak, hovoríme o nepriamej rekurzii.

Príklad REK-1: Vytvorte rekurzívnu funkciu na výpočet n!

Analýza: Vyjdeme z tzv. rekurentnej definície n-faktoriálu:

1. 0! = 1
2. n! = n.(n-1)! pre n>0

Ako vidieť z bodu 2, na vyčíslenie n! potrebujeme vykonať súčin n krát (n-1)!, pričom výpočet (n-1)! znamená rovnaký výpočet ako n!, len s hodnotou n zmenšenou o 1.

```
function FAKT ( N : integer) : longint;
   begin
   if N = 0
      then FAKT := 1                    { nerekurzívna vetva }
      else FAKT := N * FAKT ( N-1 )     { rekurzívna vetva }
   end;
```

Keďže z príkladu FNC-1 vieme, že klasický výpočet n! obsahuje cyklus, musí byť aj v príklade REK-1 niekde „schovaný“. K cyklu nás privedie poznanie, že počítač, ak nevie nejaký výraz, pre jeho zložitosť, vyčísliť, odkladá požiadavky na operácie do zásobníka a vracia sa k nim, keď ich už vie finalizovať. Preto pri výpočte napríklad 5! počítač odloží najprv požiadavku 5 krát „čosi“ (4! nepozná), potom 4 krát, 3 krát, 2 krát, 1 krát „čosi“, ale tu už „čosi“ je 0! a to je 1 (nerekurzívna vetva). Preto sa počítač vracia a vykonáva odložené operácie, t.j. 1.1.2.3.4.5 čím získa výsledok 120. Preto musíme vždy myslieť aj na ukončenie rekurzie, ukončenie volania danej procedúry alebo funkcie, čo zabezpečí nerekurzívna vetva.

Rekurzívny výpočet je časovo aj pamäťovo náročnejší ako iteračný výpočet (pomocou cyklu), niektoré problémy však rieši veľmi elegantne a relatívne jednoducho.

Príklad REK-2: Vytvorte rekurzívnu funkciu na výpočet mocniny x^n, x reálne číslo, n prirodzené číslo vrátane nuly.

Analýza: Potrebujeme znova poznať z matematiky rekurentnú definíciu mocniny, ktorá hovorí:

Pre $x \neq 0$:

1. $x^0 = 1$
2. $x^n = x \cdot x^{n-1}$ pre n>0

Vidíme, že v bode 2 máme na ľavej aj pravej strane rovnaký „problém“ – výpočet mocniny.

```
function MOCNINA ( X : real; N : word ): real;
  begin
  if N = 0
     then MOCNINA := 1
     else MOCNINA := X * MOCNINA ( X , N-1 )
  end;
```

Parameter X nie je nevyhnutný, X môže byť aj globálna premenná. Nezabudnite, že 0^0 nie je definované (ošetriť hneď na vstupe hodnôt X a N).

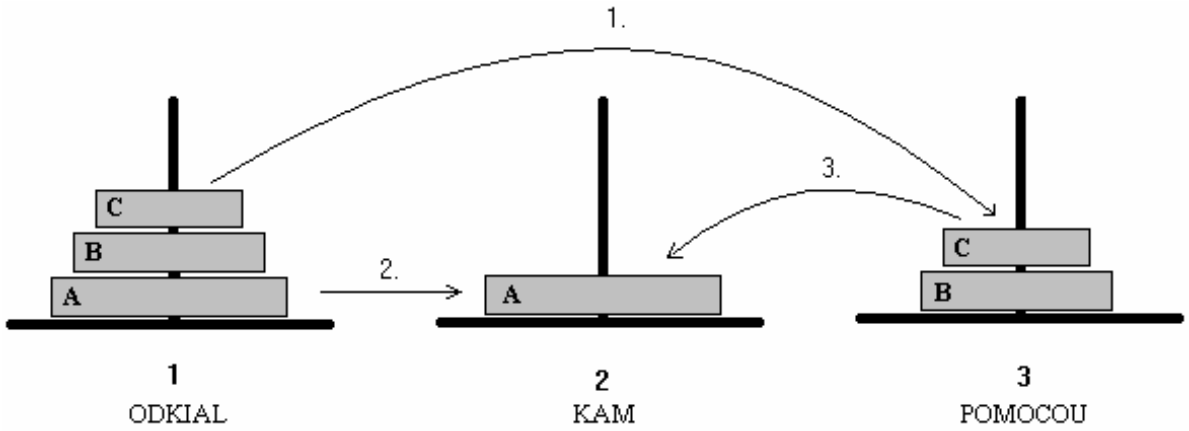
Príklad REK-3*: Vytvorte program riešiaci problém Hanojské veže.

Analýza: Ako ukážku sily a princípu práce rekurzie uvedieme známy problém nazvaný Hanojské veže. V starobylom kláštore ukrytom v ďalekom kúte Ázie vraj 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 platí:

- 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 (kolíky) sú označené 1, 2 a 3 (skutočné parametre). Pre označenie kolíkov sme zvolili premenné ODKIAL, KAM a POMOCOU (formálne parametre).



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,
2. potom môžeme disk A premiestniť z kolíka 1 na kolík 2 a
3. nakoniec presunúť vežu zvyšných N-1 diskov z kolíka 3 na kolík 2 pomocou kolíka 1.

Myšlienka použiť rekurziu by nám mohla napadnúť, ak si uvedomíme, že problém preniesť vežu s N diskami obsahuje v sebe problém preniesť vežu s N-1 diskami (ďalej už „len“ preložiť disk a naň vrátiť vežu s N-1 diskami).

Tu je program:

```
program HANOJ;
uses crt;
var  POCET:integer;

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

  procedure PRENES_DISK(Z,NA:integer);
    begin
    writeln(Z,' -> ',NA)
    end;

begin
if N>0
  then begin
       PRENES_VEZU(N-1,ODKIAL,POMOCOU,KAM);     {1}
       PRENES_DISK(ODKIAL,KAM);
       PRENES_VEZU(N-1,POMOCOU,KAM,ODKIAL);     {2}
       end;
end;

BEGIN
clrscr;
write('POCET DISKOV: ');
readln(POCET);
PRENES_VEZU(POCET,1,2,3);
readln;
END.
```

Jeho jednoduchosť asi prekvapí každého. Úspešnosť rekurzie spočíva v tom, čo sme uviedli v príklade REK-1, totiž, že ak počítač „nevie niečo vypočítať“, „odloží to“ do zásobníka. Tak počítač odkladá PRENES_VEZU 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 PRENES_DISK atď. Zložitosť výpočtu by mala byť zrejmá z výpisu, ktorý sme získali po doplnení vyššie uvedeného programu príkazmi na výpis, pričom prvé volanie procedúry PRENES_VEZU vo vetve then sme označili číslom 1 a druhé číslom 2.

```
POCET DISKOV = 3
Prenes vezu s 3 disk. pomocou volania 0  (prvé volanie z hlavného programu)
Prenes vezu s 2 disk. pomocou volania 1
Prenes vezu s 1 disk. pomocou volania 1
Prenes vezu s 0 disk. pomocou volania 1
1 -> 2
Prenes vezu s 0 disk. pomocou volania 2
1 -> 3
Prenes vezu s 1 disk. pomocou volania 2
Prenes vezu s 0 disk. pomocou volania 1
2 -> 3
Prenes vezu s 0 disk. pomocou volania 2
1 -> 2
Prenes vezu s 2 disk. pomocou volania 2
Prenes vezu s 1 disk. pomocou volania 1
Prenes vezu s 0 disk. pomocou volania 1
3 -> 1
Prenes vezu s 0 disk. pomocou volania 2
3 -> 2
Prenes vezu s 1 disk. pomocou volania 2
```

```
Prenes vezu s 0 disk. pomocou volania 1
1 -> 2
Prenes vezu s 0 disk. pomocou volania 2
```

Neriešené úlohy na rekurziu:

4. Vytvorte program na výpočet najväčšieho spoločného deliteľa (NSD) dvoch prirodzených čísel pomocou rekurzie, ak platí:

$$NSD(A,B) = \begin{cases} A & \text{ak } A=B \\ NSD(A-B,B) & \text{ak } A>B \\ NSD(A,B-A) & \text{ak } B>A \end{cases}$$

5. Vytvorte program na výpočet najväčšieho spoločného deliteľa (NSD) dvoch prirodzených čísel pomocou rekurzie, ak platí:

$$NSD(A,B) = \begin{cases} A+B & \text{ak } A=0 \text{ alebo } B=0 \\ NSD(A \bmod B,B) & \text{ak } A>B \\ NSD(A,B \bmod A) & \text{ak } B>A \end{cases}$$

6. Objasnite prácu rekurzívnej procedúry OTOC:

```
program ZRKADLO;
uses crt;
procedure OTOC;
  var z:char;
  begin
  read(z);
  if z<>'.'
    then OTOC
    else writeln;
  write(z)
  end;
BEGIN
clrscr;
writeln('Veta konci bodkou !':48);write(' ');
OTOC;
readkey
END.
```

7. Vytvorte program na výpočet ľubovoľného Fibonacciho čísla pomocou rekurzívnej a nerekurzívnej (iteračnej) funkcie a porovnajte časy výpočtu - urobte záver.

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

   vzorcami: $F_0 = 0$, $F_1 = 1$

   $F_i = F_{i-1} + F_{i-2}$ pre $i \geq 2$

8. Vytvorte podprogram na výpočet ľubovoľného člena Pascalovho trojuholníka, po zadaní nezáporných celých čísel n a k , ak platí:

   $B(n, 0) = B(n, n) = 1$

   $B(n, k) = B(n-1, k-1) + B(n-1, k)$ pre $1 \leq k \leq n$

9. Vytvorte funkciu CISLICA(n,k), ktorej výsledkom je k-ta číslica sprava prirodzeného čísla n.
10. *Vytvorte program, ktorý zadané prirodzené číslo n rozloží na prvočinitele, t.j. na prvočísla c_1, c_2, ...,c_k , pričom platí: $n = c_1 . c_2 . ... . c_k$.
11. *Vytvorte program, ktorý zistí, na koľko častí sa rozpadne štvorčekový papier po vystrihnutí niektorých štvorčekov.
12. *Vytvorte program na zistenie všetkých možných spôsobov vyplatenia zadanej sumy pri známych hodnotách použitých mincí (počet mincí rôznych hodnôt je neobmedzený).

Jednorozmerné pole

Predstavme si, že chceme vypočítať napríklad aritmetický priemer z veľkého počtu čísel a výstup má byť v tvare: Aritmetický priemer z čísel: *vypísané všetky čísla* je .... Na uchovanie väčšieho počtu údajov počas vykonávania programu s doteraz prebranými údajovými typmi nevystačíme. Pomôže nám nový údajový typ pole.

Konečnú skupinu rovnakých údajov (čísel, reťazcov a pod.) môžeme v programovaní výhodne uložiť do jednej štruktúrovanej premennej, do poľa. Pole sa skladá z pevného počtu zložiek rovnakého typu, pričom každú zložku možno sprístupniť pomocou indexu. Rôzne polia môžu obsahovať rôzne počty prvkov a aj ich hodnoty môžu byť rôznych typov (raz sú to čísla typu integer, druhýkrát reťazce typu string atď.). Preto rôzne polia predstavujú rôzne typy poľa, ktoré sa líšia počtom prvkov, typom zložky alebo typom indexu. Aby počítač vedel, akú veľkú pamäť má vyhradiť pre hodnoty premennej typu pole (skrátene pre pole), musíme v programe najprv typ pole definovať.

Definícia typu pole má tvar:

type mt = *array* [ti] *of* tz

kde mt je meno typu pole – identifikátor, ti je typ indexu – musí byť ordinálny typ a tz je typ zložky poľa.
Ako typ indexu sa najčastejšie používa tzv. interval, definovaný: min .. max, kde min a max sú hodnoty toho istého ordinálneho typu (medzi min a max sú práve dve bodky!).

Napríklad:	Pole vľavo uvedeného typu bude obsahovať:
type POLE1 = array [1..20] of integer;	20 čísel typu integer s indexami 1, 2, ..., 20
type POLE2 = array [-100..100] of boolean;	201 hodnôt false alebo true s indexami -100, -99, ...
type POCET = array [´A´..´Z´] of byte;	26 celých čísel od 0 do 255 s indexmi ´A´, ´B´,...,´Z´
type tVELA = array [byte] of string;	256 reťazcov s indexami 0, 1, 2 až 255
type HLUPOST = array [boolean] of boolean;	dva prvky s hodnotami false alebo true s indexami false a true

Uvedomte si, že súčasťou definície typu pole je aj pomenovanie daného typu. Ide o údajový typ s nami definovaným rozsahom, obsahom, indexami aj menom.

V úseku deklarácií premenných deklarujeme premenné typu pole rovnakým spôsobom ako iné premenné. Napríklad zápis: var A : POLE1 zavádza pole s názvom A typu POLE1. Sprístupnenie jednotlivých zložiek poľa A sa realizuje zápisom A [index], kde **index** je indexový výraz a musí byť rovnakého typu, ako je typ indexu z definície typu pole. Zápisu A [index] hovoríme **indexovaná premenná** a jej typ je určený typom zložky daného poľa.

Napríklad:

v poli A typu POLE1 znamená zápis:	A [1]	prvú zložku poľa A s hodnotou napríklad -7
	A [2]	druhú zložku poľa A
	A [I]	i-tu zložku poľa A
	A [20]	poslednú zložku poľa A
v poli B typu POLE2 znamená zápis:	B [-100]	prvú zložku poľa B s hodnotou napríklad true
	B [0]	101. zložku poľa B

v poli C typu POCET znamená zápis: C [´A´] prvú zložku poľa C s hodnotou napríklad 0

C [´Z´] poslednú zložku poľa C

Ak sú dve polia rovnakého typu, napríklad označené A a B, možno ich hodnoty odovzdávať naraz, napríklad príkazom B := A.

Premenná typu pole môže byť deklarovaná aj bez definície príslušného typu. Pre pole s názvom A by deklarácia mala tvar:

var A : *array* [ti] *of* tz

kde ti a tz majú rovnaký význam ako v definícii typu pole. Najprv definovať typ poľa a až potom deklarovať premennú daného typu má niekoľko výhod a preto odporúčame typ poľa najprv definovať. Napríklad v špecifikácii formálneho parametra v procedúre je dovolené na mieste typu parametra uviesť len meno poľa a nie zápis *array* [ti] *of* tz.

Pre programy s typom pole odporúčame používať podobný úsek definícií a deklarácií ako uvádzame nižšie:

```
program ...;
uses   crt;
const MAXPP = 20;              { definujeme konštantu udávajúcu maximálny počet zložiek - prvkov poľa }
type   POLE = array [ 1..MAXPP ] of integer;
var    A : POLE;
       N, I : integer;         { N bude aktuálny počet prvkov poľa A }
```

Načítanie prvkov poľa je čiastkový problém, ktorý je vhodné vyriešiť v procedúre. Ak v celom programe pracujeme len s jedným poľom, môžeme použiť procedúru bez parametrov, t.j. pracovať s globálnymi premennými. Ak v programe pracujeme s viacerými poliami, treba použiť parametre (ukážeme si v príklade neskôr). Princípom vstupu prvkov poľa je zapísať prvú zadanú hodnotu ako 1. prvok poľa, druhú ako 2. prvok poľa, atď. až po zápis posledného N-tého prvku poľa. N-krát sa opakuje rovnaká činnosť – zápis do poľa, preto použijeme cyklus so známym počtom opakovaní – príkaz for.

Príklad procedúry riešiacej načítanie prvkov globálneho poľa A s počtom prvkov N:

```
procedure VSTUP;       { bez parametrov }
  begin
  write(´Počet prvkov poľa: ´);
  readln(N);
  for I := 1 to N do          { Premenná I bude postupne nadobúdať hodnoty 1, 2, 3, ..., N }
    begin
    write(I:2,´. prvok: ´);    { Zápis I:2 spôsobí, že jednotky budú pod jednotkami a desiatky pod desiatkami }
    readln(A[I])
    end
  end;
```

Aj zobrazenie prvkov poľa je čiastkový problém riešiteľný v procedúre. Jednotlivé hodnoty poľa môžu byť na samostatných riadkoch (nevýhodné), oddelené medzerami alebo čiarkami.

Príklad procedúry slúžiacej na zobrazenie prvkov globálneho poľa A oddelených medzerami (predpokladáme, že výstupné hodnoty majú najviac 7 znakov):

procedure VYSTUP;

```
begin
for I := 1 to N do
  write(A[I]:8);          { keďže je v riadku 80 znakov, v každom bude zobrazených do desať hodnôt }
writeln
end;
```

Príklad procedúry slúžiacej na zobrazenie prvkov globálneho poľa A oddelených čiarkami:

procedure VYSTUP;

```
begin
for I := 1 to N-1 do              { čiarky majú byť za prvou až predposlednou hodnotou }
  write(A[I],´, ´);
writeln(A[N])
end;
```

Príklad POLE-1: Vytvorte program, ktorý načíta najviac 30 celých čísel z intervalu 0 až 999 a zobrazí ich.

Analýza: Požadovaný program stačí doslova poskladať z predchádzajúcich procedúr, len s malými úpravami: MAXPP treba zväčšiť na 30 a typ zložky poľa môže byť byte (potom v prvej procedúre VYSTUP vystačíme aj s formátom A[I]:4). Neznáma je už len príkazová časť hlavného programu:

```
BEGIN
clrscr;
VSTUP;                          { podľa „chuti“ môžete doplniť príkaz clrscr; }
writeln(´Prvky pola:´);
VYSTUP;
readln
END.
```

Odporúčame uložiť si tento program do súboru s názvom POLE a máme základ pre mnoho programov pracujúcich s poľom – každý by predsa mal mať vstup a výstup prvkov poľa. Po otvorení súboru POLE.PAS ho treba cez položku File – Save as... premenovať na aktuálny názov a doplniť o procedúru riešiacu zadanú úlohu.

Často prvky poľa môžu byť aj náhodne počítačom vygenerované čísla, stačí, ak zadáme len ich počet. Procedúra VSTUP_NAHODNE za nás „vyberie“ prvky poľa:

procedure VSTUP_NAHODNE;

```
begin
write(´Počet prvkov poľa: ´);
readln(N);
for I := 1 to N do
  A[I]:=random(10)              { pole A bude obsahovať celé čísla od 0 po 9 }
end;
```

V hlavnom programe za BEGIN treba vložiť príkaz randomize, aby sme znáhodnili výber prvkov poľa po každom spustení programu.

Prácu s poľom si precvičíme najmä na činnosti nazvanej vyhľadávanie. **Vyhľadávanie** v poli je činnosť s cieľom zistiť, či zložka so zadanými vlastnosťami sa nachádza v poli. Cieľom môže byť aj zistenie, koľkokrát sa zložka so zadanými vlastnosťami nachádza v poli, na ktorých miestach, prípadne požiadavka, ak sa zložka v poli nenachádza, vložiť ju za posledný prvok poľa a pod.

Príklad POLE-2: Vytvorte podprogram na zistenie najväčšieho prvku daného poľa.

Analýza: Maximum hľadajme v poli celých čísel. Na začiatku hľadania je najväčším prvkom prvý prvok poľa (jeho hodnotu uložíme do premennej MAX). Potom musíme zistiť, či druhý prvok poľa nie je väčší, ak áno, našli sme nové maximum (do premennej MAX musíme zapísať hodnotu druhého prvku poľa). Podobne porovnáme tretí prvok poľa s MAX a ak treba aktualizujeme MAX (zmeníme hodnotu MAX na hodnotu tretieho prvku poľa), potom štvrtý prvok, atď., až kým neprejdeme celé pole. Prejsť celé pole od prvého resp. druhého po posledný prvok znamená použiť príkaz for.

```
procedure MAXIMUM;
  var MAX:integer;           { lokálna premenná MAX }
  begin
  MAX:=A[1];                 { priradenie počiatočnej hodnoty premennej MAX }
  for I:=2 to N do           { zabezpečí prejdenie celého poľa od druhého prvku}
    if A[I]>MAX then MAX:=A[I];
  writeln('Z cisel:');
  VYSTUP;                          {zobrazí prvky poľa }
  writeln('je najvacsie ',MAX);
  end;

BEGIN
clrscr;
VSTUP;
MAXIMUM;
readln
END.
```

Procedúru MAXIMUM môžeme napísať aj ako funkciu, keďže výsledkom danej procedúry je jedna jednoduchá hodnota. Funkciu je výhodné použiť, ak chceme maximum vyviesť.

Program od funkcie MAXIMUM:

```
function MAXIMUM:integer;
  var MAX:integer;
  begin
  MAX:=A[1];
  for I:=2 to N do
    if A[I]>MAX then MAX:=A[I];
  MAXIMUM:=MAX;                    {priradenie najväčšej hodnoty menu funkcie,}
  end;                             {cez ktoré sa vyvezie daná hodnota}

BEGIN
clrscr;
VSTUP;
VYSTUP;
writeln('je najvacsie ',MAXIMUM);  {zápis funkcie na mieste,kde chceme zobraziť}
readln                             {výsledok funkcie}
END.
```

Príklad POLE-3: Vytvorte podprogram na zistenie, koľkokrát sa zadaný prvok nachádza v poli naviac 100 celých čísel.

Analýza: Zistiť, koľkokrát sa zadaný prvok nachádza v poli znamená prejsť celé pole a vždy keď sa prezeraný prvok poľa rovná hľadanému, zväčšiť počet výskytov o jednotku.

```
procedure ZISTI_POCET;
  var HLPRVOK,POCET:integer;
  begin
  write('Zadaj prvok, ktoreho pocet vyskytov mam zistit: ');
  readln(HLPRVOK);
  POCET:=0;
```

```
  for I:=1 to N do
    if A[I]=HLPRVOK then POCET:=POCET+1;      { analogicky inc(POCET); }
  if POCET=0
    then writeln('Hladany prvok sa v poli nevyskytuje!')
    else writeln('Hladany prvok sa v poli vyskytuje ', POCET,'-krat.')
  end;
BEGIN
clrscr;
VSTUP;
ZISTI_POCET;
readln
END.
```

Procedúra môže byť opäť napísaná aj ako funkcia, jej použitie vzhľadom na požiadavku zadať hľadaný prvok je trochu komplikovanejšia:

```
function ZISTI_POCET(HLPRVOK:integer):integer; {parameter nahradzovaný hodnotou}
  var POCET:integer;
  begin
  POCET:=0;
  for I:=1 to N do
    if A[I]=HLPRVOK then inc(POCET);
  ZISTI_POCET:=POCET
  end;

BEGIN
clrscr;
VSTUP;
write('Zadaj prvok, ktoreho pocet vyskytov mam zistit: ');
readln(HLADANY_PRVOK);              {HLADANY_PRVOK musí byť globálna premenná}
writeln('Hladany prvok sa v poli vyskytuje ', ZISTI_POCET(HLADANY_PRVOK),'-
krat.');
readln
END.
```

Príklad POLE-4: Vytvorte podprogram na zistenie, či sa prvok s danou hodnotou vyskytuje v poli.

Analýza: Kým doteraz sme museli prejsť celé pole, v tejto úlohe môžeme prehľadávanie poľa ukončiť po nájdení hľadanej hodnoty v poli. Z algoritmického hľadiska to znamená použiť časovo efektívnejší cyklus s neznámym počtom opakovaní. Výsledkom prehľadávania má byť len zistenie, či sa prvok v poli nachádza (vyvezie sa hodnota true) alebo nenachádza (vyvezie sa hodnota false).

```
function NACHADZA_SA:boolean;
  begin
  I:=0;
  repeat
    inc(I)
  until (A[I]=HLPRVOK) or (I=N);
  NACHADZA_SA:=A[I]=HLPRVOK              { Tento zápis nahrádza príkaz if }
  end;

BEGIN
clrscr;
VSTUP;
write('Zistit vyskyt prvku s hodnotou: ');
readln(HLPRVOK);
writeln('Prvok s hodnotou ',HLPRVOK,' sa v poli:');
VYSTUP;
if NACHADZA_SA
  then writeln('nachadza.')
  else writeln('nenachadza.');
readln
END.
```

Príklad POLE-4b: Ako zaujímavosť uvádzame rovnaký problém – zistiť, či sa prvok s danou vlastnosťou vyskytuje v poli, pričom však vieme, že pole je utriedené. Pod utriedeným poľom rozumieme vzostupné utriedenie, t.j. najmenší prvok je prvý a najväčší prvok je posledný, preto platí: A[I]≤A[I+1] pre všetky prípustné hodnoty I. Použiť by sme mohli aj predchádzajúcu funkciu NACHADZA_SA, ktorá používa tzv. lineárne vyhľadávanie – pole prehľadáva od prvého prvku prvok za prvkom, my však chceme časovo efektívnejší algoritmus (ak sa v tisíc prvkovom poli nachádza hľadané číslo napr. na 999 mieste, lineárne vyhľadávanie musí prezrieť 999 prvkov, nasledujúca funkcia však najviac 10!).

Analýza: Princípom binárneho vyhľadávanie, lebo tak je toto vyhľadávanie pomenované, je určenie stredu prehľadávanej oblasti a zistenie, či sme našli hľadaný prvok a ak nie, či sa nachádza v jeho dolnej (ľavej) časti alebo v hornej (pravej) časti. Tento postup sa opakuje dovtedy, kým sa nenájde hľadaný prvok, alebo keď už nie je čo prehľadávať (dolná hranica prehľadávanej oblasti je nad jej hornou hranicou).

```
function NACHADZA_SA:boolean;
  var DH,HH,S:integer;
  begin
  DH:=1; HH:=N;
  repeat
    S:=(DH+HH) div 2;
    if A[S]<>HLPRVOK
      then if A[S]<HLPRVOK
             then DH:=S+1
             else HH:=S-1
  until (A[S]=HLPRVOK) or (DH>HH);
  NACHADZA_SA:=A[S]=HLPRVOK
  end;
```

Ďalším typom úloh môžu byť úlohy zistiť, na ktorých miestach v poli sa vyskytuje prvok so zadanou vlastnosťou. Úloha môže byť formulovaná: zistiť prvý výskyt, všetky výskyty, posledný výskyt a pod.

Príklad POLE-5: Vytvorte podprogram na nájdenie prvého výskytu prvku so zadanou vlastnosťou.

Analýza: Nájsť miesto výskytu znamená vlastne určiť index prvku, ktorý má požadovanú vlastnosť (ak sú indexy poľa 1, 2, ..., N). Ak sa hľadaná hodnota v poli nevyskytuje, nech sa vyvezie 0. Keďže ide o jednu jednoduchú hodnotu, môžeme použiť funkciu. My sme sa rozhodli pre procedúru, aby sme ukázali použitie procedúry namiesto funkcie a aj preto, že funkčný zápis by sme museli použiť dvakrát (všade tam v hlavnom programe, kde sme použili premennú MIESTO), čo nie je efektívne. Ak sa chceme vyhnúť viacnásobnému použitiu funkčného zápisu s nezmenenými hodnotami, môžeme výsledok funkcie uložiť v hlavnom programe do premennej a ďalej pracovať len s touto premennou. V nasledujúcej časti programu sú premenné HLPRVOK a MIESTO globálne, nezabudnite ich deklarovať.

```
procedure PRVY_VYSKYT;
  begin
  I:=0;
  repeat
    inc(I)
  until (A[I]=HLPRVOK) or (I=N);
  if A[I]=HLPRVOK
    then MIESTO:=I
    else MIESTO:=0
  end;
```

```
BEGIN
clrscr;
VSTUP;
write('Zistit miesto prveho vyskytu prvku s hodnotou: ');
readln(HLPRVOK);
clrscr;
writeln('Prvok s hodnotou ',HLPRVOK,' sa v poli:');
VYSTUP;
PRVY_VYSKYT;
if MIESTO>0
  then writeln('nachadza na ',MIESTO,'. mieste.')
  else writeln('nenachadza.');
readln
END.
```

Veľmi zaujímavé sú aj kombinácie predchádzajúcich úloh, ako napríklad zistiť, či sa prvok s požadovanou vlastnosťou nachádza v poli a ak áno, koľkokrát, prípadne na ktorých miestach, prípadne aj koľkokrát a na ktorých miestach. Tieto kombinované úlohy väčšinou neznamenajú riešiť každú úlohu v samostatnom cykle (neefektívne), ale pri jednom prechode poľom možno „vyriešiť" všetky úlohy. Úlohy podobného typu sme dali medzi neriešené úlohy, teraz si uveďme aspoň jednu zložitejšiu.

Príklad POLE-6: Vytvorte program na zistenie najmenšieho prvku v poli, koľkokrát sa vyskytuje a na ktorých miestach v poli.

Analýza: Program nechávame na „samoštúdium". Všimnite si priradenie počiatočných hodnôt v procedúre MINIMUM a ako sme sa „pohrali" s výstupom (VYSTUP2).

```
program POLE_6;
uses  crt;
const MAXPP=20;
type  POLE=array[1..MAXPP] of integer;
var   A,VYSKYT:POLE;
      N,I,MIN,POCET:integer;

procedure VSTUP;
  begin
  write('Pocet prvkov pola: ');
  readln(N);
  for I:=1 to N do
    begin
    write(I:2,'. prvok: ');
    readln(A[I]);
    end;
  end;

procedure VYSTUP1;
  begin
  for I:=1 to N-1 do
    write(A[I],', ');
  writeln(A[N]);
  end;

procedure MINIMUM;
  begin
  MIN:=A[1]; POCET:=1; VYSKYT[1]:=1;      { priradenie počiatočných hodnôt }
  for I:=2 to N do
    if A[I]<=MIN
      then if A[I]=MIN        { prvok s hodnotou rovnou momentálnemu minimu }
             then begin
                  inc(POCET);
                  VYSKYT[POCET]:=I
                  end
```

```
            else begin            { nové minimum }
                 MIN:=A[I];
                 POCET:=1;
                 VYSKYT[1]:=I
                 end
  end;

procedure VYSTUP2;
  begin
  writeln('je najmensim prvok ',MIN);
  write('a vyskytuje sa ',POCET,'-krat ');
  if POCET=1
    then write('na mieste: ',VYSKYT[1])
    else begin
         write('na miestach: ');
         for I:=1 to POCET do write(VYSKYT[I],'  ')
         end
   end;

BEGIN
clrscr;
VSTUP;
clrscr;
writeln('V poli:');
VYSTUP1;
MINIMUM;
VYSTUP2;
readln
END.
```

Ďalšiu techniku, alebo ak chcete „fintu“, ktorú by mal ovládať každý programátor, je použitie tzv. nárazníka. Jeho princíp a použitie si ukážeme na nasledujúcom príklade:

Príklad POLE-7: Vytvorte podprogram, ktorý zistí, či sa hľadaný prvok nachádza v poli, a ak nie, vloží ho za posledný prvok.

Analýza: Princíp použitia nárazníka spočíva vo vložení hľadaného prvku do poľa, v tomto prípade za posledný prvok. To zabezpečí zastavenie prehľadávania (prvok sa určite v takto upravenom poli nachádza) najneskôr na nárazníku. Ak sa prehľadávanie poľa zastaví skôr, hľadaný prvok sa nachádza už v pôvodnom poli. Ak sa hľadaný prvok v poli nenachádza, už ho máme vložený za posledný prvok, len nesmieme zabudnúť zväčšiť počet prvkov poľa (hodnotu N).

```
procedure NARAZNIK;
  begin
  A[N+1]:=HLPRVOK;       { vlozili sme naraznik }
  I:=0;
  repeat
    I:=I+1
  until A[I]=HLPRVOK;
  if I=N+1 then N:=N+1   { prvok sa v poli nenachadzal, pridany dalsi prvok }
  end;

BEGIN
clrscr;
VSTUP;
write('Zistit vyskyt prvku s hodnotou: ');
readln(HLPRVOK);
writeln('Povodne pole:'); VYSTUP;
NARAZNIK;
writeln('Nove pole:'); VYSTUP;
readln
END.
```

Na nasledujúcom príklade „z trochu iného súdka“ si ukážeme, aké je dôležité dobre si zvoliť typ pole.

Príklad POLE-8: Vytvorte program, ktorý po zadaní textu prekonvertujte malé písmená v texte na veľké a zobrazí tabuľku počtu výskytov jednotlivých písmen abecedy v texte.

Analýza: Pole zvolíme tak, aby indexami poľa boli postupne všetky veľké písmená abecedy a hodnotami daného poľa budú čísla udávajúce počet výskytov jednotlivých písmen v texte.

```
program POLE_8;
uses  crt;
type  POLE=array['A'..'Z'] of byte;
var   TEXT:string;
      POCET:POLE;

procedure VSTUP;
  procedure KONVERZIA;
    var I:byte;
    begin
    for I:=1 to length(TEXT) do
      TEXT[I]:=upcase(TEXT[I])           { malé písmená v TEXTe zmení na veľké }
    end;
  begin
  write('Zadaj text: '); readln(TEXT);
  KONVERZIA
end;

procedure ZISTI_POCTY;
  var I:byte;
  procedure VYNULUJ;
    var ZNAK:char;
    begin
    for ZNAK:='A' to 'Z' do
      POCET[ZNAK]:=0
    end;
  begin
  VYNULUJ;
  for I:=1 to length(TEXT) do
    if TEXT[I] in ['A'..'Z']
      then inc(POCET[TEXT[I]]);   { TEXT[I] predstavuje konkrétne písmeno textu }
  end;

procedure VYSTUP;
  var ZNAK:char;
  begin
  writeln('Znak          Pocetnost':45);
  for ZNAK:='A' to 'Z' do
    begin
    writeln(ZNAK:24,POCET[ZNAK]:17);
    if ZNAK='W' then begin readkey; clrscr end
    end
  end;

BEGIN
clrscr;
VSTUP;
ZISTI_POCTY;
VYSTUP;
readln
END.
```

Typové konštanty

Typové konštanty môžeme prirovnať k premenným, ktorých počiatočné hodnoty sú uvedené už v úseku definícií a deklarácií. Môžeme ich aj používať rovnako ako premenné takého istého typu.

Deklarácia typovej konštanty má tvar:

const identifikátor_premennej : identifikátor_typu = typová_konštanta

Napríklad:

```
const   SYM : boolean = TRUE;
        stringANO : string[3] = ´ANO´ ;
        A : array[1..10] of integer = (15,2,-4,6,9,12,0,-7,6,11);
        CISLICE : array[0..9] of char = (´0´,´1´,´2´,´3´,´4´,´5´,´6´,´7´,´8´,´9´);
        CISLICE : array[0..9] of char = ´0123456789´;      { táto deklarácia je totožná s predchádzajúcou }
```

Príklad POLE-9: V príklade VCS-1 sme v programe použili na výpočet počtu dní do konca roka (dkr) „otrasný“ zápis v príkaze case. Pri použití typovej konštanty poľa sa zápis podstatne zjednoduší. Do ukážky sme pridali aj výpočet počtu dní od začiatku roka (ozr).

```
const dvm:array[1..12] of 1..31=(31,28,31,30,31,30,31,31,30,31,30,31);
      { dvm – počet dní v mesiaci }
...
writeln;
dkr:=dkm;
for i:=12 downto m+1 do
  dkr:=dkr+dvm[i];
if (m<2) and priestupny then inc(dkr);
write('Do konca roka ');
case dkr of
      1: writeln('zostáva 1 deň.');
  2,3,4: writeln('zostávajú ', dkr,' dni.');
  else writeln('zostáva ', dkr,' dní.')
end;
{ výpočet počtu dní od začiatku roka }
writeln;
ozr:=d;
for i:=1 to m-1 do
  ozr:=ozr+dvm[i];
if (m>2) and priestupny then inc(ozr);
write('Od zaciatku roka ');
case ozr of
      1: writeln('ubehol 1 deň.');
  2,3,4: writeln('ubehli ', ozr,' dni.');
  else writeln('ubehlo ', ozr,' dní.')
end;
readln
end.
```

Samozrejme, ak poznáme počet dní do konca roka, počet dní od začiatku roka sa dá vypočítať aj jednoduchšie.

Príklad POLE-10: Vytvorte program na prevod správy – textu do morseovej abecedy.

Analýza: Príklad je typický pre použitie typovej konštanty pole. Doplnili sme ho aj o zvukový výstup. Program ako „nepovinný“ ponechávame na samoštúdium.

```
program POLE_10;
uses  crt;
const TAB:array['A'..'Z'] of string[4] = ('.-','-...','-.-.','-..','.','..-.',
'--.','....','..','.---','-.-','.-..','--','-.','---','.--.','--.-','.-.','...',
' ','...-', '...-','.--','-..-','-.--','--..');
      takt=100;
      bodka=takt;
      ciarka=3*takt;
      medzi_znakmi=takt;
      medzi_bc=takt;
      Hz=1000;
var   s:string;
      i:byte;

procedure zvuk(MSznak:string);
  var i:byte;
  begin
  write(MSznak);
  for i:=1 to length(MSznak) do
    begin
    if MSznak[i]='|'
      then Delay(medzi_znakmi)
      else begin
           Sound(Hz);
           if MSznak[i]='.'
             then Delay(bodka)
             else Delay(ciarka);
           NoSound
           end;
    Delay(medzi_bc)
    end
  end;

BEGIN
clrscr;
write('Zadaj retazec: '); readln(s);
for i:=1 to length(s) do
  case s[i] of
    'A'..'Z','a'..'z': zvuk( TAB[ UpCase(s[i]) ] + '|' );
    ' '              : writeln
  end;
readln
END.
```

Okrem vyhľadávania sa často údaje aj triedia. Utriediť prvky poľa znamená vykonať činnosť, po končení ktorej pre prvky poľa napr. A platí: A[I] ≤ A[I+1] pre všetky dovolené hodnoty I. Triediacich algoritmov je veľa. Každý by mal ovládať aspoň ten najjednoduchší, ktorým môže byť napríklad bubblesort.

Príklad POLE-11: Vytvorte podprogram na utriedenie číselného poľa bubblesortom.

Analýza: Princípom práce bubblesortu je prechádzať poľom a porovnávať dva vedľa seba stojace prvky; ak je ľavý prvok väčší ako pravý, vymeniť ich. Ak si predstavíte túto činnosť, malo by byť zrejmé, že po prvom prechode poľom je najväčší prvok na svojom miesta, t.j. na konci poľa. Po druhom prechode poľom je na svojom mieste druhý najväčší prvok, atď. Pri n-prvkovom poli musí byť po n-1 prechodoch pole utriedené. Podstatou algoritmu sú dva vnorené cykly: vonkajší cyklus zabezpečuje pri n-prvkovom poli n-1 prechodov daným poľom (pp – počet prechodov) a vnútorný cyklus zabezpečuje pohyb v poli od prvého po n-tý prvok (premenná i). Vnútorný cyklus možno ešte zefektívniť, ak zohľadníme aj to, že po prvom prechode poľom je najväčší prvok na svojom mieste a preto pri ďalších prechodoch poľom už nemusíme ísť až na koniec poľa;

analogicky po druhom prechode poľom je už aj druhý najväčší prvok poľa na svojom mieste a preto v nasledujúcich prechodoch poľom nemusíme porovnávať už ani predposledný prvok, atď.

1. prechod	treba porovnať n prvkov	t.j. ísť po n-1. prvok
2. prechod	treba porovnať n-1 prvkov	t.j. ísť po n-2. prvok
3. prechod	treba porovnať n-2 prvkov	t.j. ísť po n-3. prvok
...		
pp. prechod		treba ísť po n-pp. prvok

```
procedure bubblesort;
  var pp:integer;
  procedure vymena;
    var pom:real;
    begin
    pom:=a[i]; a[i]:=a[i+1]; a[i+1]:=pom;
    end;
  begin
  for pp:=1 to n-1 do         { počet prechodov }
    for i:=1 to n-pp do       {pohyb v poli }
      if a[i]>a[i+1] then vymena;
  end;

BEGIN
clrscr;
vstup;
writeln('Povodne pole:'); vystup;
bubblesort;
writeln('Utriedene pole:'); vystup;
readln
END.
```

Aspoň jedným príkladom si pripomenieme, že aj údajový typ string (reťazec) je špeciálnym prípadom jednorozmerného poľa. Teoreticky ide o pole typu array [0..255] of char, ktorého prvky môžeme načítať „naraz“ príkazom read a zobraziť tiež „naraz“ bez cyklu for príkazom write (znak s indexom 0 má osobitnú funkciu – je v ňom uložená aktuálna dĺžka reťazca). Môžeme však sprístupniť i-ty znak reťazca indexovanou premennou ako u poľa.

Príklad POLE-12: Vytvorte program, ktorý zistí, či zadané dva reťazce sa skladajú z tých istých písmen a v rovnakom počte. Takéto reťazce nazývame anagramy.

Analýza: Najľahšie sa zistí, či majú zadané reťazce rovnakú dĺžku. Ak ich dĺžky sú rôzne, nemôžu byť anagramy. Ak sa ich dĺžky rovnajú, musíme zistiť, či sa skladajú z rovnakých písmen a v rovnakom počte. Znamená to zobrať prvý znak v prvom reťazci a hľadať jeho výskyt v druhom reťazci, ak je všetko „v poriadku“, zobrať druhý znak z prvého reťazca atď. Teda cyklus v cykle, kde vonkajší cyklus nastavuje prvý, druhý,... znak prvého reťazca a vnorený cyklus „prechádza“ druhým reťazcom a hľadá v ňom zodpovedajúci znak. Najefektívnejšie sú cykly s neznámym počtom opakovaní. Ďalším problémom je zabezpečiť, aby sme zistili, či sa písmená zhodujú aj v počtoch. Najjednoduchšie je asi „vyrábať“ pri prechode druhým reťazcom postupne reťazec totožný s prvým. Pre ozrejmenie programu si spravte pomocný výpis upravovaného druhého reťazca príkazom write(y), ako to je v poznámke programu.

```
program POLE_12;
uses crt;
var  slovo1,slovo2:string[80];

function anagram(x,y:string):boolean;
  var i,j,dlzkax,dlzkay:byte;
      anag:boolean;
      pom:char;
  begin
  dlzkax:=length(x); dlzkay:=length(y);
  if dlzkax=dlzkay
    then begin
         anag:=true;
         i:=1;
         while anag and (i<=dlzkax) do              { vonkajší cyklus }
           begin
           j:=i;
           while (x[i]<>y[j]) and (j<dlzkay) do     { vnútorný cyklus }
             j:=j+1;
           anag:=x[i]=y[j]; pom:=y[i]; y[i]:=y[j]; y[j]:=pom;
           i:=i+1;                          { doplňte príkaz writeln(y) }
           end;
         anagram:=anag
         end
    else anagram:=false
  end;

BEGIN
clrscr;
repeat
  write('Prve  slovo: '); readln(slovo1);
  write('Druhe slovo: '); readln(slovo2);
  if anagram(slovo1,slovo2)
    then writeln('JE TO ANAGRAM':44)
    else writeln('NIE JE TO ANAGRAM':46);
  writeln('KONIEC - stlac Esc');
  writeln
until readkey=chr(27);
END.
```

Ak sa niekomu zapáčil pohyb znaku po obrazovke, môže pokračovať v príklade CRP-6, a pokúsiť sa naprogramovať jednu z prvých hier na počítačoch – červíka Wurmi.

Príklad POLE-13: Vytvorte program – hru, v ktorej budú na obrazovke náhodne vygenerované krížiky (x) - jed a listy - potrava (♠ chr(6)). Šípkami na riadenie pohybu kurzora nech sa ovláda pohyb červíka Wurmi (znak chr(1)), ktorého úlohou je „požierať“ listy a vyhýbať sa jedu. Ak narazí na jed, hra končí. Hra končí aj „narazením do okrajov obrazovky“.

Analýza: Pri programovaní tejto hry použite príklad CRP-6, v ktorom máme vyriešené ovládanie pohybu znaku na obrazovke vrátane ošetrenia „trafenia“ znaku na obrazovke a „narazenie do okraja obrazovky“. Ďalšie zlepšenie hry si vyžaduje použitie polí, aby bolo možné zapamätať si pozície (x-ové a y-nové súradnice) jedu a potravy. Po každom premiestnení Wurmiho treba testovať, či „nestojí“ na jede alebo potrave. Ak stojí na potrave, treba započítať, že „zjedol“ ďalší list. Hra končí, keď:

- „zožerie“ všetky vygenerované listy
- „zožerie“ jed
- „narazí“ do okrajov obrazovky.

```
program POLE_13;
uses  crt,dos;

const pocet=5;                { počet vygenerovaných jedov a listov }
      ESC=chr(27);
      sipkaVPRAVO=chr(77);
      sipkaVLAVO =chr(75);
      sipkaHORE  =chr(72);
      sipkaDOLE  =chr(80);
      list       =chr(6);

var   stl,ria,body:integer;
      px,py,jx,jy:array[1..pocet] of integer;

procedure skry_kurzor;
  var reg:registers;
  begin
  reg.ah:=1;
  reg.ch:=32;
  reg.cl:=0;
  intr($10,reg)
  end;

procedure ohrada;
  var i:integer;
  begin
  textcolor(white);
  write(chr(201)); for i:=2 to 79 do write(chr(205)); write(chr(187));
  for i:=2 to 23 do begin write(chr(186)); write(chr(186):79) end;
  write(chr(200)); for i:=2 to 79 do write(chr(205)); write(chr(188));
  end;

procedure generuj_potravu_jed;
  var i:integer;
  begin
  for i:=1 to pocet do
    begin
    px[i]:=random(78)+2; py[i]:=random(22)+2;  { generuje potravu }
    jx[i]:=random(78)+2; jy[i]:=random(22)+2;  { generuje jed }
    end
  end;

procedure vykresli_potravu_jed;
  var i:integer;
  begin
  for i:=1 to pocet do
    begin
    gotoxy(px[i],py[i]); write(list);
    gotoxy(jx[i],jy[i]); write('x');
    end;
  end;

procedure inicializacia;
  var limit:integer;
  begin
  randomize;
  generuj_potravu_jed;
  skry_kurzor;
  clrscr;
  ohrada;
  vykresli_potravu_jed;
  stl:=40; ria:=12; gotoxy(stl,ria); write(chr(1));
  body:=0
  end;
```

```
function jed:boolean;   { testuje, či Wurmi nie je na poličku jed }
  var i:integer;
  begin
  i:=0;
  repeat
    inc(i)
  until ((jx[i]=stl) and (jy[i]=ria)) or (i=pocet);
  jed:=(jx[i]=stl) and (jy[i]=ria)
  end;

procedure potrava;      { testuje, či Wurmi nie je na políčku list }
  var i:integer;
  begin
  i:=0;
  repeat
    inc(i)
  until ((px[i]=stl) and (py[i]=ria)) or (i=pocet);
  if (px[i]=stl) and (py[i]=ria)
    then begin inc(body); px[i]:=0; py[i]:=0 end;
  gotoxy(3,2); write(body);
  end;

procedure pohyb;
  var stlznak:char;
  begin
  repeat
    if keypressed then stlznak:=readkey;
    gotoxy(stl,ria); write(' ');
    case stlznak of
      ESC:halt;
      sipkaVPRAVO:inc(stl);
      sipkaVLAVO :dec(stl);
      sipkaHORE  :dec(ria);
      sipkaDOLE  :inc(ria);
      end;
    gotoxy(stl,ria); write(chr(1)); potrava; delay(150);
  until (stl=1) or (stl=80) or (ria=1) or (ria=24) or jed or (body=pocet);
  if jed
    then begin gotoxy(33,12); write(' J E D ! ! ! ') end
    else if body=pocet
           then begin gotoxy(20,12); write('V Y B O R N E !!! VYHRAL SI!!!') end
           else begin gotoxy(25,12); write('B U M !!! OHRADA!!!') end
  end;

BEGIN
inicializacia;
pohyb;
readln
END.
```

Neriešené úlohy na jednorozmerné pole a typ string:

14. Vytvorte program na zistenie najmenšieho a najväčšieho prvku v poli najviac 100 celých čísel a výmenu najmenšieho prvku s prvým a najväčšieho prvku s posledným prvkom v poli.
15. Vytvorte program na nájdenie prvých dvoch najväčších (najmenších) čísel v poli najviac 20 rôznych celých čísel.
16. Vytvorte program na nájdenie k (k≤n) maxím (1. môžu byť aj rovnaké, 2. k rôznych maxím) v poli s n prvkami.

    Návod: využite upravený bubblesort.

17. Vytvorte program na zistenie počtu núl (kladných čísel, párnych čísel, čísel deliteľných zadaným číslom, čísel zo zadaného intervalu a pod.) v zadanom číselnom poli.
18. Vytvorte program na zistenie posledného výskytu (všetkých výskytov) zadaného znaku v texte.
19. Vytvorte program na zistenie polohy maxima a minima v danom poli (ošetrite aj viacnásobný výskyt).
20. Vytvorte program, ktorý prvky poľa napr. 2, 4, 5, 3, 8, 1, 3, 5, 9, 4, 7 zobrazí v tvare:

    2, 4, 5,
    3, 8, (3 < 5)
    1, 3, 5, 9, (1 < 8)
    4, 7 (4 < 9)
21. Vytvorte program, ktorý z pôvodného poľa odstráni všetky prvky so zadanou vlastnosťou (nepárne čísla, záporné čísla, rovné zadanej hodnote, znaky zo zadaného intervalu a pod.).
22. Vytvorte program, ktorý zlúči dve polia s prvkami rovnakého typu (prvky druhého poľa dá za prvky prvého poľa).
23. Vytvorte program, ktorý zlúči dve usporiadané polia s prvkami rovnakého typu tak, že výsledné pole bude tiež usporiadané.
24. Vytvorte program na výpočet a zobrazenie Pascalovho trojuholníka v tvare:

    1
    1 1
    1 2 1
    1 3 3 1
    1 4 6 4 1
    1 1+4 4+6 6+4 4+1 1
    ...

    Návod: Využite vzorce v zadaní REK-8, nepoužite však rekurziu ale pole.
25. Vytvorte program na uloženie prvkov pôvodného poľa opačne, vymeniť prvý prvok s posledným, druhý s predposledným atď.
26. Vytvorte program na posunutie prvkov poľa o k miest doprava (doľava). Posledných k prvkov poľa bude posunutých na začiatok poľa.
27. Vytvorte program na usporiadanie prvkov (čísel, znakov, reťazcov) daného poľa zostupne (od najväčšieho po najmenší).
28. Vytvorte program na výpočet ciferného súčtu prirodzeného čísla využitím typu string.
29. *Vytvorte program na zobrazenie všetkých prvočísiel od 2 po 60 000 metódou tzv. Eratosténovho sita. Návod: Napíšeme si čísla od 2 do 60 000. Dáme vypísať 2 a škrtneme všetky jej násobky po 60 000. Zoberieme a vypíšeme ďalšie číslo: 3 (nie je škrtnuté, preto je prvočíslo) a škrtneme všetky jeho násobky. Zoberieme ďalšie číslo: 4, je škrtnuté a preto nie je prvočíslo. Zoberieme ďalšie číslo: 5, nie je škrtnuté, preto ho vypíšeme ako prvočíslo a škrtneme všetky jeho násobky. Číslo 6 je škrtnuté. Číslo 7 vypíšeme (nie je škrtnuté) a škrtneme všetky jeho násobky (14, 21, 28,...). Takto pokračujeme až po 60 000. Môžeme použiť napríklad typ pole array [2..60000] of boolean, kde true znamená, že dané číslo nie je škrtnuté (je prvočíslo) a false znamená, že je škrtnuté.
30. Zefektívnite bubblesort tým, že ukončíte triedenie (prechody poľom), ak pri poslednom prechode už nedošlo ani k jednej výmene susedných prvkov (pole musí byť už utriedené).
31. *Vytvorte program, ktorý zistí, koľko anagramov sa nachádza v zadanom zozname slov.

Dvojrozmerné pole

V definícii typu pole:

type mt = *array* [ti] *of* tz

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 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 pole a tz je typ zložky.

Dovolený je aj skrátený zápis definície typu pole:

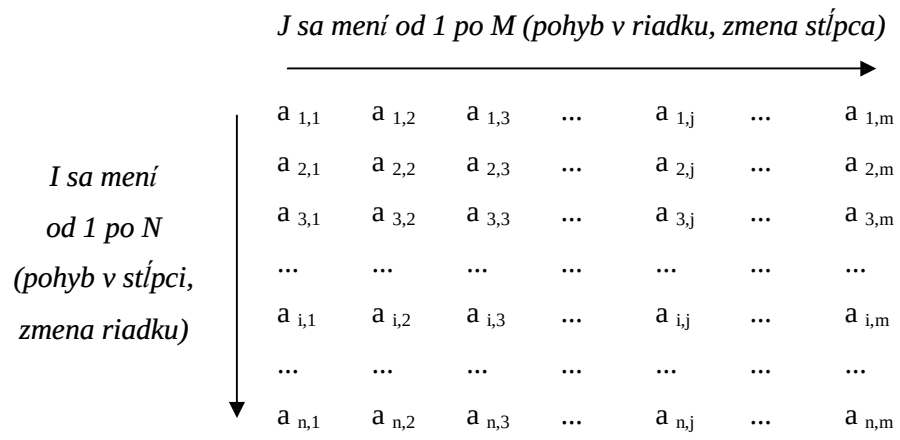
type mt = *array* [ti1 , ti2] *of* tz

Napríklad:

```
type    MATICA1 = array [1..10 ] of array [1..10] of integer;
        MATICA2 = array [1..10,1..10] of integer;       { definícia rovnocenná s MATICA1 }
        SACHOVNICA=array[´a´..´h´,1..8] of boolean;
        NEZMYSEL=array[boolean,boolean] of char;
```

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. Pre ľahšie pochopenie príkladov budeme ďalej používať nasledujúcu symboliku (častejšie ako dvojrozmerné pole budeme používať pojem matica, pretože tá predstavuje model na riešenie nášho problému; až keď je známy model na riešenie, môžeme ho realizovať v počítači dvojrozmerným poľom):

- matica typu N x M („en krát em“) obsahuje N riadkov a M stĺpcov
- pre zmenu riadku, t.j. pohyb v stĺpci budeme používať premennú I
- pre zmenu stĺpca, t.j. pohyb v riadku budeme používať premennú J
- schematicky (matica typu N x M):



Z hľadiska algoritmických konštrukcií pre prácu s maticou väčšinou potrebujeme cyklus v cykle. Vonkajší cyklus zabezpečuje najčastejšie nastavenie príslušného riadku (for I:=1 to N do) a vnútorný cyklus pohyb v nastavenom riadku t.j. zmenu stĺpca od 1 po M (for J:=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 MAT-1: Vytvorte program zabezpečujúci načítanie typu (rozmerov) a prvkov matice a jej zobrazenie na obrazovke monitora.

Analýza: Program obsahuje dva vstupy. V procedúre VSTUP treba zadať okrem rozmerov aj všetky prvky matice z klávesnice, v procedúre VSTUP_NAHODNE po zadaní rozmerov matice program náhodne vygeneruje prvky matice – celé čísla od 0 po 9. Keďže z procedúr vstupu sa majú novozískané hodnoty vyviesť, parametre musia byť nahradzované referenciou. Pri výstupe, hodnoty stačí len doviesť do procedúry, ide o náhradu formálnych parametrov hodnotou.

```
program MATICA_1;
uses  crt;
const MAXPP=10;
type  MATICA=array[1..MAXPP,1..MAXPP] of integer;
var   A:MATICA;
      N,M,I,J:integer;

procedure VSTUP(var N,M:integer;var A:MATICA);
  begin
  write('Zadaj rozmery matice <= ',MAXPP,': ');
  readln(N,M);
  for I:=1 to N do                { nastaví riadok }
    for J:=1 to M do              { nastaví stĺpec }
      begin
      write('Prvok ',I,',',J,': ');
      readln(A[I,J]);
      end;
  end;

procedure VSTUP_NAHODNE(var N,M:integer;var A:MATICA);
  begin
  write('Zadaj rozmery matice <= ',MAXPP,': ');
  readln(N,M);
  randomize;
  for I:=1 to N do
    for J:=1 to M do
      A[I,J]:=random(10);
  end;

procedure VYSTUP(N,M:integer;A:MATICA);
  begin
  for I:=1 to N do           { nastavuje riadok }
    begin
    for J:=1 to M do         { nastavuje stĺpec – pohyb v riadku }
      write(A[I,J]:4);       { hodnoty zobrazené v riadku }
    writeln;                 { presunutie „kurzora" na nový riadok }
    end;
  end;

BEGIN
clrscr;
writeln('Prvky matice nahodne (nie - N)?');
if upcase(readkey)='N'
  then VSTUP(N,M,A)
```

```
  else VSTUP_NAHODNE(N,M,A);
VYSTUP(N,M,A);
readln;
END.
```

Najprv si uvedieme niekoľko matematických príkladov na prácu s maticami.

Príklad MAT-2: Vytvorte program na vynásobenie matice celých čísel celým číslom k.

Analýza: Vynásobiť číselnú maticu číslom znamená vynásobiť každý jej prvok daným číslom, t.j. pre každé dovolené i, j: B[i,j] novej matice sa rovná k∗A[i,j].

```
procedure VYNASOB(N,M:integer;A:MATICA;var B:MATICA);
  var K:integer;
  begin
  write('Maticu vynasobit celym cislom: ');
  readln(K);
  for I:=1 to N do
    for J:=1 to M do
      B[I,J]:=K*A[I,J]
  end;

BEGIN
clrscr;
writeln('Prvky matice nahodne (nie - N)?');
if upcase(readkey)='N'
  then VSTUP(N,M,A)
  else VSTUP_NAHODNE(N,M,A);
VYSTUP(N,M,A);
VYNASOB(N,M,A,B);
VYSTUP(N,M,B);
readln;
END.
```

Príklad MAT-3: Vytvorte program na výpočet súčtu dvoch matíc rovnakých typov.

Analýza: Súčtom dvoch matíc A a B typu NxM je matica C typu NxM, kde C[i,j] = A[i,j] + B[i,j]. V procedúrach vstupu a výstupu je nevyhnutné použiť parametre, keďže procedúry chceme použiť pre matice A, B aj C!

```
program MATICA_3;
uses  crt;
const MAXPP=10;
type  MATICA=array[1..MAXPP,1..MAXPP] of integer;
var   A,B,C:MATICA;
      N,M,I,J:integer;

procedure ROZMERY(var N,M:integer);
  begin
  write('Zadaj rozmery matice <= ',MAXPP,': ');
  readln(N,M);
  end;

procedure VSTUP(N,M:integer;var A:MATICA);
...

procedure VSTUP_NAHODNE(N,M:integer;var A:MATICA);
...

procedure VYSTUP(N,M:integer;A:MATICA);
...

procedure SCITAT(N,M:integer;A,B:MATICA;var C:MATICA);
  begin
  for I:=1 to N do
    for J:=1 to M do
```

```
      C[I,J]:=A[I,J]+B[I,J]
  end;


BEGIN
randomize;
clrscr;
ROZMERY(N,M);
VSTUP_NAHODNE(N,M,A);
VSTUP_NAHODNE(N,M,B);
writeln('Prva matica: ');
VYSTUP(N,M,A);
writeln('Druha matica: ');
VYSTUP(N,M,B);
SCITAT(N,M,A,B,C);
writeln('Sucet prvej a druhej: ');
VYSTUP(N,M,C);
readln;
END.
```

Príklad MAT-4: Pre zasvätených, ktorí vedia, ako sa násobia matice, uvádzame bez komentára procedúru na súčin dvoch matíc. Matica A musí byť typu NxL, matice B typu LxM a výsledná matica C bude typu NxM.

```
procedure SUCIN(N,L,M:integer;A,B:MATICA;var C:MATICA);
  begin
  for I:=1 to N do
    for J:=1 to M do
       begin
       C[I,J]:=0;
       for K:=1 to L do
         C[I,J]:=C[I,J] + A[I,K] * B[K,J]
       end;
  end;
```

Príklad MAT-5: Vytvorte program, ktorý zistí, či zadaná štvorcová matica je symetrická.

Analýza: Štvorcová matica (počet riadkov sa rovná počtu stĺpcov) je symetrická, ak je symetrická podľa hlavnej diagonály (hlavná diagonála je tvorená prvkami A[1,1], A[2,2], A[3,3], ..., A[N,N]). Preto pre každý prvok matice musí platiť: A[I,J] = A[J,I] pre všetky dovolené hodnoty I a J.

Nie najefektívnejší, ale jednoduchší program znamená použitie cyklov for, ktoré "zistia", či sa prvky pod hlavnou diagonálou rovnajú príslušným prvkom nad hlavnou diagonálou (ak sa „dolný trojuholník rovná hornému, musí sa aj horný rovnať dolnému"). Keďže výsledkom podprogramu je hodnota true – ak je matica symetrická alebo hodnota false – ak nie je symetrická, môžeme použiť aj funkciu.

```
program MATICA_5;
uses  crt;
const MAXPP=10;
type  MATICA=array[1..MAXPP,1..MAXPP] of integer;
var   A:MATICA;
      N,I,J:integer;
procedure VSTUP(var N:integer;var A:MATICA);
....
procedure VYSTUP(N:integer;A:MATICA);
...
function SYMETRICKA(N:integer;A:MATICA):boolean;
  var SYM:boolean;
  begin
  SYM:=true;
  for I:=2 to N do
```

```
    for J:=1 to I-1 do
      if A[I,J]<>A[J,I] then SYM:=false; { SYM:=SYM and (A[I,J]=A[J,I]); }
  SYMETRICKA:=SYM
  end;
BEGIN
clrscr;
VSTUP(N,A);
writeln('Matica: ');
VYSTUP(N,A);
if SYMETRICKA(N,A)
  then writeln('je symetricka.')
  else writeln('nie je symetricka.');
readln;
END.
```

Príklad MAT-6: Vytvorte program na nájdenie najmenšej a najväčšej hodnoty v každom riadku matice.

Analýza: Každý riadok matice je jednorozmerné pole „doplnené“ o index riadku, v ktorom sa nachádza. Ak vieme hľadať minimum a maximum v jednorozmernom poli, stačí toto hľadanie vložiť do cyklu, ktorý bude meniť riadky od prvého po posledný.

```
program MATICA_6;
uses  crt;
const MAXPP=10;
type  MATICA=array[1..MAXPP,1..MAXPP] of integer;
      POLE=array[1..MAXPP] of integer;
var   A:MATICA;
      MIN,MAX:POLE;
      N,M,I,J:integer;

procedure VSTUP(var N,M:integer;var A:MATICA);
...
procedure VSTUP_NAHODNE(var N,M:integer;var A:MATICA);
...
procedure VYSTUP(N,M:integer;A:MATICA);
...

procedure MIN_MAX(N,M:integer;A:MATICA;var MIN,MAX:POLE);
  begin
  for I:=1 to N do
    begin
    MIN[I]:=A[I,1];
    MAX[I]:=A[I,1];
    for J:=2 to M do
      begin
      if A[I,J]<MIN[I] then MIN[I]:=A[I,J];
      if A[I,J]>MAX[I] then MAX[I]:=A[I,J]
      end;
    end;
  end;

procedure VYSTUPPOLA(N:integer;A:POLE);
  begin
  for I:=1 to N-1 do
    write(A[I],', ');
  writeln(A[N])
  end;

BEGIN
clrscr;
randomize;
writeln('Prvky matice nahodne (nie - N)?');
if upcase(readkey)='N'
  then VSTUP(N,M,A)
```

```
  else VSTUP_NAHODNE(N,M,A);
MIN_MAX(N,M,A,MIN,MAX);
writeln('V matici: ');
VYSTUP(N,M,A);
writeln('minima v riadkoch: ');
VYSTUPPOLA(N,MIN);
writeln('maxima v riadkoch: ');
VYSTUPPOLA(N,MAX);
readln;
END.
```

Príklad MAT-7: Vytvorte program na utriedenie prvkov v každom riadku matice.

Analýza: Aby sme zvládli tento príklad, musíme vedieť aspoň jeden triediaci algoritmus. Pri jednorozmernom poli sme sa zaoberali bubblesortom, ktorý použijeme aj teraz. Ako v predchádzajúcom príklade, aj tu treba len bubblesort „aplikovať“ na každý riadok matice, čo znamená vložiť bubblesort do cyklu na nastavenie riadku.

```
procedure UTRIED_V_RIADKU(N,M:integer;var A:MATICA);
  var PP:integer;
  procedure VYMENA(var X,Y:integer);
    var POM:integer;
    begin
    POM:=X; X:=Y; Y:=POM;
    end;
  begin
  for I:=1 to N do                  { nastaví riadok }
    for PP:=1 to M-1 do             { počet prechodov riadkom }
      for J:=1 to M-PP do           { pohyb v riadku }
        if A[I,J]>a[I,J+1] then VYMENA(A[I,J],A[I,J+1]);
  end;

BEGIN
clrscr;
writeln('Prvky matice nahodne (nie - N)?');
if upcase(readkey)='N'
  then VSTUP(N,M,A)
  else VSTUP_NAHODNE(N,M,A);
VYSTUP(N,M,A);
UTRIED_V_RIADKU(N,M,A);
writeln('Matica utriedenia v riadkoch: ');
VYSTUP(N,M,A);
readln;
END.
```

Ako treba upraviť program, aby sa zachovala aj pôvodná matica A? Podprogram, ktorý utriedi prvky matice v stĺpcoch, sa len málo líši od podprogramu UTRIED_V_RIADKU. Odlaďte ho.

Príklad MAT-8: Vytvorte program, ktorý utriedi riadky matice podľa hodnoty prvého prvku v riadku, t.j. podľa prvého stĺpca.

Analýza:

Napríklad v matici:	2	4	1	5
	3	5	2	7
	1	2	1	4

bude posledný riadok presunutý do prvého, prvý do stredného a stredný riadok do tretieho.

Utriedená matica:	1	2	1	4
	2	4	1	5

3 5 2 7

Keďže je potrebné vymieňať celé riadky, už v úseku definícií typ dvojrozmerné pole zadefinujeme ako pole riadkov. Tiež si treba uvedomiť, že všetky prvky v prvom stĺpci sú prístupné cez zápis A[I,1], kde I určuje riadok. No a dokonale treba ovládať nejaký triediaci algoritmus, napríklad bubblesort.

```
program MATICA_8;
uses  crt;
const MAXPP=10;
type  RIADOK=array[1..MAXPP] of integer;
      MATICA=array[1..MAXPP] of RIADOK;
var   A:MATICA;
      N,M,I,J:integer;

procedure VSTUP_NAHODNE(var N,M:integer;var A:MATICA);
...
procedure VYSTUP(N,M:integer;A:MATICA);
...
procedure UTRIED_PODLA_PRVEHO(N,M:integer;var A:MATICA);
  var R:RIADOK;
      PP:integer;
  procedure VYMENA(var X,Y:RIADOK);       { typ RIADOK! }
    var POM:RIADOK;
    begin
    POM:=X; X:=Y; Y:=POM
    end;
  begin
  for PP:=1 to N-1 do
    for I:=1 to N-PP do
      if A[I,1]>A[I+1,1] then VYMENA(A[I],A[I+1])
  end;

BEGIN
clrscr;
VSTUP_NAHODNE(N,M,A);
writeln('Povodna matica: ');
VYSTUP(N,M,A);
UTRIED_PODLA_PRVEHO(N,M,A);
writeln('Riadky utriedene podla prveho prvku v riadku: ');
VYSTUP(N,M,A);
readln;
END.
```

Na záver „matematických“ príkladov s maticami uvedieme „hviezdičkový“ príklad.

Príklad MAT-9*: Vytvorte program, ktorý „rozvinie“ riadky a stĺpce matice do jednorozmerného poľa v smere chodu hodinových ručičiek.

Analýza:

Napríklad maticu: 2 4 1 5

3 5 2 7

1 2 1 4

rozvinie program do poľa: 2 4 1 5 7 4 1 2 1 3 5 2

```
program MAT-9;
uses  crt;
const MAXPP=10;
type  MATICA=array[0..MAXPP,0..MAXPP] of integer;
      POLE=array[0..MAXPP*MAXPP] of integer;
var   A:MATICA;
      B:POLE;
      i,j,k,n,m,l,p,h,d:integer;
```

```
procedure VSTUP;
...
procedure VYSTUP;
...
procedure VYTVOR_POLE;
 procedure vpravo;
   begin
   for j:=l to p do begin k:=k+1; B[k]:=A[h,j]; end;
   end;
 procedure dole;
   begin
   for i:=h to d do begin k:=k+1; B[k]:=A[i,p]; end;
   end;
 procedure vlavo;
   begin
   for j:=p downto l do begin k:=k+1; B[k]:=A[d,j]; end;
   end;
 procedure hore;
   begin
   for i:=d downto h do begin k:=k+1; B[k]:=A[i,l]; end;
   end;
 begin
 k:=0; l:=1; p:=m; h:=1; d:=n;
 repeat
   if k<n*m then vpravo; h:=h+1;
   if k<n*m then dole;   p:=p-1;
   if k<n*m then vlavo;  d:=d-1;
   if k<n*m then hore;   l:=l+1;
 until k=n*m;
 end;
procedure VYSTUP_POLA;
 begin
 writeln('"Rozvinuta" matica:');
 for k:=1 to n*m do write(B[k]:4);
 writeln;
 end;
BEGIN
clrscr;
VSTUP;
VYSTUP;
VYTVOR_POLE;
VYSTUP_POLA;
readln
END.
```

Mnohí z nás poznajú hru ŽIVOT, v ktorej sa simuluje život jednobunkových organizmov v rovine. Po vložení počiatočnej generácie program „počíta“ ďalšie generácie. Organizmus prežije do ďalšej generácie, ak ma 2 alebo 3 susedov, ináč zomrie. Živý organizmus vznikne, ak má práve 3 susedov.

Príklad MAT-10: Vytvorte program simulujúci život jednobunkových organizmov za vyššie uvedených podmienok.

Analýza: Potrebné sú dve matice. V jednej „prebieha život“, druhá je pomocná pri výpočte novej generácie. Pre získanie nových poznatkov sme namiesto čísel 0 – prázdna pozícia a 1 – živý organizmus použili programátorom definovaný typ s hodnotami nezivý (ordinálne číslo 0) a zivy (ordinálne číslo 1). Program ponechávame na samoštúdium.

```
program MAT-10;
uses  crt;
const okraj=24;
      plot =23;
```

```
type  torganizmu=(nezivy,zivy);     { programátorom vymenovaný typ }
      tsuradnic=0..okraj;
      tpola=array[tsuradnic,tsuradnic] of torganizmu;
var   a,pom:tpola;
      q:char;

procedure vycisti_pole;
  var i,j:tsuradnic;
  begin
  clrscr;
  for i:=0 to okraj do
    for j:=0 to okraj do
      begin
      a[i,j]:=nezivy;
      pom[i,j]:=nezivy;
      end;
  end;
procedure vystup;
  var i,j:tsuradnic;
  begin
  clrscr;
  for i:=1 to plot do
    begin
    for j:=1 to plot do
      if a[i,j]=zivy
        then write('*')
        else write(' ');
    writeln
    end;
  end;

procedure vstup;
  var i,j:integer;
  begin
  repeat
    write('Zadaj suradnice x,y ziveho organizmu ( KONIEC -> 0 ) ');
    read(i);
    if i>0
      then begin
           readln(j);
           if (i<okraj)and(j>0)and(j<okraj)
             then a[i,j]:=zivy
             else begin writeln('Chybne suradnice');delay(2000) end;
           vystup
           end
  until i=0;
  end;

procedure nova_generacia;
  var i,j:tsuradnic; pocet:0..8;
  begin
  for i:=1 to plot do
    for j:=1 to plot do
      begin
      pocet:=ord(a[i-1,j-1])+      { počíta počet susedov okolo a[i,j] }
             ord(a[i-1,j])+
             ord(a[i-1,j+1])+
             ord(a[i,j-1])+
             ord(a[i,j+1])+
             ord(a[i+1,j-1])+
             ord(a[i+1,j])+
             ord(a[i+1,j+1]);
      if (pocet=3)or((pocet=2)and(a[i,j]=zivy))
        then pom[i,j]:=zivy
        else pom[i,j]:=nezivy;
      end;
```

```
  a:=pom;
  vystup;
  end;
BEGIN
vycisti_pole;
repeat
  vstup;
  repeat
    nova_generacia;
    write('Chces dalsiu generaciu (N) ? ');q:=upcase(readkey);
  until q='N';
  gotoxy(1,24);
  write('Chces doplnit zadanie (A) ? ');q:=upcase(readkey);
until q<>'A';
END.
```

Neriešené úlohy na dvojrozmerné pole:

11. Vytvorte program na výpočet súčtu všetkých prvkov číselnej matice.
12. Vytvorte program, ktorý sčíta všetky prvky na hlavnej diagonále (ide z ľavého horného rohu matice do pravého dolného rohu) štvorcovej matice.
13. Vytvorte program, ktorý sčíta všetky prvky na vedľajšej diagonále (ide z pravého horného rohu matice do ľavého dolného rohu) štvorcovej matice.
14. Vytvorte program na zistenie počtu núl (zadaného čísla, kladných čísel, deliteľných zadaným číslom, zo zadaného intervalu a pod.) v zadanej číselnej matici.
15. Vytvorte program na nájdenie najmenšieho (najväčšieho) prvku matice a jeho výmenu s prvým (posledným) prvkom matice.
16. Vytvorte program na nájdenie najmenšieho a najväčšieho prvku každého riadku matice a jeho výmenu s prvým a posledným prvkom v danom riadku matice.
17. Vytvorte program na nájdenie najmenšieho a najväčšieho prvku v každom stĺpci matice a jeho výmenu s prvým a posledným prvkom v danom stĺpci.
18. Vytvorte program na nájdenie zadanej hodnoty v matici a jej nahradenie novou hodnotou.
19. Vytvorte program na zistenie, či sa zadaný znak nachádza v matici znakov a koľkokrát sa nachádza.
20. Vytvorte program na zistenie miesta výskytu (všetkých, prvého, posledného) prvku so zadanou vlastnosťou v matici.
21. Vytvorte program, ktorý vo štvorcovej matici vymení prvky podľa hlavnej diagonály.
22. Vytvorte program, ktorý v matici vymení prvý stĺpec (riadok) s posledným, druhý s predposledným, atď.
23. Vytvorte program, ktorý zistí, či zadaná štvorcová matica je diagonálna (okrem hlavnej diagonály a[i,j] = 0).
24. Vytvorte program na otočenie štvorcovej matice o 90^0.
25. Vytvorte program, ktorý od miesta „čľupnutia kameňa do matice“ zvýši pôvodne nulové hodnoty matice s narastajúcou vzdialenosťou od miesta „čľupnutia“ o 1.
26. Vytvorte program, ktorý pri pohybe v matici bude meniť hodnoty prechádzaných prvkov.
27. Vytvorte program, ktorý pri pohybe v matici bude meniť hodnoty v okolí prechádzaných prvkov.

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 KARTA = 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;
               Moze : boolean
             end;

type BOD = record
             x,
             y : real;
             farba : 0..14;
             blik : boolean
           end;

type ZLOZKA = record
                HODNOTA : string;
                POCET   : byte
              end;

var KC : record Re, Im: real end;        { deklarovaná premenná typu record }
```

Ak p je premenná typu záznam, k jej položke s názvom pi sa dostaneme zápisom: p.pi. Napríklad, ak K je premenná typu KARTA, možno použiť zápisy: K.OsCislo, K.Meno, K.DatNar.Den, K.DatNar.Rok, K.Adresa.Obec, K.Priemer[1] a pod.

Príklad ZAZ-1: Vytvorte program na určenie vzájomnej polohy bodu a kružnice v rovine s využitím údajového typu záznam.

Analýza: Bod napr. B je určený v rovine dvoma súradnicami B_x a B_y. Kružnica je určená stredom S so súradnicami S_x a S_y a polomerom r. Vzdialenosť medzi bodom B a stredom kružnice k, bodom S, možno

vypočítať pomocou Pythagorovej vety. Bod B leží vo vnútornej oblasti kružnice k (S;r), ak jeho vzdialenosť od stredu kružnice k je menšia ako polomer r. Ak vzdialenosť /BS/ = r, bod B leží na kružnici a ak je táto vzdialenosť väčšia ako polomer r, bod leží mimo oblasť kružnice k.

```
program ZAZ_1;
uses crt;
type BOD=record
           x,y:real;
         end;
     KRUZNICA=record
                s:BOD;
                r:real;
              end;
var A:BOD;
    k:KRUZNICA;
    vzd :real;

procedure VSTUP;
  begin
  writeln('Zadaj suradnice bodu ');
  readln(A.x,A.y);
  writeln('Zadaj stred kruznice a polomer');
  readln(k.s.x,k.s.y,k.r);
  end;

function VZDIALENOST:real;
  begin
  vzdialenost:=sqrt(sqr(A.x-k.s.x)+sqr(A.y-k.s.y));
  end;

BEGIN
clrscr;
VSTUP;
vzd:=VZDIALENOST;
if vzd>k.r
  then writeln('Bod lezi vo vonkajsej oblasti kruznice.')
  else if vzd=k.r
         then writeln('Bod lezi na kruznici.')
         else writeln('Bod lezi vo vnutornej oblasti kruznice.');
readln;
END.
```

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.položka skrátiť na zápis položka.

Napríklad: with k do begin writeln('Zadaj stred kruznice a polomer'); readln(s.x , s.y , r) end;

with Z , DatNar do begin OsCislo:=99; Den:=1; Mes:=1; Rok:=2000 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 ZAZ-2: Vytvorte jednoduchú kartotéku na evidenciu osôb (evidenčné číslo, meno a adresa: miesto, ulica).

Analýza: Na precvičenie údajového typu record vytvoríme jednoduchú kartotéku. Jednotlivé karty budú uložené v poli, čo v skutočnosti nikdy nebýva, lebo ukončením programu by sme prišli o všetky údaje v kartách. Na zapamätanie údajov aj po vypnutí počítača slúžia vonkajšie pamäte (najmä pevný disk). Ukladať údaje na vonkajšie pamäte umožňuje údajový typ súbor, o ktorom si povieme už v nasledujúcej kapitole.

```
program ZAZ_2;
uses  crt;
const pk=10;        { pocet kariet }
      dmena=15;
      dadr=25;
type  karta=record
           ec:byte;
           meno:string[dmena];
           adr:record
               miesto,
               ulica:string[dadr];
               end;
            end;
      kartoteka=array[1..pk] of karta;
var   k:kartoteka;
      apk,i:byte;
      q:char;

procedure vstup_karta;
  begin
  clrscr;
  apk:=apk+1;
  with k[apk],adr do
    begin
    ec:=apk;
    gotoxy(1,8);
    writeln('Evidencne cislo : ':40,ec);
    writeln;
    write('Meno a priezvisko : ':40);readln(meno);
    writeln;
    write('  Adresa - miesto : ':40);readln(miesto);
    writeln;
    write('         -  ulica : ':40);readln(ulica);
    end;
  end;

procedure vystup_kartoteka;
  procedure vystup_karta(i:byte);
    begin
    clrscr;
    writeln('Vystup');
    gotoxy(1,8);
    with k[i],adr do
      begin
      writeln('  Evidencne cislo : ':40,ec);
      writeln;
      writeln('Meno a priezvisko : ':40,meno);
      writeln;
      writeln('  Adresa - miesto : ':40,miesto);
      writeln;
      writeln('          - ulica : ':40,ulica);
      end;
    readln;
    end;
```

```
  begin     {procedura vystup_kartoteka }
  for i:=1 to apk do vystup_karta(i);
  end;

procedure menu;
  begin
  repeat
    clrscr;
    gotoxy(1,10);
    writeln('VLOZIT KARTU..................... +':55);
    writeln;
    writeln('VYPISAT KARTOTEKU.............. Enter':55);
    writeln;
    writeln('KONIEC........................... K':55);
    q:=upcase(readkey);
    case q of
      '+'     : vstup_karta;
      chr(13) : vystup_kartoteka;
    end;
  until q='K';
  end;

BEGIN
apk:=0;
vstup_karta;
menu;
END.
```

Variantný záznam

Pri používaní údajového typu záznam môže vyvstať požiadavka na návrh takej štruktúry záznamu, v ktorej sa na daný prvok nemusí vzťahovať celý zoznam položiek uvedených v zázname, ale iba jeho určitá časť. Takto môže vzniknúť niekoľko alternatívnych typov – variant - jedného typu záznam. Takúto situáciu nám umožňuje riešiť tzv. variantný záznam, ktorý sa skladá z pevnej časti a z variantnej časti.

Variantný záznam má tvar:

```
record
    pevná časť;
    case rozlišovacia_položka : rozlišovací_typ_variantnej_časti of
       rozlišovacia_konštanta1 : ( položka1 : typ1;
                                   položka2 : typ2;
                                   ...          );
       rozlišovacia_konštanta2 : (položka1 : typ1;
                                   položka2 : typ2;
                                   ...          );
       ...
end
```

kde rozlišovací typ variantnej časti môže byť len ordinálny typ.

Napríklad:

```
type    TypPohlavia =(MUZ,ZENA);
        TypMiery=(PRSIA,PAS,BOKY);
```

```
OSOBA = record
            Priezvisko,
            Meno        : string[15];
            DatNar      : string[10];
            case Pohlavie : TypPohlavia of
               MUZ : (Hmotnost : real;
                      Brada    : boolean);
               ZENA : (Miery : array[TypMiery] of  byte)
          end;
```

Príklad ZAZ-3: Program ZAZ-2 doplňte o variantný záznam evidujúci u mužov fúzy a u žien počet detí.

Analýza: Aj keď je program veľmi podobný predchádzajúcemu, uvádzame ho podrobne až po celkom spoločnú časť.

```
program ZAZ_3;
uses  crt;
const pk=10;            { pocet kariet }
      dmena=15;
      dadr=25;
type  karta=record
           ec:byte;
           meno:string[dmena];
           adr:record
               miesto,
               ulica:string[dadr];
               end;
           case pohl:(muz,zena) of
             muz:(fuzy:boolean);
             zena:(deti:integer);
           end;
     kartoteka=array[1..pk] of karta;
var  k:kartoteka;
     apk,i:byte;
     q:char;

procedure vstup_karta;
  var odp:char;
  begin
  clrscr;
  apk:=apk+1;
  with k[apk],adr do
    begin
    ec:=apk;
    gotoxy(1,8);
    writeln('Evidencne cislo : ':40,ec);
    writeln;
    write('Meno a priezvisko : ':40);readln(meno);
    writeln;
    write('  Adresa - miesto : ':40);readln(miesto);
    writeln;
    write('         -  ulica : ':40);readln(ulica);
    writeln;
    write('  Muz (ano - A) ? : ':40);
    if upcase(readkey)='A'
      then begin
           writeln('ano');
           pohl:=muz;
           writeln;
           write('Ma fuzy (ano - A) ? : ':40);
           fuzy:=upcase(readkey)='A';
```

```
           if fuzy then writeln('ano') else writeln('nie')
           end
      else begin
           writeln('nie');
           pohl:=zena;
           writeln;
           write('Pocet deti : ':40);readln(deti);
           end;
    delay(500)
    end;
  end;

procedure vystup_kartoteka;
  procedure vystup_karta(i:byte);
    begin
    clrscr;
    writeln('Vystup');
    gotoxy(1,8);
    with k[i],adr do
      begin
      writeln('  Evidencne cislo : ':40,ec);
      writeln;
      writeln('Meno a priezvisko : ':40,meno);
      writeln;
      writeln('  Adresa - miesto : ':40,miesto);
      writeln;
      writeln('           - ulica : ':40,ulica);
      writeln;
      if pohl=muz
        then if fuzy
               then writeln('Fuzy : ano':43)
               else writeln('Fuzy : nie':43)
        else writeln('Pocet deti : ':40,deti);
      end;
    readln;
    end;
  begin
  for i:=1 to apk do vystup_karta(i);
  end;
...
```

Na záver údajového typu záznam uvádzame „červíka Wurmiho“ (Príklad POLE-13) doplneného o jeho rast, pridanie článku tela, pri každom zožratí potravy (listu).

Príklad ZAZ-4: Program POLE-13 doplňte o „telo“ červíka Wurmiho, ktoré narastie o jeden článok pri každom zožratí potravy.

Analýza: Program POLE-13 treba doplniť o proces zapamätania si polohy článkov tela Wurmiho. Z viacerých možností sme vybrali na zapamätanie polohy článkov tela jednorozmerné pole TELO, v ktorom sú ako položky recordu zapamätané x-ová a y-nová súradnica 1., 2. atď. článku tela.

Program ponechávame na samoštúdium, podprogramy totožné s programom POLE-13 neuvádzame.

```
program ZAZ_4;
uses  crt,dos;
const pocet=5;
...
var   body,dlzka:integer;
      px,py,jx,jy:array[1..pocet] of integer;
      telo:array[0..pocet] of record x,y:integer end;

procedure skry_kurzor;
...
```

```
procedure ohrada;
...
procedure generuj_potravu_jed;
...
procedure vykresli_potravu_jed;
...
procedure inicializacia;
  var limit:integer;
  begin
  randomize;
  generuj_potravu_jed;
  skry_kurzor;
  clrscr;
  ohrada;
  vykresli_potravu_jed;
  dlzka:=0; telo[0].x:=40; telo[0].y:=12;
  gotoxy(telo[0].x,telo[0].y); write(tvaricka);
  body:=0;
  end;

function jed:boolean;
  var i:integer;
  begin
  i:=0;
  repeat
    inc(i)
  until ((jx[i]=telo[0].x) and (jy[i]=telo[0].y)) or (i=pocet);
  jed:=(jx[i]=telo[0].x) and (jy[i]=telo[0].y)
  end;

function potrava:boolean;
  var i:integer;
  begin
  i:=0;
  repeat
    inc(i)
  until ((px[i]=telo[0].x) and (py[i]=telo[0].y)) or (i=pocet);
  if (px[i]=telo[0].x) and (py[i]=telo[0].y)
    then begin potrava:=true; inc(body); px[i]:=0; py[i]:=0 end
    else potrava:=false;
  gotoxy(3,2); write(body);
  end;

procedure pohyb;
  var stlznak:char;
      i:integer;
  begin
  repeat
    if keypressed then stlznak:=readkey;
    if potrava then inc(dlzka);
    gotoxy(telo[dlzka].x,telo[dlzka].y); write(' ');
    for i:=dlzka downto 1 do
      begin
      telo[i]:=telo[i-1];
      gotoxy(telo[i].x,telo[i].y); write('0');
      end;
    case stlznak of
      ESC:halt;
      sipkaVPRAVO:inc(telo[0].x);
      sipkaVLAVO :dec(telo[0].x);
      sipkaHORE  :dec(telo[0].y);
      sipkaDOLE  :inc(telo[0].y);
      end;
    gotoxy(telo[0].x,telo[0].y); write(tvaricka);
    delay(100);
  until  (telo[0].x=1)or(telo[0].x=80)or(telo[0].y=1)or(telo[0].y=24)or  jed  or
```

```
(body=pocet);
  if jed
    then begin gotoxy(33,12); write(' J E D ! ! ! ') end
    else if body=pocet
           then begin gotoxy(20,12); write('V Y B O R N E !!! VYHRAL SI!!!') end
           else begin gotoxy(25,12); write('B U M ! ! !  O H R A D A ! ! !') end
  end;

BEGIN
inicializacia;
pohyb;
readln
END.
```

Neriešené úlohy na záznam:

5. Navrhnite úsek definícií a deklarácií pre evidenčnú kartu knihy a čitateľa v knižnici.
6. Vytvorte program na evidenciu kníh (čitateľov) knižnice.
7. Program z úlohy 6 doplňte o vyradenie čitateľa (knihy) z evidencie.
8. Program z úlohy 6 doplňte o vyhľadanie knihy (čitateľa) po zadaní mena.
9. Vytvorte program na evidenciu motorových vozidiel.
10. Program z úlohy 9 doplňte o variantný záznam, ktorý podľa typu vozidla bude evidovať napr. pri osobnom aute počet miest a nosnosť, pri autobuse počet miest, klimatizáciu a pod., pri nákladnom aute nosnosť, nákladnú plošinu a pod.
11. Vytvorte program pracujúci ako elektronický dvojjazyčný (viacjazyčný) slovník.
12. Vytvorte program, ktorý po zadaní textu zistí počet výskytov jednotlivých slov v texte.
13. Pre typ RETAZEC = record

    POCETZN : 0..255;
    ZNAKY : array[1..255] of char

    end;

    napíšte procedúru, ktorá spojí dva reťazce do tretieho.
14. K údajovému typu z príkladu 13 napíšte funkciu, ktorá zistí, či jeden reťazec je obsiahnutý v druhom (je podreťazcom).

Údajový typ súbor

Ak potrebujeme uchovať údaje aj po vypnutí počítača resp. po ukončení programu, musíme ich uložiť na vonkajšie pamäťové médium – disk, disketu a pod. Zapisovať a čítať údaje z vonkajších pamäťových médií nám v TP umožňuje údajový typ súbor.

Súbor je štruktúrovaný údajový typ, ktorý sa skladá z teoreticky neobmedzeného počtu zložiek, všetky rovnakého typu. Prakticky je počet zložiek súboru obmedzený kapacitou vonkajšej pamäte.

Definícia typu súbor má tvar: *type* mt = *file of* tz

kde mt je meno typu a tz je typ zložky, ktorý nesmie byť typ súbor ani typ obsahujúci ako zložku typ súbor.

Napríklad:	type CELE = file of integer;	typ súbor obsahujúci celé čísla
	type KARTOTEKA = file of KARTA;	typ umožňujúci uloženie kartotéky do súboru
	type BODY = file of record x,y,z:real end;	typ súbor obsahujúci trojice x,y,z
	var F : file of string;	premenná typu súbor obsahujúci reťazce

Súbory môžu byť vnútorné (pracovné) a vonkajšie. Pracovné súbory si môže vytvoriť program pri riešení pamäťovo náročnejších úloh, TP im dáva príponu $$$. Po skončení programu sa môžu z vonkajšej pamäte odstrániť. Ďalej sa budeme zaoberať len vonkajšími súbormi, ktoré existujú aj po skončení programu a umožňujú trvalé uloženie dát.

Základné príkazy pre prácu so súbormi umožňujú len sekvenčné spracovanie súboru, t.j. vytváranie aj čítanie súboru len zložku po zložke (od prvej k druhej atď.). V každom okamihu spracovania súboru je prístupná len jedna zložka súboru, na ktorú „ukazuje“ prístupová premenná súboru. Ak F je premenná typu súbor, k nej prislúchajúca prístupová premenná má označenie F^ a vznikne automaticky pri deklarácii premennej typu súbor.

Deklaráciou premennej typu súbor vznikne len abstraktný vonkajší súbor, sú určené len jeho logické vlastnosti. K spojeniu logického súboru s fyzickým – skutočne existujúcim na konkrétnom vonkajšom pamäťovom médiu, dôjde príkazom *assign* (logické_meno , fyzické_meno) kde logické meno je premenná typu súbor a fyzické meno je reťazec obsahujúci názov súboru na disku, prípadne doplnený aj o cestu. Napríklad: assign(F,´CITATEL.DAT´), assign(SUB,´C:\TP\CELE1.DBF´).

Zápis údajov do súboru (napríklad F):

Zápis údajov do súboru (po stotožnení logického a fyzického mena súboru!) prebieha v dvoch fázach:

1. súbor sa pripraví na zápis príkazom *rewrite*(F). Vytvorí sa prázdny súbor (prázdna hodnota súboru), prístupová premenná F^ sa nastaví na „koniec“ súboru. Ak súbor už obsahuje nejaké zložky, budú odstránené!
2. vlastný zápis do súboru sa realizuje príkazom *write*(F,V), kde V je výraz rovnakého typu, ako je typ zložky súboru. Po vyhodnotení výrazu V, získaní konkrétnej hodnoty, dôjde k jej zápisu do súboru F. Bod 2 možno ľubovoľný počet krát opakovať.

Čítanie údajov zo súboru (napríklad F):

Aj sekvenčné čítanie údajov zo súboru prebieha v dvoch fázach:

1. súbor sa pripraví na čítanie príkazom *reset*(F). Príkaz spôsobí nastavenie prístupovej premennej F^ na začiatok súboru.
2. vlastné čítanie zo súboru sa realizuje príkazom *read*(F,X), kde X je premenná rovnakého typu ako je typ zložky súboru. Po vykonaní príkazu je v premennej X hodnota načítaná zo súboru. Čítanie zo súboru možno opakovať, až kým prístupová premenná nie je za poslednou položkou súboru.

Na zistenie pozície prístupovej premennej v súbore slúži funkcia eof (end of file), ktorá má hodnotu true, ak prístupová premenná je na konci súboru (hodnota F^ nie je definovaná), inak má hodnotu false. To možno výhodne využiť na načítanie všetkých zložiek súboru schémou:

```
reset(F);
while not eof(F) do      { pokiaľ nie je koniec súboru opakuj }
   begin
   read(F,X);            { načíta hodnotu zo súboru do premennej X }
   write(X)              { zobrazí načítanú hodnotu na obrazovke }
   end;
```

Ukončenie práce so súborom (napríklad F):

Po ukončení práce so súborom sa súbor musí zavrieť príkazom *close*(F).

Príklad SUB-1: Vytvorte program na vytvorenie súboru a zobrazenie zložiek súboru obsahujúceho čísla typu integer.

Analýza: Program obsahuje dve samostatné procedúry. Procedúra VSTUP umožňuje vytvoriť, po zadaní mena fyzického súboru, súbor obsahujúci celé čísla a procedúra VYSTUP zobrazuje zložky – celé čísla, zvoleného súboru. Program je pre pohodlie doplnený o menu.

```
program SUB_1;
uses  crt;
var   F:file of integer;
      ODP:char;

procedure VSTUP;
  var FM:string[12];
      X:integer;
  begin
  write('Meno suboru na disku: ');
  readln(FM);
  assign(F,FM);
  rewrite(F);
  repeat
    write('Vlozit cislo: ');
    readln(X);
    write(F,X);
    writeln('Vlozit dalsie cislo (nie - N) ?':52);
  until upcase(readkey)='N';
  close(F);
  end;

procedure VYSTUP;
  var FM:string[12];
      X:integer;
  begin
  write('Meno suboru na disku: ');
  readln(FM);
  assign(F,FM);
  reset(f);
  while not eof(F) do
```

```
    begin
    read(F,X);
    write(X:4);
    end;
  close(F);
  readln
  end;

BEGIN
repeat
  clrscr;
  gotoxy(1,10);
  writeln('Vytvorit subor ............... V':50);
  writeln;
  writeln('Zobrazit subor ............... Z':50);
  writeln;
  writeln('Koniec ....................... K':50);
  repeat ODP:=upcase(readkey); until (ODP='V')or(ODP='Z')or(ODP='K');
  clrscr;
  case ODP of
    'V' : VSTUP;
    'Z' : VYSTUP;
  end
until ODP='K'
END.
```

TP má pre prácu so súbormi ďalšie procedúry a funkcie. Funkcia *FileSize*(F) vracia číslo udávajúce počet zložiek súboru F. Procedúra *Seek*(F,n) nastaví prístupovú premennú za n-tú zložku súboru F. Preto príkaz Seek(F,FileSize(F)) nastaví prístupovú premennú F^ za poslednú zložku súboru a umožní nám napríklad zápis na koniec súboru F. Príkaz Rename(F,nové_fyz._meno) premenuje externý súbor (po príkaze assign). Príkaz Erase(F) zmaže externý súbor (po príkaze assign).
Uvedené príkazy si precvičíme v nasledujúcom príklade.

Príklad SUB-2: Program SUB-1 (predchádzajúci príklad) doplňte o podprogramy na vytvorenie prázdneho súboru, nazistenie počtu zložiek súboru, na pridanie zložky na koniec súboru a na odstránenie zvolenej zložky zo súboru.
Analýza: Program SUB-2 je určený na ozrejmenie základných techník práce s údajovým typom súbor. Ošetrili sme otvorenie súboru funkciou IOResult, ak súbor so zadaným menom na disku neexistuje, vypíše sa správa: Súbor neexistuje! Najkomplikovanejšie je odstránenie zvolenej zložky zo súboru, kde sa musí použiť aj pomocný súbor (zo zdrojového súboru sa číta a do pomocného zapisuje), ktorý po zapísaní zvolených zložiek je premenovaný na pôvodný zdrojový súbor (pôvodný súbor musí byť najprv vymazaný).

```
program SUB_2;
uses  crt;
var   F:file of integer;
      FM:string[12];
      ODP:char;
      CHYBA:boolean;

procedure PRAZDNY;
  var FM:string[12];
  begin
  write('Meno suboru na disku: ');
  readln(FM);
  assign(F,FM);
  rewrite(F);
  close(F)
```

```
  end;

procedure VSTUP;
  var FM:string[12];
      X:integer;
  begin
  write('Meno suboru na disku: ');
  readln(FM);
  assign(F,FM);
  rewrite(F);
  repeat
    write('Vlozit cislo: ');
    readln(X);
    write(F,X);
    writeln('Vlozit dalsie cislo (nie - N) ?':52);
  until upcase(readkey)='N';
  close(F);
  end;

procedure VYSTUP;
  var FM:string[12];
      X:integer;
  begin
  write('Meno suboru na disku: ');
  readln(FM);
  assign(F,FM);
  {$I-} reset(F); {$I+}
  CHYBA:=IOResult<>0;
  if CHYBA
    then writeln('Subor neexistuje!')
    else begin
         if filesize(F)=0 then writeln('Subor je prazdny!');
         while not eof(F) do
           begin
           read(F,X);
           write(X:4)
           end;
         close(F)
         end;
  readln
  end;

procedure POCET;
  var FM:string[12];
  begin
  write('Meno suboru na disku: ');
  readln(FM);
  assign(F,FM);
  {$I-} reset(F); {$I+}
  CHYBA:=IOResult<>0;
  if CHYBA
    then writeln('Subor neexistuje!')
    else begin
         writeln('Pocet zloziek: ',filesize(F));
         close(F);
         end;
  readln
  end;

procedure PRIDAT_NA_KONIEC;
  var FM:string[12];
      X:integer;
  begin
  write('Meno suboru na disku: ');
  readln(FM);
  assign(F,FM);
```

```
  {$I-} reset(F); {$I+}
  CHYBA:=IOResult<>0;
  if CHYBA
    then writeln('Subor neexistuje!')
    else begin
         write('Pridat hodnotu: ');
         readln(X);
         seek(F,filesize(F));
         write(F,X);
         close(F);
         end
  end;

procedure ODSTRANIT;
  var FPom:file of integer;
      FM:string[12];
      X,HlHodnota:integer;
  begin
  write('Meno suboru na disku: ');
  readln(FM);
  assign(F,FM);
  {$I-} reset(F); {$I+}
  CHYBA:=IOResult<>0;
  if CHYBA
    then writeln('Subor neexistuje!')
    else begin
         assign(FPom,'POMOCNY.$$$');
         rewrite(FPom);
         write('Odstranit hodnotu: ');
         readln(HlHodnota);
         while not eof(F) do
           begin
           read(F,X);
           if X<>HlHodnota then write(FPom,X)
           end;
         close(F); close(FPom);
         assign(F,FM); erase(F);
         assign(FPom,'POMOCNY.$$$'); rename(FPom,FM)
         end
  end;

BEGIN
repeat
  clrscr;
  gotoxy(1,6);
  writeln('Vytvorit prazdny subor ....... S':50);
  writeln;
  writeln('Vytvorit neprazdny subor ..... V':50);
  writeln;
  writeln('Zobrazit subor ............... Z':50);
  writeln;
  writeln('Pocet zloziek suboru ......... P':50);
  writeln;
  writeln('Pridat zlozku na koniec ...... C':50);
  writeln;
  writeln('Odstranit zlozku zo suboru ... O':50);
  writeln;
  writeln('Koniec ....................... K':50);
  repeat ODP:=upcase(readkey); until ODP in ['S','V','Z','P','C','O','K'];
  clrscr;
  case ODP of
    'S' : PRAZDNY;
    'V' : VSTUP;
    'Z' : VYSTUP;
    'P' : POCET;
    'C' : PRIDAT_NA_KONIEC;
```

```
    'O' : ODSTRANIT;
  end
until ODP='K'
END.
```

Príklad SUB-3: Kartotéku v programe ZAZ_2 pracujúcu s poľom prerobte na „skutočnú“ kartotéku pracujúcu so súborom.

Analýzy: Program využíva len už vyššie použité procedúry a funkcie a preto ho nebudeme podrobnejšie opisovať. Odporúčame premenovať si súbor ZAZ-2 na SUB-3 a potom urobiť príslušné zmeny.

```
program SUB_3;
uses  crt;
const dmena=15;
      dadr=25;
type  karta=record
           ec:integer;
           meno:string[dmena];
           adr:record
               miesto,
               ulica:string[dadr];
               end;
            end;
      kartoteka=file of karta;
var   f:kartoteka;
      k:karta;
      q:char;

procedure vstup_karta;
  var pocet:integer;
  begin
  reset(f);
  pocet:=filesize(f);
  seek(f,pocet);
  with k,adr do
    begin
    inc(pocet);
    ec:=pocet;
    gotoxy(1,8);
    writeln('Evidencne cislo : ':40,ec);
    writeln;
    write('Meno a priezvisko : ':40);readln(meno);
    writeln;
    write('  Adresa - miesto : ':40);readln(miesto);
    writeln;
    write('          -  ulica : ':40);readln(ulica);
    end;
  write(f,k);
end;

procedure vystup_karta;
   begin
   clrscr;
   gotoxy(1,8);
   with k,adr do
     begin
     writeln('  Evidencne cislo : ':40,ec);
     writeln;
     writeln('Meno a priezvisko : ':40,meno);
     writeln;
     writeln('  Adresa - miesto : ':40,miesto);
     writeln;
     writeln('          - ulica : ':40,ulica);
     end;
   readln;
```

```
   end;

procedure vystup_kartoteka;
  begin
  reset(f);
  while not eof(f) do
    begin
    read(f,k);
    vystup_karta;
    end;
  end;

procedure najdi;
  var hlmeno:string[dmena];
  begin
  gotoxy(20,12);
  write('Zadaj meno hladaneho: ');
  readln(hlmeno);
  reset(f);
  while not eof(f) do
    begin
    read(f,k);
    if k.meno=hlmeno then vystup_karta
    end
  end;

procedure odstran;
  var hlmeno:string[dmena];
      fn:kartoteka;
  begin
  gotoxy(20,12);
  write('Zadaj meno na odstranenie: ');
  readln(hlmeno);
  assign(fn,'pracov.pom');
  reset(f);rewrite(fn);
  while not eof(f) do
    begin
    read(f,k);
    if k.meno<>hlmeno then write(fn,k);
    end;
  close(f); close(fn);
  assign(f,'pracov.dat');  erase(f);
  assign(fn,'pracov.pom'); rename(fn,'pracov.dat');
  assign(f,'pracov.dat');
end;

procedure pocet;
  begin
  reset(f);
  gotoxy(30,12);
  writeln('Pocet pracovnikov: ',filesize(f));
  readln;
  end;

procedure menu;
  begin
  repeat
    clrscr;
    gotoxy(1,6);
    writeln('           ZALOZIT KARTU..................... 1');
    writeln;
    writeln('           NAJST PODLA MENA.................. 2');
    writeln;
    writeln('           ODSTRANIT S MENOM................. 3');
    writeln;
    writeln('           POCET PRACOVNIKOV................. 4');
```

```
    writeln;
    writeln('            ZOZNAM PRACOVNIKOV.................. 5');
    writeln;
    writeln('            KONIEC.............................. 0');
    q:=readkey;
    clrscr;
    case q of
      '1': vstup_karta;
      '2': najdi;
      '3': odstran;
      '4': pocet;
      '5': vystup_kartoteka;
    end;
  until q='0';
  close(f);
  end;

BEGIN
clrscr;
assign(f,'pracov.dat');
gotoxy(10,12);
write('Vytvorit novy subor pracovnikov (ano - A) ? ');
{$I-} reset(f); {$I+}
if (IOResult<>0) or (upcase(readkey)='A') then rewrite(f);
menu;
END.
```

Príklad SUB-4: Vytvorte program, ktorý spojí dva súbory do jedného. Nech súbory obsahujú najviac dvadsaťznakové slová.

Analýza: Podprogram zabezpečujúci spojenie dvoch súborov je pomerne jednoduchý. Môžeme nastaviť prístupovú premennú na koniec v prvom súbore a pokračovať v zápise prvkov z druhého súboru. Ak má mať spojený súbor nový názov, stačí prvý súbor (v ňom sú všetky prvky) premenovať. Program sme doplnili aj o utriedenie prvkov súboru bubblesortom. Na triedenie súborov síce existujú špeciálne triediace algoritmy, pre zaujímavosť sme sa však „pohrali“ s bubblesortom. Všimnite si, že dve rôzne logické premenné súboru majú priradené to isté fyzické meno. Je to možné len za určitých podmienok (v bubblesorte sa nemení počet zložiek súboru, len ich poradie) a preto sa takýmto „perličkám“ radšej vyhýbajte.

```
program SUB_4;
uses  crt;
const DLSLOVA=20;
type  SLOVO=string[DLSLOVA];
      SUBOR=file of SLOVO;
      NAZOV=string[12];
var   FM1,FM2,FMS:NAZOV;

procedure VSTUP(FM:NAZOV);
  var F:SUBOR;
      S:SLOVO;
  begin
  assign(F,FM);
  rewrite(F);
  repeat
    write('Vlozit slovo: ');
    readln(S);
    write(F,S);
    writeln('Koniec - stlac 0':45)
  until upcase(readkey)='0';
  close(F)
  end;

procedure VYSTUP(FM:NAZOV);
```

```
  var F:SUBOR;
      S:SLOVO;
  begin
  assign(F,FM);
  reset(F);
  while not eof(F) do
    begin
    read(F,S);
    write(S,' ')
    end;
  writeln;
  close(F)
  end;

procedure SPOJ(FM1,FM2,FMS:NAZOV);
  var F1,F2:SUBOR;
      S:SLOVO;
  begin
  assign(F1,FM1); reset(F1);
  assign(F2,FM2); reset(F2);
  seek(F1,filesize(F1));
  while not eof(F2) do
    begin
    read(F2,S);
    write(F1,S)
    end;
  close(F1); close(F2);
  assign(F1,FMS); erase(F1);
  assign(F1,FM1); rename(F1,FMS);
  end;

procedure BUBBLESORT(FM:NAZOV);
  var F,FPom:SUBOR;
      A,B:SLOVO;
      DL,PP,I:integer;
  begin
  assign(F,FM); assign(FPom,FM);
  reset(F);
  DL:=filesize(F);
  for PP:=1 to DL-1 do
    begin
    reset(F); reset(FPom);
    read(F,A);
    for I:=1 to DL-PP do
      begin
      read(F,B);
      if A<B
        then begin write(FPom,A); A:=B end
        else write(FPom,B)
      end;
      write(FPom,A)
    end;
  close(F);close(FPom)
  end;

BEGIN
clrscr;
write('Meno prveho suboru na disku: '); readln(FM1);
VSTUP(FM1);
write('Meno druheho suboru na disku: '); readln(FM2);
VSTUP(FM2);
write('Meno spojeneho suboru na disku: '); readln(FMS);
clrscr;
writeln; writeln('Prvy subor:');
VYSTUP(FM1);
writeln; writeln('Druhy subor:');
```

```
VYSTUP(FM2);
writeln; writeln('Spojeny subor:');
SPOJ(FM1,FM2,FMS);
VYSTUP(FMS);
writeln; writeln('Utriedeny subor:');
BUBBLESORT(FMS);
VYSTUP(FMS);
readln
END.
```

Textové súbory

Existujú súbory dát, ktoré sú členené na riadky. Takéto súbory nazývame textové a ich typ sa označuje štandardným identifikátorom typu *text*. V textovom súbore F okrem funkcie eof(F) je deklarovaná aj funkcia eoln(F) (end of line), slúžiaca na rozpoznanie konca riadku. Príkazy read(F,P) a write(F,V) sú rozšírené o príkazy readln(F,P) – z textového súboru F sa prečíta lexikálna jednotka (hodnota) zodpovedajúca typu premennej P a prejde sa na nový riadok, a writeln(F,V) – do textového súboru F sa zapíše hodnota výrazu V a oddeľovač riadkov eoln.

Najznámejšie súbory typu text sú štandardný vstupný súbor **input**, ktorému je priradené vstupné zariadenia klávesnica, a štandardný výstupný súbor **output**, ktorému je priradené výstupné zariadenie monitor. Definovanie ich typu ako súborov typu text je implicitné. K otvoreniu oboch súborov dochádza automaticky po spustení programu. Príkazy read(input,P), readln(input,P), write(output,V) a writeln(output,V) sa používajú bez identifikátorov input a output.

Príklad SUB-5: Vytvorte program, ktorý bude čítať znaky z klávesnice (vstupný súbor input) a pre každý riadok zobrazí, koľkokrát sa v ňom vyskytuje jeho prvý znak.

Analýza: Program musí obsahovať dva vnorené cykly. Vonkajší cyklus zabezpečuje čítanie znakov, až kým nie je načítaný koniec súboru. Vnútorný cyklus číta znaky, až kým nie je koniec riadku, potom musí odriadkovať. Vložiť znak koniec riadku nie je problém, stačí stlačiť Enter (chr(13)). Ak si spomenieme, že „v starom dobrom DOSe“ sa koniec súboru vkladal kombináciou kláves Ctrl+Z, sme „zachránení“.

```
program SUB_5;
var pocet:integer;
    z,s:char;
begin
writeln('Vkladaj text, ukoncenie - Ctrl+Z !!!');
while not eof do
  begin
  read(z);
  pocet:=1;
  while not eoln do
    begin
    read(s);
    if s=z then inc(pocet);
    end;
  writeln('Pocet vyskytov znaku ',z,' je ',pocet,'-krat');
  readln;
  end;
end.
```

Príklad SUB-6: Pokúste sa vysvetliť, prečo oba ďalej uvedené programy SUB-6a aj SUB-6b pracujú rovnako. Ich úlohou je uložiť do súboru na disk text z klávesnice.

```
Program SUB_6a;
var z:char;
    T:text;
    FM:string[12];

BEGIN
write('Meno suboru na disku: ');
readln(FM);
assign(T,FM);
rewrite(T);
writeln('Vkladaj text, koniec - vloz Ctrl+Z !!!');
while not eof do
  begin
  while not eoln do
    begin
    read(z);
    write(T,z)
    end;
  readln;
  writeln(T)
  end;
writeln;
writeln('Ulozeny subor:');
reset(T);
while not eof(T) do
  begin
  while not eoln(T) do
    begin
    read(T,z);
    write(z);
    end;
  readln(T);
  writeln
  end;
close(T);
readln
END.

program SUB_6b;
uses crt;
var  z:char;
     T:text;
     FM:string[12];

BEGIN
clrscr;
write('Meno suboru na disku: ');
readln(FM);
assign(T,FM);
rewrite(T);
writeln('Vkladaj text, koniec - stlac `');
repeat
  read(z);
  if z<>'`' then write(T,z)
until z='`';
writeln;
writeln('Ulozeny subor:');
reset(T);
while not eof(T) do
  begin
  read(T,z);
  write(z);
  end;
close(T);
readln;readln
END.
```

Neriešené úlohy na súbor:

7. Kartotéku z príkladu SUB-3 doplňte o položku rodné číslo a podprogram, ktorý zistí počet osôb mužského a ženského pohlavia v kartotéke (u mužov je v rodnom čísle na 3. mieste 0 alebo1, u žien 5 alebo 6).
8. Kartotéku z príkladu SUB-3 doplňte o položku plat a podprogram, ktorý vypíše osoby, ktorých plat spĺňa zadané kritériá (menší, väčší, rovný).
9. Vytvorte program, ktorý zo súboru reálnych čísel prepíše do nového súboru len celé čísla.
10. Úloha z jednorozmerného poľa o vyhľadávaní, počte výskytov, miesta výskytu a pod. zrealizujte pre súbor celých čísel (znakov, reťazcov).
11. Vytvorte program, ktorý z daného textového súboru odstráni všetky viacnásobné medzery a prázdne riadky.
12. Vytvorte program slúžiaci ako elektronický dvojjazyčný slovník.
13. Vytvorte program, ktorý bude čítať znaky z klávesnice a pre každý riadok určí súčet ordinálnych čísel znakov vyskytujúcich sa v riadku.

Dynamické premenné

Údajové typy, s ktorými sme sa zaoberali doteraz, patria medzi tzv. statické údajové typy. Premenné prislúchajúce statickým typom sú zavedené v úseku definícií a deklarácií a je im trvalo vyhradené miesto v pamäti počas behu programu. Počet a rozsah premenných sa počas práce v bloku, t.j. aj v programe, nemôže meniť.

Jazyk pascal umožňuje vytvárať premenné nie len v úseku deklarácií premenných, ale aj v príkazovej časti. Dokonca možno tieto premenné v príkazovej časti aj rušiť. Premenné s týmito vlastnosťami nazývame **dynamické premenné** a príslušné údajové typy nazývame dynamické údajové štruktúry.

Z toho, čo sme uviedli doteraz, vyplýva, že dynamické premenné nemožno zaviesť v úseku deklarácií a teda zabezpečiť prístup k ich hodnotám pomocou ich identifikátorov. Dynamické premenné vznikajú a zanikajú počas realizácie programu a sprístupnenie ich hodnôt sa robí pomocou nového údajového typu **ukazovateľ**.

Ak t je identifikátor nejakého typu, tak typ ukazovateľ definujeme zápisom:

```
type u = ^ t
```

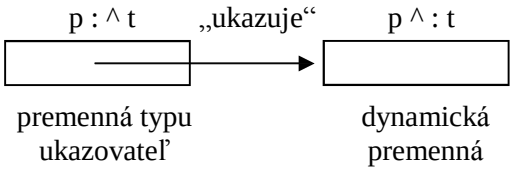
kde u je meno (identifikátor) typu ukazovateľ. Hovoríme aj, že typ t je zviazaný s typom u a množina hodnôt premennej typu ukazovateľ „ukazuje“ na prvky typu t.

Napríklad:

```
type   UKAZ1 = ^ integer;
       UKAZ2 = ^ PRVOK;
```

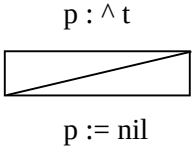
Ak p je premenná typu ukazovateľ, tak premenná typu zviazaného s typom ukazovateľ (premenná typu t) sa nazýva dynamická premenná a označuje sa p^.

Grafická interpretácia:



Pomocou hodnoty premennej typu ukazovateľ sprístupňujeme dynamickú premennú. V úseku definícií a deklarácií je zavedená len premenná typu ukazovateľ! Do množiny hodnôt premennej typu ukazovateľ patrí vždy aj hodnota *nil*, jediná konštanta typu ukazovateľ, ktorá neukazuje na nijaký prvok (na žiadnu dynamickú premennú).

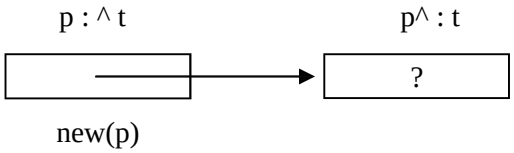
Graficky:



Premenná p má síce priradenú hodnotu, ale „neukazuje“ na žiadnu dynamickú premennú.

Príkazom **new(p)**, kde p je premenná typu ukazovateľ, sa vytvorí dynamická premenná typu t s nedefinovanou hodnotou a ukazovateľ na túto premennú sa uloží do premennej p. Príkazom new(p) priraďujeme premennej p hodnotu, ktorou je zodpovedajúca dynamická premenná a ktorú označujeme p^.

Graficky:



Príkazom **dispose(p)**, kde p je premenná typu ukazovateľ, sa zruší dynamická premenná p^, na ktorú ukazovala premenná p. Hodnota premennej p nie je po ukončení príkazu dispose(p) definovaná. Pamäťový priestor, ktorý zaberala dynamická premenná, sa dal k dispozícii na ďalšie použitie procesorom.

Zhrnutie:

var p : ^t ; p

(v pamäti počítača sa vyhradí pamäťové miesto pre statickú premennú typu ukazovateľ, t.j. 32 bitov pre „akúsi“ adresu)

p := nil ; p

(do pamäťového miesta vyhradeného pre premennú typu ukazovateľ sa uloží číslo neexistujúcej adresy, napríklad 0)

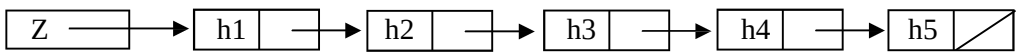
new (p) ; p ?

p^ : t

(do pamäťového miesta vyhradeného pre premennú typu ukazovateľ sa uloží adresa vytvorenej dynamickej premennej p^)

Lineárny jednosmernezreťazený zoznam

Z dynamických štruktúr sa najčastejšie využíva lineárny jednosmernezreťazený zoznam. Graficky ho možno znázorniť:



kde Z je premenná typu ukazovateľ (pointer), ktorá ukazuje na prvú dynamickú premennú v zozname, ktorého prvky sú typu záznam (record) s dvoma položkami: s hodnotou (h1 až h5) a s ukazovateľom na ďalší prvok zoznamu. Ukazovateľ posledného prvku zoznamu už nemá kde ukazovať a preto má hodnotu nil.
Postup v zozname je sekvenčný, od ukazovateľa na prvý prvok cez ukazovatele na nasledujúci prvok až na koniec zoznamu. V úseku definícií a deklarácií je zavedený len ukazovateľ na zoznam (premenná Z typu ^t) a typ dynamickej premennej (t). Vlastné prvky zoznamu, dynamické premenné, vznikajú podľa potreby počas práce programu príkazom new. Na pohyb v zozname používame pomocnú premennú typu ukazovateľ a premennú Z nechávame väčšinou ukazovať vždy na začiatok zoznamu (inak sa nemáme možnosť dostať na začiatok zoznamu).

Program pracujúci so zoznamom by začínal:

```
type    UKAZ=^PRVOK;
        PRVOK=record
                    hodnota: typ_hodnoty;
                    dalsi: UKAZ
                end;
var Z : UKAZ;
```

Prácu s lineárnym jednosmernezreťazeným zoznamom demonštruje nasledujúci príklad.

Program DYN-1: Vytvorte program umožňujúci vo forme menu vytvoriť prázdny zoznam, naplniť ho hodnotami, vypísať hodnoty zo zoznamu, odstrániť zvolenú hodnotu, uložiť hodnoty zoznamu do súboru a naopak, načítať ich zo súboru do zoznamu.

Analýza: Program je ponechaný na samoštúdium, odporúčame však kresliť si jednotlivé situácie pomocou zavedenej symboliky. Všimnite si, v akom poradí prvky ukladané do zoznamu!

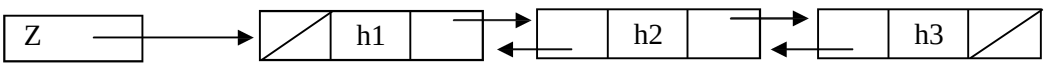
```
program DYN_1;
uses crt;
type ukazovatel=^prvok;
     prvok=record
             h:integer;
             u:ukazovatel
           end;
     subor=file of integer;
var  z:ukazovatel;
     sub:subor;
     q:char;

procedure vytvor(var z:ukazovatel);
  begin
  z:=nil
  end;

function prazdny(z:ukazovatel):boolean;
  begin
  prazdny:=z=nil
  end;

procedure nahodne(var z:ukazovatel);
  var p:ukazovatel;
      i:integer;
  begin
  z:=nil; randomize;
  for i:=1 to random(10)+2 do
    begin
    new(p);
    with p^ do
      begin
      h:=random(10);
      u:=z;z:=p;
      end;
    end;
  end;

procedure napln(var z:ukazovatel);
  var p:ukazovatel;
      q:char;
  begin
  z:=nil;
```

```
  repeat
    writeln('Vlozit prvok ( ano -> A ) ?');q:=upcase(readkey);
    if q='A'
      then begin
           new(p);
           with p^ do
             begin
             write('Hodnota: ');readln(h);
             u:=z;z:=p;
             end;
           end;
  until q<>'A';
  end;

procedure dopln(var z:ukazovatel);
  var p:ukazovatel;
      q:char;
  begin
  repeat
    writeln('Vlozit prvok ( ano -> A ) ?');q:=upcase(readkey);
    if q='A'
      then begin
           new(p);
           with p^ do
             begin
             write('Hodnota: ');readln(h);
             u:=z;z:=p;
             end;
           end;
  until q<>'A';
  end;

procedure vymaz(var z:ukazovatel);
  var p,p1:ukazovatel;
  begin
  if not prazdny(z)
    then begin
         repeat
           p:=z;
           writeln(p^.h:5,' - vymazat ( ano -> A ) ');q:=upcase(readkey);
           if q='A'
             then begin
                  z:=p^.u;
                  dispose(p);
                  p:=z
                  end
             else begin
                  p1:=p;
                  p:=p^.u
                  end;
         until q<>'A';
         while p<>nil do
           begin
           writeln(p^.h:5,' - vymazat ( ano -> A ) ');q:=upcase(readkey);
             if q='A'
               then begin
                    p1^.u:=p^.u;
                    dispose(p);
                    p:=p1^.u
                    end
               else begin
                    p1:=p;
                    p:=p^.u
                    end;
           end
         end
```

```
  end;

procedure vypis(z:ukazovatel);
  var p:ukazovatel;
  begin
  if prazdny(z)
    then writeln('Zoznam je prazdny')
    else begin
         p:=z;
         write(p^.h:5);
         while p^.u<>nil do
           begin
           p:=p^.u;
           write(p^.h:5)
           end
         end
  end;

procedure utried(var z:ukazovatel);
  var p,q,k:ukazovatel; pom:integer;
      bola_vymena:boolean;
  begin
  if (z<>nil)and(z^.u<>nil)
    then repeat
         p:=z^.u; q:=z; bola_vymena:=false;
         while (p<>nil)and(p<>k) do
           begin
           if p^.h<q^.h
             then begin
                  pom:=p^.h; p^.h:=q^.h; q^.h:=pom;
                  bola_vymena:=true
                  end;
           q:=p; p:=p^.u
           end;
         k:=q
         until not bola_vymena;
  end;

procedure zapis_sub(z:ukazovatel);
  var p:ukazovatel;
  begin
  rewrite(sub);
  p:=z;
  repeat
    write(sub,p^.h);
    p:=p^.u
  until p^.u=nil;
  write(sub,p^.h);
end;

procedure citaj_sub(var z:ukazovatel);
  var p:ukazovatel;
  begin
  reset(sub);
  z:=nil;
  while not eof(sub) do
    begin
    new(p);
    with p^ do
      begin
      read(sub,h);
      u:=z;z:=p;
      end;
    end;
end;
```

```
BEGIN
assign(sub,'dynamic.sub');
rewrite(sub);
vytvor(z);
repeat
  clrscr;gotoxy(1,5);
  WRITELN('VYTVOR PRAZDNY ..... 1':53);
  WRITELN('VYTVOR NAHODNE ..... 2':53);
  WRITELN('NAPLN ..... 3':53);
  WRITELN('DOPLN ..... 4':53);
  WRITELN('VYMAZ ..... 5':53);
  WRITELN('UTRIED ..... 6':53);
  WRITELN;
  WRITELN('VYPIS ..... 7':53);
  WRITELN;
  WRITELN('NACITAJ ZO SUBORU ..... 8':53);
  WRITELN('ULOZ DO SUBORU ..... 9':53);
  WRITELN;
  WRITELN('KONIEC ..... 0':53);
  q:=readkey;clrscr;
  case q of
    '1':vytvor(z);
    '2':nahodne(z);
    '3':napln(z);
    '4':dopln(z);
    '5':vymaz(z);
    '6':utried(z);
    '7':vypis(z);
    '8':citaj_sub(z);
    '9':zapis_sub(z);
  end;
if (q>'2') and (q<'5') or (q='7') then readln
until q='0';
close(sub)
END.
```

Na záver uvedieme príklad, ktorý využíva lineárny obojsmernezreťazený zoznam, ktorý možno graficky znázorniť (3-prvkový):



Program DYN-2: Vytvorte program na výpočet cifier n! s využitím dynamických dátových štruktúr.

Analýza: Výpočet n! zlyháva u štandardných celočíselných typov už pri n=13. Nasledujúci program počíta cifry n-faktoriálu pre „neobmedzené“ n. Podstatou je „manuálne“ násobenie, cifru za cifrou.

```
program DYN_3;
uses  crt;
const Esc=chr(27);
type  ukaz=^prvok;
      prvok=record
              cifra:0..9;
              pred,za:ukaz
            end;
var   cislica,prva:ukaz;
      posl:pointer;
      i,n,pocet,prenos,stl,ria:integer;
                          sucin:integer; { longint }
procedure text_pocitam;
  begin
    stl:=wherex;ria:=wherey;
    gotoxy(34,1);textcolor(15+128);writeln('POCITAM  ');
```

```
    gotoxy(stl,ria);normvideo;
  end;

procedure text_faktorial;
  begin
    stl:=wherex;ria:=wherey;if ria=1 then ria:=2;
    gotoxy(34,1);writeln('FAKTORIAL');
    gotoxy(stl,ria)
  end;

BEGIN
repeat
  clrscr;
  text_faktorial;
  write('N = ');readln(n);write(n,' ! = ');
  text_pocitam;
  new(cislica);
  with cislica^ do begin cifra:=1;pred:=nil;za:=nil end;
  posl:=cislica;prva:=cislica;prenos:=0;sucin:=1;
  for i:=1 to n do
    begin
    cislica:=posl;
    repeat
      with cislica^ do
        begin
        sucin:=cifra*i+prenos;
        cifra:=sucin mod 10;
        prenos:=sucin div 10;
        cislica:=pred
        end
    until cislica=nil;
    while prenos<>0 do
      begin
      new(cislica);
      with cislica^ do
        begin
        pred:=nil;za:=prva;prva^.pred:=cislica;prva:=cislica;
        cifra:=prenos mod 10;prenos:=prenos div 10;
        end;
      end;
    end;
  text_faktorial;
  cislica:=prva;pocet:=0;
  repeat
    write(cislica^.cifra);pocet:=pocet+1;cislica:=cislica^.za;
    dispose(prva); prva:=cislica;
  until cislica=nil;
  writeln;writeln('Pocet cifier v cisle: ',pocet, '        Koniec - Esc')
until readkey=Esc
END.
```

Neriešené úlohy na dynamické dátové štruktúry:

3. Zásobník je dynamická dátová štruktúra, na ktorej sú definované operácie:
   a) pridať prvok na vrch zásobníka
   b) odobrať prvok z vrchu zásobníka
   c) zistiť, či je zásobník prázdny
   d) vytvoriť prázdny zásobník

   Zrealizujte zoznam pracujúci ako zásobník (zoznam typu LIFO, last-in first-out).
4. Vytvorte program kontrolujúci správnosť ozátvorkovania výrazu s využitím v príklade 3 opísaného zásobníka.
5. Rad je dynamická dátová štruktúra, na ktorej sú definované operácie:
   a) pridať prvok na koniec radu
   b) odobrať prvok zo začiatku radu
   c) zistiť, či je rad prázdny
   d) vytvoriť prázdny rad

   Zrealizujte zoznam pracujúci ako rad (zoznam typu FIFO, first-in first-out).
6. Vytvorte program na výpočet Fibonacciho čísel pomocou trojprvkového radu.

   Fibonacciho postupnosť: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,...

   vzorcami: $F_0 = 0 \quad F_1 = 1 \quad F_i = F_{i-1} + F_{i-2} \quad \text{pre } i \geq 2$
7. Vytvorte program na výpočet hodnôt Pascalovho trojuholníka pomocou radu.
8. Vytvorte podprogramy na vyhľadávanie, zistenie počtu výskytov, miesta výskytu a pod. prvkov so zvolenými vlastnosťami v lineárnom jednosmernezreťazenom zozname.
9. Vytvorte program, ktorý spojí dva lineárne jednosmernezreťazené zoznamy do jedného.

ZÁKLADY PROGRAMOVANIA

TURBO PASCAL

Jozef Piroško

Názov: Základy programovania - Turbo Pascal
Autor : RNDr. Jozef Piroško
Vydanie: prvé
Vydané: marec 2000

OBSAH

Úvod

Skriptá Základy programovania – Turbo Pascal sú sumarizáciou poznatkov z klasického kurzu programovania. Nekladú si za cieľ podrobne vyložiť všetky pojmy. Tiež výklad a poradie tém má klasické usporiadanie, od najjednoduchšieho k zložitejšiemu, od základných pojmov k zložitejším štruktúram. Výklad a príklady sa nevyhýbajú úlohám s matematickým základom, ako sa o to pokúšajú modernejšie učebnice programovania. Domnievam sa, že ak máte odmietavý vzťah k matematike, nemôže vás programovanie a algoritmizácia zaujať. Musíte byť trochu matematik, aby ste presne a jednoznačne vedeli formulovať a zapísať svoje myšlienky a trochu umelec, aby ste dokázali abstraktne myslieť a vytvoriť si svoje originálne riešenie daného problému. Základné „ťahy štetcom“ sa možno naučíte práve z týchto skrípt.

Obsahom výkladu sú základné algoritmické konštrukcie sekvencia, vetvenie a cyklus, a príkazy Turbo Pascalu, umožňujúce zrealizovať tieto konštrukcie v programe. Preto sa v príkladoch používajú len štandardné údajové typy integer, real, boolean a char a jediný štruktúrovaný typ string. Nekládol som si za cieľ o každom zavedenom pojme uviesť všetko, skôr ukázať jeho najtypickejšie použitie, preto sa miestami vedome dopúšťam nepresností alebo skôr neúplností, mali by však byť skutočne nepodstatné; možno sa dopúšťam aj chýb, ale tých určite nevedome. Tiež si nekladiem nároky na autorstvo použitých príkladov. Vybral som ich z veľkého množstva príkladov, s ktorými som sa stretol za viac ako desať rokov praxe. O originálnosti riešenia ťažko hovoriť, keďže ide o „triviálne“ príklady. Týmto spĺňam aj požiadavku študentky Maji S., ktorá ma raz poprosila o vydanie Triviálností pána procesora Pirošku (premenovanie je tiež dielom študenta, tentoraz Tomáša Z.).

Skriptá, okrem výkladu základných štruktúr a príkazov, obsahujú aj veľké množstvo riešených a neriešených úloh. Najprv by ste sa sami mali potrápiť nad úlohou, a až potom siahnuť po riešení v skriptách. Riešenia si treba ozrejmiť, tým sa naučíte minimálne „čítať“ programy a možno aj nové riešenie daného problému. Ideálne by bolo, keby ste si pri štúdiu programu kládli a najmä odpovedali na otázky charakteru: Počíta program správne aj pre takéto hodnoty?, Musí to byť zapísané práve takto?, Počiatočná hodnota premennej musí byť práve takáto?, Čo sa stane, keď to napíšem takto?, Čo keď zamením poradie týchto príkazov? A ak tento príkaz nahradím týmto? atď. Svoje tvrdenia si vždy overte priamo v TP odladením zmeneného programu.

Na záver vám prajem veľa originálnych riešení, najprv sa však treba naučiť „v pote tváre zopár základných ťahov štetcom“.

Algoritmus, vlastnosti algoritmu, formy zápisu algoritmu, algoritmické konštrukcie

Ak sa začnete zaujímať o počítače a najmä programovanie, určite narazíte na slovo algoritmus. Algoritmus je návod, postup, ako vyriešiť určitý problém. Tým problémom môže byť výmena pneumatiky pri defekte, zostavenie mlynčeka na mletie mäsa tak, aby to, čo vznikne zomletím, bolo ešte ďalej použiteľné, bezpečné prejdenie na druhú stranu cesty, „vzorné“ umytie zubov atď. Z týchto príkladov by malo byť zrejmé, že algoritmy nás učia od narodenia a na každom kroku („toto sprav takto a toto zas takto...“). Snáď v každej domácnosti sa nachádza zbierka algoritmov na prípravu jedál – kuchárska kniha. Každý návod na obsluhu, ktorý by ste mali dostať pri kúpe nového výrobku, je vlastne algoritmom. Vo všetkých týchto príkladoch je vykonávateľom algoritmu človek. Pri zápise algoritmu určeného pre človeka si môžeme dovoliť použiť spojenia „krátko povariť“, „pritiahnuť až na doraz“, „vyladiť stanicu“ a pod. Ak je však vykonávateľom algoritmu stroj, napr. počítač, sotva by porozumel uvedeným príkazom. Preto aj my prejdeme na odbornejšiu terminológiu (chceme sa predsa venovať počítačom a nie vareniu). **Algoritmus** je konečná postupnosť elementárnych príkazov, vykonanie ktorých, pre prípustné vstupné údaje, po konečnom počte krokov (akcií) vedie mechanicky k získaniu korektných výstupných údajov. Strašná veta, skúsme ju „rozpitvať“. Konečná postupnosť elementárnych príkazov znamená, že ide o príkazy, ktoré sú zapísané v presne stanovenom poradí, je ich konečný počet a pre vykonávateľa algoritmu sú elementárne, t.j. každý z príkazov pozná a vie ho vykonať. Vykonávanie algoritmu je proces, do ktorého vo všeobecnosti vstupujú na začiatku prípustné vstupné údaje a ich spracovaním získavame nové – výstupné údaje (na čo by sme ináč „trápili“ počítač). Celý proces samozrejme nesmie trvať nekonečne dlho (to by sme sa nedopracovali k žiadnym výsledkom) a, keďže počítač je síce usilovný ale hlupák[1], celý proces musí prebiehať mechanicky - bez rozmýšľania. Pod korektnými výstupnými údajmi rozumieme zmysluplné výsledky adekvátne vstupným údajom; ak napríklad opakovane použijeme rovnaké vstupné údaje, musíme dostať tie isté výsledky.

Z tejto analýzy vyplývajú aj **vlastnosti** algoritmu, z ktorých by sme uviedli len niekoľko:

1. hromadnosť – algoritmus slúži na riešenie celej skupiny úloh určitého typy[2]; aby sme dodržali túto vlastnosť, musí byť algoritmus napísaný všeobecne (univerzálne). Z matematiky vieme, že zovšeobecniť zápis nám umožňujú premenné. Už na základnej škole nás učili, že napríklad obsah obdĺžnika sa vypočíta S = a.b, kde a a b sú strany obdĺžnika a nie S = 5.7 pre nejaký konkrétny obdĺžnik. Preto prvý zápis môže byť príkazom algoritmu, druhý nie. Zaujímavé je, že algoritmus musí byť napísaný všeobecne ale jeho realizácia vždy prebieha s konkrétnymi údajmi; premenné nadobúdajú konkrétne hodnoty v závislosti od zadaných vstupných hodnôt. Ak budú vstupné údaje - strany obdĺžnika a = 5 a b = 7, tak konkrétny výpočet bude S = 5.7 a po ňom premenná S nadobudne hodnotu 35. Hromadnosť vlastne znamená, že algoritmus slúži na riešenie určitej skupiny konkrétnych úloh, ktoré sa líšia len vstupnými hodnotami.

---

[1] Jeden múdry pán raz povedal: „Počítač je usilovný hlupák na triedenie jednotiek a núl“. Dúfam, že s ním súhlasíte.

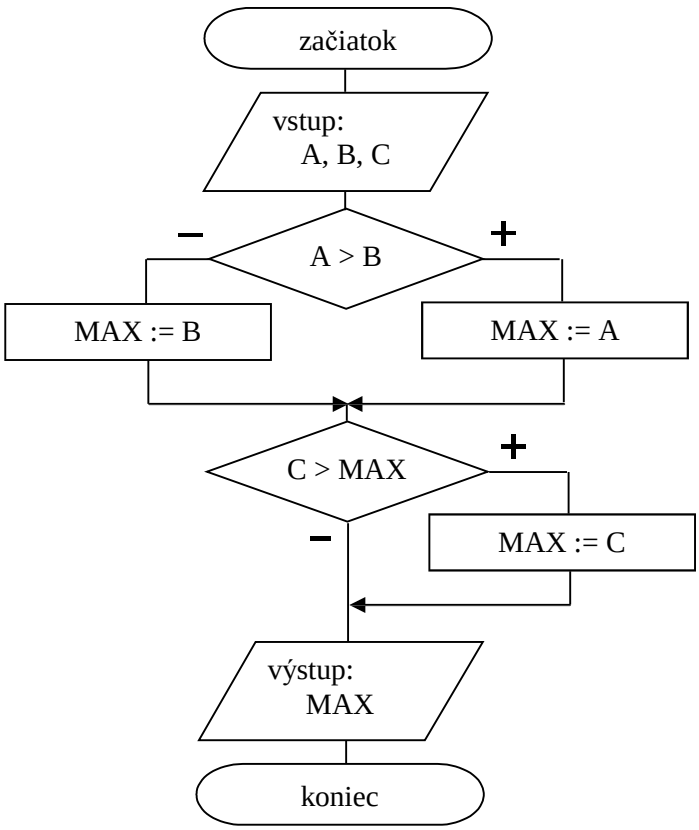
[2] My predsa nevieme vymeniť len tú konkrétnu pneumatiku na aute, na ktorom nás to učili, ale u daného typu auta kedykoľvek ktorúkoľvek pneumatiku; alebo prejsť cez cestu by sme mali vedieť bezpečne cez ktorúkoľvek cestu, podobnú tej, na ktorej nás to učili; zuby si tiež vieme umyť, či ich máme 32 alebo menej, atď.

2. deterministickosť (jednoznačnosť) – jednotlivé kroky (akcie) a ich poradie je presne a jednoznačne určené. Deterministickosť zaručuje, že pri opätovnom použití algoritmu pre tie isté vstupné údaje dostaneme rovnaký výsledok. Umožňuje tiež realizáciu algoritmu počítačom (počítač sa nemusí sám rozhodovať).

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; ide o staršiu formu zápisu algoritmu.
  Príklad: Algoritmus na nájdenie najväčšieho z troch čísel vo forme vývojového diagramu.



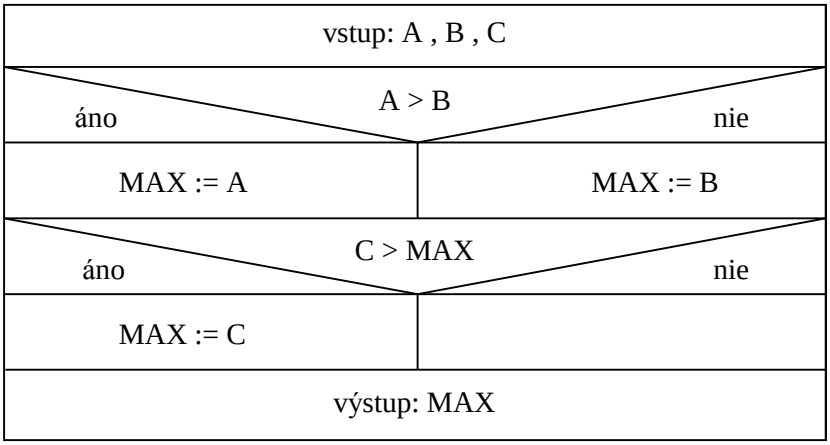
  Algoritmus sa vykonáva zhora nadol. Vetvou „+“ sa pôjde, keď bude podmienka splnená a vetvou „ -“, keď podmienka nebude splnená. Symbol „:=“ čítame „priraď“.

- slovný zápis používa tzv. kľúčové (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[1] – najnovšia grafická forma zápisu algoritmu.

  Príklad: Algoritmus na nájdenie najväčšieho z troch čísel zapísaný N-S diagramom.



Položme si otázku: Je každá úloha algoritmicky riešiteľná, t.j. existuje na každý problém algoritmus, pomocou ktorého ho vieme vyriešiť? Intuitívne cítime, že asi nie. Na algoritmicky neriešiteľné úlohy síce nenarazíme „na každom kroku“, ale predsa existujú. Podarilo sa napríklad dokázať tvrdenie: Neexistuje algoritmus, ktorý by pre akýkoľvek daný program rozhodol (na základe zdrojového tvaru programu a vstupných údajov), či tento program skončí.[2] Zatiaľ sme neobjasnili slovo *program*. Veľmi jednoducho: **program** je algoritmus úlohy zapísaný v jazyku, ktorému rozumie počítač, v tzv. programovacom jazyku. Onedlho sa jedným z programovacích jazykov začneme podrobnejšie zaoberať.

Ak je úloha algoritmicky riešiteľná, jej algoritmus možno vytvoriť kombináciou len troch algoritmických konštrukcií (Na prvý pohľad aké jednoduché!). **Algoritmické konštrukcie** sú: sekvencia, vetvenie a cyklus.

Teraz prerušíme výklad algoritmických konštrukcií a povieme si niečo o programovaní a programovacom jazyku Pascal.

Programovanie, programovací jazyk Turbo Pascal

Tvorbu algoritmov nazývame **algoritmizácia** a tvorbu programov **programovanie**. Keďže program je „len“ algoritmus zapísaný v programovacom jazyku, prioritnou činnosťou je algoritmizácia. Málokto však vydrží najprv „vyrobiť“ nedozerné množstvo algoritmov (naučiť sa algoritmizáciu) a až potom začať programovať. Je to pochopiteľné, pretože až počítač pomocou programu mu dokonale „zhmotní“ to, čo vymyslel na algoritmizácii. Preto aj my, aby sme si hneď overili, čo sme vymysleli, spojíme algoritmizáciu a programovanie. Pri preberaní algoritmických konštrukcií si hneď uvedieme aj príkazy programovacieho jazyka, ktoré nám umožňujú danú konštrukciu zrealizovať v programe. Budeme sa zaoberať základmi programovacieho jazyka Turbo Pascal, implementáciou jazyka Pascal, ktorý navrhol v roku 1971 profesor

[1] Tento názov sme zvolili pre podobné formy zápisu algoritmu, ako sú napríklad kopenogramy (autori Kofránek, Pecinovský a Novák) alebo N-S diagramy (autori Nassi a Schneiderman).

[2] Toto tvrdenie napríklad vedie k všeobecnejšiemu tvrdeniu: Neexistuje algoritmus, ktorý rozhodne, či v programe je alebo nie je chyba. Už vieme, prečo je „programátorský chlebík“ taký ťažký.

informatiky Niklaus Wirth[1]. Pascal bol vytvorený na vyučovanie programovania a má vlastnosti a kontrolné mechanizmy nútiace, alebo aspoň umožňujúce, tvoriť zrozumiteľné a prehľadné programy s čo najmenším počtom začiatočníckych chýb.

Program nie je len mechanickým prepisom algoritmu do programovacieho jazyka. Kým v algoritme sa snažíme vystihnúť podstatu riešenia problému a nezaoberáme sa podrobnosťami komunikácie s užívateľom, program sa snažíme spraviť užívateľsky „prítulný“ (budeme hovoriť o užívateľskom komforte). Na to slúžia ďalšie, rozširujúce príkazy. Tieto a ďalšie vlastnosti programovacieho jazyka Turbo Pascal si budeme postupne ozrejmovať pri konkrétnych programoch.

Pri práci v Turbo Pascale odporúčame:

- v Options – Environment – v položke Preferences... v skupine Auto save zapnúť voľbu Editor files a zapamätať ju položkou Save \...\ TURBO.TP; táto voľba spôsobí, že pred prekladom programu sa tento automaticky najprv uloží na disk (ak sme súbor s programom ešte nepomenovali, vyzve nás vložiť názov súboru), čo pri prípadnom použití tlačidla RESET neznamená stratu programu (najprv však skúste Ctrl+Break alebo v prostredí Windows 9x Ctrl+Alt+Del)
- v Options – Compiler... zapnúť v skupine Runtime errors všetky kontroly na chyby počas behu programu a zapamätať ich položkou Save \...\ TURBO.TP (jednotlivé kontroly preberieme podľa aktuálnosti)
- aj písanie nových programov začínať cez F3 Open a hneď vložiť názov súboru, do ktorého má byť nový program ukladaný
- čo najviac používať kontextový help: napísať kľúčové slovo, presunúť pod neho kurzor a stlačiť Ctrl+F1
- používať po označení textu pomocou myši alebo Shift a „kurzorových šípok“ na kopírovanie kombináciu Ctrl+C a K, premiestňovanie Ctrl+C a V a mazanie Ctrl+C a Y (alebo položky z ponuky Edit)
- programy spúšťať pomocou Ctrl+F9. Ak sme program práve napísali alebo v ňom spravili zmenu, dôjde najprv k prekladu programu, pred ktorým sa vykoná aj kontrola správnosti zápisu (odhalia sa skomolené vyhradené slová, chýbajúce bodkočiarky a pod.) a úplnosti programu (odhalia sa nedeklarované premenné, nedovolené priradenia a pod.). Táto kontrola neodhalí tzv. logické chyby (chyby v algoritme) alebo chyby, ktoré môžu nastať počas behu programu (zle zadaná vstupná hodnota, delenie nulou[2] a pod.)

Pri písaní programu si programátor volí názov programu, mená použitých premenných, konštánt, typov a iných objektov. Keďže norma Pascalu dovoľuje používať len písmená anglickej abecedy, vo zvolených názvoch nesmú byť použité diakritické znamienka (dĺžne, mäkčene a pod.). Tieto názvy sú tzv. identifikátory. **Identifikátor** je postupnosť písmen (anglickej abecedy) a číslic začínajúca písmenom; medzi písmená je dodefinovaný aj podčiarnik „_“. Identifikátory sú napríklad: CISLO, SUCET, obvod_obdlznika, ObsahStvorca, Premenna1, MENO, S, X, y, A, a, N5, z_13, ZNAK, text, OBSAHUJE, ... Identifikátory nie sú napríklad: 1A (nezačína písmenom), SUCET CISEL (použitá medzera), premenná1 (nedovolené písmeno á), suma$ (nedovolený znak $). Nerozlišujú sa malé a veľké písmená, zápisy sucet, Sucet, SuCeT,

[1] Narodil sa v roku 1934, prednášal na Vysokej škole technickej a univerzite v Zürichu.

[2] Ak spúšťate Turbo Pascal na najnovších počítačoch Pentium, pri použití unitu Crt môže byť po spustení programu zahlásená chyba Error 200: Division by zero. Odstrániť ju možno použitím unitu FDelay pred Crt (FDelay sa nenachádza v štandardnej knižnici UNITS).

SUCET pomenúvajú v Pascale tú istú premennú. Názov objektu je dobré voliť si tak, aby nám napovedal o jeho úlohe, veľmi dlhé názvy však nie sú praktické.

Iná situácia je pri znakových reťazcoch alebo skrátene len reťazcoch. **Reťazec** je text uzavretý v apostrofoch. Napríklad: 'Súčet čísel = ', 'Zadaj tri celé čísla: ', 'Najväčšie číslo je ', 'Vlož svoje meno'. V reťazcoch sa rozlišujú veľké a malé písmená, dovolené sú diakritické znamienka, medzery, interpunkčné znamienka. Príkazom write možno reťazce zobraziť na obrazovke v tvare, ako sú zapísané, uľahčujú najmä komunikáciu medzi užívateľom a programom.

S identifikátormi a reťazcami sa budeme v programoch „stretávať na každom kroku“. Identifikátory budeme, aspoň zo začiatku, písať veľkými písmenami, aby sa čitateľ ľahšie zorientoval. A teraz jednoduchý program na zoznámenie sa so štruktúrou, spúšťaním a úpravou programov.

Príklad S-1: Vytvorte algoritmus a program, ktorý pozdraví vypísaním slova Ahoj! na obrazovke.

```
algoritmus POZDRAV_1;          program POZDRAV_1;
začiatok                       begin
píš('Ahoj');                   write('Ahoj!');
koniec.                        end.
```

V ľavom stĺpci je slovný zápis algoritmu a v pravom program – prakticky preklad do angličtiny. Po spustení Turbo Pascalu (TP) pomocou súboru turbo.exe v adresári BIN odporúčame stlačiť F3 Open (alebo F10 – File – Open...) a hneď zadať meno súboru, do ktorého chceme program uložiť. Je dobré si uvedomiť rozdiel medzi menom programu a menom súboru, v ktorom je uložený daný program. Meno programu je identifikátor, ktorý môže mať aj viac ako osem znakov. Meno súboru môže mať najviac osem dovolených znakov (súbor s programom POZDRAV_1 môžete nazvať napríklad s-1 alebo pozdrav1; príponu .pas pridá Turbo Pascal sám). Po zadaní mena súboru a kliknutí na tlačidlo Open alebo stlačení klávesu Enter Turbo Pascal automaticky prejde do editora („modré“ okno), slúžiaceho na písanie a upravovanie programov (ak sa v editore zobrazil text, znamená to, že so zadaným názvom už súbor v adresári existuje a práve sa otvoril – pre nový program musíme zvoliť iné meno súboru). Po napísaní programu sa môžeme pokúsiť spustiť program stlačením kláves Ctrl+F9 alebo kliknutím na Run – Run (horné menu TP možno zaktívniť stlačením klávesu F10). Po odstránení prípadných chýb sa vykonajú príkazy programu a automaticky sa TP prepne do editora. Výstupy na obrazovku, v našom prípade pozdrav Ahoj!, je na užívateľskej obrazovke, do ktorej prepneme stlačením Alt+F5 alebo F10 - Debug - User screen. Po prezretí výsledku práce programu stlačením ľubovoľného klávesu sa vrátime do prostredia TP, presnejšie editora.

Ak chceme ukončiť prácu v prostredí TP, môžeme použiť položky horného menu F10 – File – Exit alebo klávesovú skratku Alt+X. Pri neuložených programoch sa nás TP pred ukončením opýta na uloženie. Uzavrieť súbor (program) bez ukončenia TP možno stlačením Alt+F3 alebo kliknutím na zeleno-žltý štvorček v ľavom hornom rohu okna (v editore).

Ďalšia verzia programu POZDRAV demonštruje použitie príkazu ClrScr (Clear Screen), slúžiaceho na zmazanie obrazovky. Príkaz clrscr nepatrí do štandardu TP, nachádza sa v jednotke (v unite) Crt, ktorá obsahuje rozširujúce príkazy pre prácu s obrazovkou, klávesnicou, kurzorom, zvukom atď. Jednotlivé príkazy a funkcie unitu crt si môžeme prezrieť po vyvolaní kontextového helpu (po stlačení Ctrl+F1 s kurzorom v slove crt) a výbere položky Crt – Procedures. Dôležité je vedieť, že pri použití príkazov z unitu musíme hneď za menom programu uviesť uses (použi) a meno unitu, v našom prípade crt. Do programu sme vložili aj poznámky, ktoré sa v TP vkladajú medzi množinové zátvorky – prekladač všetko medzi { a }

ignoruje[1]. Rovnakú funkciu plnia aj okrúhle zátvorky s hviezdičkami, t.j. zápisy (* a *).

```
program POZDRAV_2;
uses crt; { unit-jednotka obsahujuca prikazy rozsirujuce standard Pascalu }
begin
clrscr;   { prikaz na zmazanie obrazovky, nachadza sa v jednotke crt }
write('Ahoj!');
end.
```

Tretia verzia programu odstraňuje nepohodlné prepínanie medzi užívateľskou obrazovkou a editorom. Príkaz vstupu readln (podrobnejšie sa ním budeme zaoberať neskôr) zastaví beh programu, môžeme si pozrieť výstup na užívateľskej obrazovke, a až po stlačení klávesu Enter program pokračuje resp. skončí, keďže nasleduje end s bodkou (bodka signalizuje prekladaču: koniec programu).

```
program POZDRAV_3;
uses crt;
begin
clrscr;
write('Ahoj!');
readln    { pocitac caka na stlacenie klavesu Enter,
            pri verzii TP7 mozete pouzit aj prikaz readkey }
end.
```

V ďalšej verzii programu POZDRAV chceme, aby sa nás počítač opýtal na meno a slušne nás pozdravil: „Ahoj, naše meno !“. Nemôžeme zvoliť konkrétne meno, lebo sa všetci nevoláme rovnako. Preto musíme použiť premennú, ktorá bude nadobúdať hodnoty raz Jano, potom Miša, Peter, Alexandra,... Počítač musí mať v pamäti vyhradené miesto pre hodnotu každej premennej (ako by si ju ináč pamätal?). Aby vedel, že má rezervovať pamäťové miesto a aké veľké, musíme mu to povedať. Robí sa to tzv. deklaráciou premenných, ktorá začína slovkom var (variable – premenná). Cez meno premennej sa odvolávame na priradené pamäťové miesto a jeho veľkosť určujeme typom premennej. Ak je hodnotou premennej celé číslo, hovoríme o type integer, ak reálne číslo, hovoríme o type real. O údajových typoch si povieme neskôr. Hodnotou našej premennej MENO je reťazec znakov, anglicky string. Pri použití typu string sa v pamäti vyhradí miesto až pre 255 znakov (má niekto dlhšie krstné meno, asi nie).

```
program POZDRAV_4;
uses crt;
var  MENO:string;
begin
clrscr;
writeln('Ako sa volas ?');
readln(MENO);
writeln('Ahoj, ',MENO,'!');
readln
end.
```

Posledná verzia programu POZDRAV je určená pre tých, ktorí si chcú vyskúšať príkaz GotoXY z unitu Crt. Potrebujete vedieť, že obrazovka má štandardne 80 znakov v riadku (stĺpcov v smere x) a 25 riadkov na strane (v smere -y).

```
program POZDRAV_5;
uses crt;
var  MENO:string;
```

[1] Dopustili sme sa malej nepresnosti, lebo keby prvým znakom za zátvorkou bol $, prekladač by očakával tzv. direktívu.

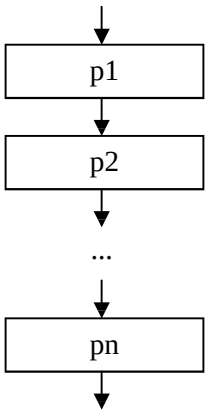
```
begin
clrscr;
writeln('Ako sa volas ?');
readln(MENO);
clrscr;
gotoxy(35,12);
writeln('Ahoj, ',MENO,'!');
readln
end.
```

To bola odbočka do prostredia Turbo Pascalu a k prvému programu a teraz späť k algoritmickým konštrukciám.

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 TP:

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.

Každá z verzií programu POZDRAV (príklad S-1) obsahuje príkazy, ktoré sa po spustení programu vykonajú postupne za sebou, ako sú zapísané. Príklad S-1 je príkladom na sekvenciu.

Príkaz priradenia

Slúži na priradenie hodnoty premennej.

Má tvar: **p := v** kde p je premenná a v výraz; symbol „:=“ sa číta „priraď“

Vykonanie: vyhodnotí sa výraz na pravej strane a jeho hodnota sa priradí premennej na ľavej strane príkazu priradenia.

Príklady: A := 7; MENO := 'Peter'; I := 5; I := I + 1; OBSAH := A*A (hviezdička je symbol pre násobenie)

Je dobré predstaviť si, čo sa v skutočnosti v počítači udeje. Napríklad pre príkaz I := I + 1. Premenná I zaberá v pamäti určité pamäťové miesto, znázornime si ho takto: I [ ] Tu je zapamätaná hodnota premennej I. Po spustení programu tam môže byť náhodné číslo, v Turbo Pascale vo verzii 7 tam je automaticky vložená 0. V našom prípade nech tam je hodnota 5, t.j. I [5]

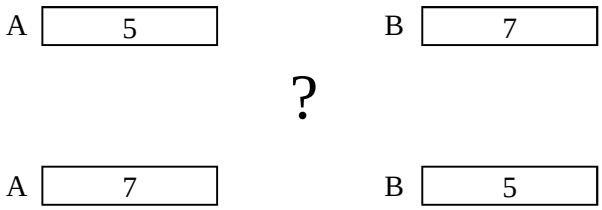
Príkaz I := I + 1 zoberie hodnotu premennej I, t.j. 5, zväčší ju o 1 na 6 a túto hodnotu zapíše do premennej na ľavej strane príkazu priradenia, teda do I [6] . Predchádzajúca hodnota 5 je nenávratne stratená!

Aj keď to je asi najjednoduchší a najčastejší príkaz, dobre si ho premyslite. Zapamätajte si, že sa vykonáva sprava doľava a predchádzajúca hodnota premennej na ľavej strane sa prepíše novou hodnotou.

Príkaz priradenia má rovnaký tvar v algoritmizácii aj v Pascale, tu sa však vyžaduje, aby premenná p a výraz v boli rovnakého typu, čo je pochopiteľné, lebo ťažko by sa napríklad výraz typu string (reťazec znakov) priraďoval premennej typu integer (celé čísla); dovolená je len jedna výnimka: celé číslo (integer) môže byť priradené premennej pre reálne čísla (typu real).

Príklad: Napíšte postupnosť (sekvenciu) príkazov priradenia, ktoré vymenia hodnoty dvoch premenných. Napríklad premenná A nech má hodnotu 5 a premenná B hodnotu 7. Po vykonaní sekvencie nech je v A hodnota 7 a v B hodnota 5.

Analýza:



Aj keď nechceme čitateľa podceňovať, sekvencia

```
A := B;
B := A;
```

nie je správna (odsimulujte si, čo sa udeje v počítači). Ak si uvedomíte prečo, správne riešenie ľahko zistíte. Vykonaním príkazu A := B sa prepíše pôvodná hodnota premennej A (5), ktorú však ešte potrebujeme. Riešením je odložiť si ju skôr, než dôjde k jej prepísaniu, niekde „bokom“, do pomocnej premennej napr. POM. Správna sekvencia: POM := A; A :=B; B := POM;

Poznámky:

1. Úloha má ešte jedno analogické riešenie (POM := B;...).
2. Ak hodnoty premenných A a B sú čísla, nemusíme použiť ani tretiu premennú (napr.: A := A-B; B := B+A; A := B-A;).
3. Úloha výmeny hodnôt dvoch premenných je analogická s úlohou výmeny tekutín (napr. voda a mlieko) v dvoch pohároch navzájom.

Príklad S-2: Vytvorte algoritmus a program na výpočet obsahu a obvodu obdĺžnika.

Analýza: Algoritmus často rieši určitý problém z nejakej vednej disciplíny, v tomto prípade z matematiky. Preto treba mať znalosti z daného vedného odboru, v našom prípade poznať vzorce na výpočet obsahu a obvodu obdĺžnika. Ďalej je dobré si uvedomiť, čo musíme poznať pre daný výpočet (rozmery obdĺžnika - vstupné údaje) a čo očakávame, že vykonaním algoritmu získame (obsah a obvod - výstupné údaje). Keďže základnou vlastnosťou algoritmu je hromadnosť, vstupné aj výstupné údaje zadávame cez premenné. Pre rozmery obdĺžnika sú to bežne písmená A a B, pre obsah a obvod si zvolíme názvy premenných OBSAH a OBVOD. Pri zostavovaní algoritmu uvažujte, ako by ste daný problém riešili bez počítača („manuálne“) a tento postup zapíšte vo forme algoritmu.

manuálne	algoritmus	program

Začínam
potrebujem vedieť rozmery obdĺžnika
použijem vzorec na výpočet obsahu S = a.b
použijem vzorec na výpočet obvodu
O = 2.(a+b)
oznámim výsledok
skončil som.
	začiatok
	čítaj (A, B);
	OBSAH := A*B;
	
	OBVOD := 2*(A+B);
	píš (OBSAH, OBVOD);
	koniec.
		begin
		readln(A, B);
		OBSAH := A*B;
		
		OBVOD := 2*(A+B);
		write(OBSAH, OBVOD);
		end.

Program treba doplniť o deklaráciu premenných. V prvej alternatíve budeme počítať len s celými číslami (typ integer), v druhej s reálnymi (typ real). Programy sú doplnené o užívateľsky komfort, pod ktorým rozumieme predovšetkým pred každý príkaz vstupu read dať príkaz výstupu write, ktorý vypíše na obrazovke, čo treba zadať a tiež jednoznačný výstup výsledných hodnôt.

Program očakáva vloženie dvoch celých čísel, ktoré môžeme vložiť naraz, oddelené medzerou (nie čiarkou!) alebo napíšeme jedno, stlačíme Enter a potom druhé a Enter.

```
program OBDLZNIK_CELE;
uses crt;
var A, B, OBSAH, OBVOD: integer;
begin
clrscr;
write('Zadaj strany obdlznika (cele cisla): ');
readln(A,B);
OBSAH:=A*B;
OBVOD:=2*(A+B);
writeln('Obsah obdlznika = ',OBSAH);
writeln('Obvod obdlznika = ',OBVOD);
readln
end.
```

Typickou chybou začiatočníkov je, že po odladení programu (odstránení chýb, ktoré nájde prekladač TP) už ďalej „slepo veria počítaču“ a program ďalej netestujú. Musíte sa prinútiť po napísaní programu ho otestovať, t.j. zadať čo najviac takých vstupných hodnôt, pre ktoré poznáte výsledky a tiež vložiť tzv. hraničné hodnoty, ktoré sú na hranici dovolených vstupných hodnôt alebo pri ktorých sa mení charakter výpočtu. Program OBDLZNIK by sme mohli otestovať pre A a B rovné 0, 1 a napríklad 2 a 4 (2.4=8 a 2.(2+4)=12) alebo iné čísla, pre ktoré si ľahko spamäti zistíme správne výsledky. Skúste aj A = 1000 a B = 1000. Ak máte zapnuté kontroly chýb počas behu programu (ako sme to odporučili na strane 6), počítač ukončí výpočet chybovým hlásením Error 215: Arithmetic overflow – preplnenie, výsledkom výpočtu je číslo väčšie ako preň vyhradené pamäťové miesto. Prečo je tomu tak, si povieme pri údajových typoch na nasledujúcej strane. Ak kontrolu nemáte zapnutú, možno si zlý výsledok 16 960 (pre obsah) ani nevšimnete! Takže dôveruj ale preveruj!

```
program OBDLZNIK_REALNE;
uses crt;
var A, B, OBSAH, OBVOD: real;
begin
clrscr;
write('Zadaj strany obdlznika: ');
readln(A,B);
OBSAH:=A*B;
OBVOD:=2*(A+B);
writeln('Obsah obdlznika s presnostou na stotiny = ',OBSAH:4:2);
writeln('Obvod obdlznika s presnostou na stotiny = ',OBVOD:4:2);
readln
end.
```

Reálne čísla vkladáme podobne ako celé, len ich zápis môže obsahovať aj desatinnú bodku. Znova všetkému nerozumiete, ale naraz sa všetko nedá. Tie tajomné čísla :4:2 objasníme pri príkaze write.

Údajové typy

Jednoduché štandardné údajové typy:

Meno typu: **Integer** (ordinálny typ)

Množina hodnôt: celé čísla z intervalu –32 768 po 32 767 (konštanta MaxInt)

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:

V TP sú pre hodnoty typu integer vyhradené v pamäti 2 B, t.j. 16 bitov. V jednom bite sa zapamätáva znamienko (0+/1-) a do zvyšných pätnástich bitov sa ukladá hodnota bez znamienka pomocou 0 a 1. Všetkých možností je 2^{15} = 32 768. Preto do daného pamäťového miesta možno uložiť všetky nezáporné celé čísla od 0 po 32 767 alebo záporné celé čísla od –1 po – 32 768. Ak je výsledkom operácie s typom integer číslo mimo tento rozsah, nastane v príklade S-2 opísaná chyba, tzv. preplnenie (v Options – Compiler... prepínač Overflow Checking – kontrola preplnenia). Ak nám uvedený rozsah nevyhovuje, môžeme použiť ďalšie celočíselné typy: Shortint, Longint, Byte a Word, ktorých rozsahy si môžete pozrieť pomocou kontextového helpu (Ctrl+F1 s kurzorom pod napr. integer). Najväčší rozsah má typ longint, kde konštanta MaxLongInt má hodnotu 2 147 483 647 (= 2^{31} –1).

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 = C a A mod B = D potom B.C + D = A a 0<=D<B. Pre záporné celé čísla funkcie div a mod v TP pracujú ináč ako v matematike!

Výsledkom: 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 zaokrúhlené reálne číslo x, napr. round(8.9) = 9, round(-5.6) = -6, round(0.5) = 1
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 približne $-1{,}7.10^{38}$ po $1{,}7.10^{38}$

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.

Ďalšie údajové typy pre reálne čísla sú single, double, extended a comp, ich použitie je však podmienené prítomnosťou alebo emuláciou matematického koprocesora (funkcie v F10 – Options - Compiler... – Numeric processing).

Niektoré funkcie sú popísané vyššie, u typu integer, a málo používané funkcie nepopíšeme.

Výsledkom: 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

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, 0 a True – pravda, nie 0; false < true.

pred(true) = false, succ(false) = true; ord(false) = 0, ord(true) = 1

Meno typu: **Char** (ordinálny typ)

Množina hodnôt: znaky tabuľky ASCII

Funkcie: upcase(z)

succ(z), pred(z), ord(z), chr(x)

Relačné operátory: <, <=, =, <>, >=, >, in

Vysvetlivky:

Znaky tabuľky ASCII aj s poradovými číslami možno zobraziť pomocou programu:

```
program ZNAKY_ASCII;
uses crt;
var I:byte;
begin
clrscr;
for I:=32 to 255 do
  write(I:6,chr(I):2);              { vypis v tabulke }
```

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

```
readln;
clrscr;
for I:=1 to 255 do
  begin
  writeln(I:6,chr(I):4);           { vypis pod sebou }
  if I mod 23 = 0 then readln      { zastavi vypis vzdy po 23 riadkoch }
  end
end.
```

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.

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 TP 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.

Štruktúrovaný údajový typ:

Meno typu: **string** alebo **string[n]** kde n je celé číslo od 1 po 255 – udáva maximálnu dĺžku reťazca

Množina hodnôt: reťazce zo znakov ASCII

Dovolené operácie: + (spája reťazce)

Funkcie: length, copy, delete, concat, insert, pos

Relačné operátory: <, <=, =, <>, >=, >

Vysvetlivky:

String vyhradí v pamäti miesto pre 255 znakový reťazec (pre najviac 255 znakov), napríklad string[5] vyhradí v pamäti miesto pre päť znakový reťazec (pre najviac 5 znakov, ostatné nebudú zapamätané).

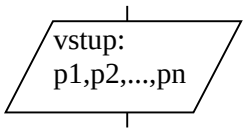
Výsledkom funkcie length(r) je celé číslo - dĺžka reťazca r. Prázdny reťazec '' má dĺžku 0.

Pri relačných operáciách sa najprv porovnajú prvé znaky v porovnávaných reťazcoch, ak sú zhodné, druhé atď.

Príkazy vstupu, príkaz read

Príkazy vstupu slúžia na vstup údajov z klávesnice.

Majú tvar:



čítaj (p1, p2, ... , pn)

kde p1 až pn sú premenné

V TP sú to modifikácie príkazu **read**:

read (p1, p2, ... , pn)
readln (p1, p2, ... , pn) kde p1 až pn sú premenné
readln

Vykonanie: zastaví sa beh programu a počítač čaká na vloženie príslušného počtu hodnôt v stanovenom poradí a deklarovaných typov.

Napríklad: readln(A,B,C); readln(TEXT); readln(X); readln(MENO); readln(ODPOVED); readln;

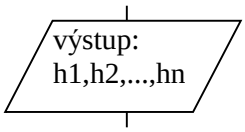
Poznámky:

- príkazy ukončené ln (line) možno interpretovať: načítaj hodnoty premenných p1 až pn a nastav čítanie na nový riadok
- jednotlivé číselné hodnoty môžu byť oddelené medzerami alebo vkladané v samostatných riadkoch zadaním hodnoty a stlačením klávesu Enter
- „prázdny“ príkaz readln neočakáva hodnotu, len stlačenie klávesu Enter
- príkaz read(p1, ... , pn) nebudeme predbežne používať
- po zadaní väčšieho čísla, ako je dovolený rozsah daného typu, sa zobrazí chybové hlásenie Error 201: Range check error - prekročenie rozsahu (v Options – Compiler... prepínač Range Checking – kontrola rozsahu hodnôt). Napríklad u typu integer po zadaní čísla väčšieho ako je maxint (32 767). Ak nedošlo k zobrazeniu chybového hlásenia, pozri str. 6
- ak pri zadávaní hodnôt zadaná hodnota nezodpovedá očakávanému typu, napr. u číselného typu vložíme miesto číslice písmeno, zobrazí sa chybové hlásenie Error 106: Invalid numeric format – zlý číselný formát (v Options – Compiler... prepínač I/O Checking – kontrola vstupno/výstupných operácií)

Príkazy výstupu, príkaz write

Príkazy výstupu slúžia na zobrazenie výstupných hodnôt na obrazovke monitora.

Majú tvary:



píš (h1, h2, ... , hn)

kde h1 až hn sú výstupné hodnoty

V TP sú to modifikácie príkazu write:

write (f1, f2, ... , fn)
writeln (f1, f2, ... , fn) kde f1 až fn sú formátovacie výrazy
writeln

Vykonanie: na obrazovke monitora sa zobrazia výstupné hodnoty v predpísaných formátoch (buď hodnoty premenných alebo text pôvodne uvedený v apostrofoch).

Napríklad: writeln('Dĺžka čiary je ',DLZKA); write('Zadaj kladné číslo: '); writeln(A,' ',B); writeln;

Poznámky:

- príkazy ukončené ln (line) možno interpretovať: zobraz výstupné hodnoty a nastav výstup (kurzor) na začiatok nového riadku
- „prázdny“ príkaz writeln nastaví výstup na nový riadok

Formátovacie výrazy:

1) h — h je výraz reprezentujúci výstupnú hodnotu
2) h : pzv — h ako vyššie, pzv je výraz reprezentujúci počet znakov výstupu
3) h : pzv : pdm — h a pzv ako vyššie, pdm je výraz reprezentujúci počet desatinných miest výstupu u typu real

Napríklad:

a) nech A je premenná typu integer

hodnota A	príkaz	na obrazovke	poznámka
12345	write(A)	12345	znamienko plus sa nezobrazuje
-12345	write(A)	-12345	znamienko mínus zaberá jeden znak
12345	write(A:10)	12345	zľava sa doplní päť medzier
-12345	write(A:7)	-12345	zľava sa doplní jedna medzera
12345	write(A:3)	12345	ignoruje počet znakov výstupu

b) nech X je premenná typu real (matematický koprocesor vypnutý)

hodnota X	príkaz	na obrazovke	poznámka
123,456789	write(X)	1.2345678900E+02	tzv. semilogaritmický tvar, E – exponent sa číta „krát desať na“, pre „+“ vynechaný znak
123,456789	write(X:10)	1.235E+02	neprehliadnite zaokrúhlenie mantisy
123,456789	write(X:2)	1.2E+02	minimálny výpis s exponentom (8 znakov)
123,456789	write(X:6:2)	123.46	6 znakov výstupu, 2 za des. bodkou
123,456789	write(X:4:2)	123.46	ignoruje počet znakov výstupu
123,456789	write(X:10:3)	123.457	zľava sa doplnia tri medzery
123,456789	write(X:12:10)	123.4567890000	maximálna presnosť zobrazenia
123,456789	write(X:5:0)	123	zľava sa doplnia dve medzery
0,0000987654321	write(X)	9.8765432100E-05	semilogaritmický tvar
0	write(X)	0.0000000000E+00	semilogaritmický tvar

c) nech S je premenná typu string

hodnota S	príkaz	na obrazovke	poznámka
Janosik	write(S)	Janosik	
Janosik	write(S:3)	Janosik	ignoruje počet znakov výstupu
Janosik	write(S:10)	Janosik	zľava sa doplnia tri medzery

d) nech Q je premenná typu boolean

hodnota Q	príkaz	na obrazovke	poznámka
true	write(Q)	TRUE	
false	write(Q:10)	FALSE	zľava sa doplní päť medzier
true	write(Q:1)	TRUE	ignoruje počet znakov výstupu

Poznámky:

- neprehliadnite zaokrúhľovanie reálnych čísel na zadaný počet desatinných miest
- bez použitia formátovacích výrazov napríklad v príkaze write(A,B), kde A a B sú typu integer, neodlíšite od seba cifry kladných čísel! Okrem formátovania je riešením aj vloženie medzery, t.j. write(A,' ',B).
- formátovacie výrazy sú výrazy, t.j. môžu mať aj tvar napríklad writeln(A*B), write(X : PRESNOST+2 : PRESNOST), kde premenná PRESNOST typu integer môže nadobúdať hodnoty 0 (jednotky), 1 (desatiny), 2 (stotiny), ... , 10
- ujasnite si vykonanie príkazu writeln('Obvod = ':40,2*(A+B):4:2)

Vráťme sa však k sekvencii.

Príklad S-3: Vytvorte program simulujúci palubný počítač auta. Známy je objem palivovej nádrže a priemerná spotreba paliva počas jazdy. Vždy pri natankovaní plnej nádrže sa vynuluje tzv. denné počítadlo kilometrov. Po zadaní prejdených kilometrov (z denného počítadla kilometrov) počítač vypočíta aktuálny dosah auta. Po zadaní vzdialenosti do cieľa vypočíta priemernú dobu jazdy.

Analýza: Dosah auta pri plnej nádrži sa vypočíta ako podiel: objem palivovej nádrže / priemerná spotreba paliva na 100 km, krát 100 km. Keďže tento podiel nemusí byť celé číslo, zaokrúhlime ho na celé km, samozrejme nadol. Aktuálny dosah sa zrejme rovná: dosah pri plnej nádrži – prejdené km. Priemerná doba jazdy sa vypočíta ako podiel: vzdialenosť do cieľa / priemerná rýchlosť za hodinu (konštanta). Výsledok je v hodinách a zobrazíme ho s presnosťou na desatiny hodiny.

```
program POCITAC;
uses  crt;
const ObjemPalNadrze = 46;    { litrov }
      PriemSpotPaliva = 6.5;  { litra na 100 km }
      PriemRychlost = 90;     { km/hod. }
var   PrejdeneKM, AktDosah, Dosah, VzdDoCiela : integer;
                             PriemDobaJazdy : real;
begin
clrscr;
write('Prejdene kilometre (z denneho pocitadla km): ');
readln(PrejdeneKM);
Dosah:=trunc(ObjemPalNadrze/PriemSpotPaliva*100);
AktDosah:=Dosah-PrejdeneKM;
writeln('Aktualny dosah: ',AktDosah,'km');
writeln;
write('Vzdialenost do ciela: ');
readln(VzdDoCiela);
PriemDobaJazdy:=VzdDoCiela/PriemRychlost;
writeln('Priemerna doba jazdy: ',PriemDobaJazdy:3:1,' hod.');
readln
end.
```

Príklad S-4: Vytvorte program na „uhádnutie“ celého čísla z intervalu 0 až 100, ak viete, že po zadaní zvyškov po delení 3 (zvysok3), 5 (zvysok5) a 7 (zvysok7) mysleného čísla sa myslene číslo rovná zvyšku po celočíselnom delení výrazu (70*zvysok3+21*zvysok5+15*zvysok7) 105-timi. Strašné, ani len zadaniu sa nedá rozumieť!

Analýza: Treba zadať zvyšky po delení 3, 5 a 7 mysleného čísla – to sú vstupné údaje a my pre ne zvolíme vstupné premenné označené zvysok3, zvysok5 a zvysok7. Myslené číslo sa rovná zvyšku po delení výrazu...,

preto si tento výraz vypočítajme. VYRAZ = 70*zvysok3+21*zvysok5+15*zvysok7. Myslené číslo sa rovná zvyšku po celočíselnom delení VYRAZu 105-timi. Zvyšok po celočíselnom delení nám dáva funkcia mod. A tu je program:

```
program UHADNEM_CISLO;
uses crt;
var zv3,zv5,zv7,myslene_cislo:byte;
begin
clrscr;
writeln('U H A D N E M   M Y S L E N E   C I S L O  D O  100 !':60);
write('Zvysok po deleni 3: '); readln(zv3);
write('Zvysok po deleni 5: '); readln(zv5);
write('Zvysok po deleni 7: '); readln(zv7);
myslene_cislo:=(70*zv3+21*zv5+15*zv7) mod 105;
writeln;
writeln('Myslel si si cislo ',myslene_cislo);
readln
end.
```

Príklad S-5: Vytvorte program, ktorý prepočíta čas v sekundách na hodiny : minúty : sekundy.

Analýza: Vieme, že hodina má 3 600 sekúnd. Koľkokrát sa 3 600 nachádza v sekundách (bezo zvyšku), je hľadaný počet hodín – to vie vypočítať funkcia div. Funkcia mod určí zvyšok sekúnd, čo zostal na minúty a sekundy. Počet minút zistíme opäť celočíselným delením zvyšných sekúnd číslom 60. No a to čo zostane, sú už len sekundy (< 60). Výpočet funkciou mod možno nahradiť aj príkazom CAS := CAS – 3600*H resp. CAS := CAS – 60*M.

```
program CAS_V_SEKUNDACH;
uses crt;
var  H, M, CAS : longint;
begin
clrscr;
write('Zadaj cas v sekundach: ');
readln(CAS);
H:=CAS div 3600; CAS:=CAS mod 3600;
M:=CAS div 60;   CAS:=CAS mod 60;
writeln(H,' : ',M,' : ',CAS);
readln
end.
```

Úloha: Dokážete upraviť predchádzajúci program tak, aby počítal iba s jednou premennou CAS?

```
program CAS_V_SEKUNDACH_2;
uses crt;
var  CAS:longint;
begin
clrscr;
write('Zadaj cas v sekundach: '); readln(CAS);
gotoxy(34,12);
writeln(CAS div 3600,' : ',CAS mod 3600 div 60,' : ',CAS mod 3600 mod 60);
readln
end.
```

Neriešené úlohy na sekvenciu:

1. Vytvorte program na výpočet obsahu a obvodu štvorca s presnosťou na desatiny.
   Návod: Vzorec pre výpočet obsahu $S = a^2$ a obvodu $O = 4a$.
2. Vytvorte program na výpočet obsahu a obvodu kruhu s presnosťou na tisíciny.
   Návod: Vzorec pre výpočet obsahu $S = \pi.r^2$ a obvodu $O = 2\pi.r$. TP pozná konštantu PI = 3,1415...
3. Vytvorte program na výpočet objemu, povrchu a telesovej uhlopriečky kvádra s presnosťou na stotiny.
   Návod: Vzorec pre výpočet objemu $V = a.b.c$, povrchu $S = 2(a.b+a.c+b.c)$, uhlopriečky $u = \sqrt{a^2+b^2+c^2}$.
4. Vytvorte program na výpočet objemu a povrchu gule s najväčšou možnou presnosťou.
   Návod: Vzorec pre výpočet objemu $V = 4/3\pi.r^3$, povrchu $S = 4\pi.r^2$.
5. Vytvorte program na výpočet počtu 10, 5, 2 a 1 Sk mincí na vyplatenie zadanej sumy (použite funkcie div a mod).
   Poznámka: Pozri program CAS_V_SEKUNDACH.
6. Vytvorte program na nákup valút za určitú sumu korún. Zmenáreň si účtuje manipulačný poplatok vo výške 1% z vloženej sumy. Skutočne platená čiastka sa zaokrúhľuje na celé koruny. Názov a kurz meny si zvoľte sami.
7. Priemerná denná teplota sa počíta ako aritmetický priemer teplôt nameraných o 6-tej ráno, o 12-tej a o 18-tej hodine, pričom údaj nameraný o 18-tej hodine sa započítava dvakrát. Vytvorte program, ktorý zo zadaných údajov vypočíta priemernú dennú teplotu.
8. Sú zadané odpory dvoch rezistorov v ohmoch. Určte hodnotu výsledného odporu pri ich sériovom a paralelnom zapojení.
   Návod: Pri sériovom zapojení $R = R_1+R_2$, pri paralelnom $R = R_1.R_2 / (R_1+R_2)$.
9. Vytvorte program na riešenie lineárnej rovnice $ax + b = 0$ a, b reálne, $a \neq 0$.
   Návod: Koreň $x = -b/a$.
10. Vytvorte program na prevod veľkosti uhla v radiánoch na stupne a minúty.
    Návod: π radiánov = 180^0, 1^0=60'. TP pozná konštantu PI.

VETVENIE, ROZHODOVANIE

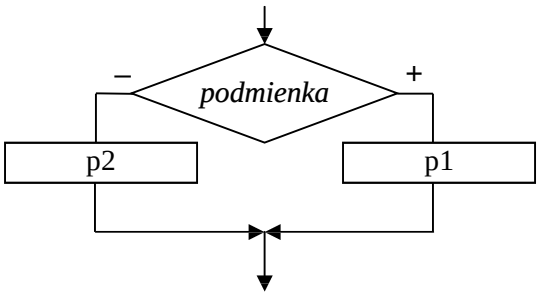
Vetvenie použijeme, ak vykonanie príkazu (alebo skupiny príkazov) je podmienené splnením alebo nesplnením určitej podmienky.

S rozhodovaním sa stretávame v bežnom živote na každom kroku. Ak bude teplo, pôjdem na kúpalisko, inak budem sedieť za počítačom. Vykonanie akcie „pôjdem na kúpalisko“ je podmienené splnením podmienky „bude teplo“. Vykonanie akcie „budem sedieť za počítačom“ je podmienené nesplnením podmienky!

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

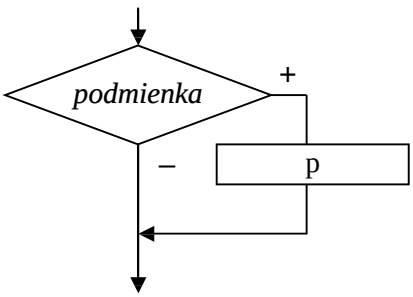
if b
then p1
else p2 kde b je výraz typu boolean, p1 a p2 sú príkazy

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

Napríklad:
```
if X < 0 then writeln('Číslo je záporné') else writeln('Číslo je nezáporné')
if PRIESTUPNY_ROK then DNI := 29 else DNI := 28
if (ZNAK >= 'A') and (ZNAK <= 'Z') then writeln('veľké písmeno') else writeln('neviem')
if odd(CISLO) then writeln('nepárne') else writeln('párne')
if true then writeln('vždy toto') else writeln('toto nikdy')
if A mod B = 0 then writeln(A,' je deliteľné ',B) else writeln(A,' nie je deliteľné ',B)
```

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 TP 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.

Napríklad: if MENO = 'Peter' then writeln('Ďalší Peter')

if CISLO > NAJVACSIE then NAJVACSIE := CISLO

if (CISLO > MaxInt) or (CISLO <= -MaxInt) then writeln('mimo rozsah integer')

if ord(ZNAK) < 32 then writeln('riadiaci znak')

if X = HLADANE_CISLO then POCET := POCET + 1

Príklad VIF-1: Vytvorte program na nájdenie najväčšieho z troch celých čísel.

Analýza: Znova odporúčame algoritmus tohto problému zostaviť obyčajným „sedliackym“ rozumom. Čo spravíme prvé, keď sa dozvieme čísla, z ktorých treba nájsť najväčšie? Asi porovnáme prvé dve čísla a z nich vyberieme väčšie (ak sú rovnaké, je jedno, ktoré vyberieme) a to následne porovnáme s tretím číslom. A máme najväčšie – maximum. Algoritmus je na strane 4.

```
program MAXIMUM_1;
uses crt;
var A, B, C, MAX : integer;
begin
clrscr;
write('Zadaj tri cele cisla: ');
readln(A,B,C);
if A>B
  then MAX:=A
  else MAX:=B;
if C>MAX
  then MAX:=C;
writeln('Z cisel ',A,', ',B,', ',C,' je najvacsie ',MAX,'.');
readln
end.
```

Aby sme demonštrovali variabilnosť riešenia už tak jednoduchého problému, uvádzame ďalšie programy riešiace rovnakú úlohu.

Iná možnosť (jej výhodou je jednoduchosť a použiteľnosť pre ľubovoľný počet čísel, treba len opakovať príkaz if ? > MAX then MAX := ?):

```
program MAXIMUM_2;
uses crt;
var A, B, C, MAX : integer;
begin
clrscr;
write('Zadaj tri cele cisla: ');
readln(A,B,C);
MAX:=A;      { priradenie pociatocnej hodnoty premennej MAX }
if B>MAX
  then MAX:=B;
if C>MAX
  then MAX:=C;
writeln('Z cisel ',A,', ',B,', ',C,' je najvacsie ',MAX,'.');
readln
end.
```

V nasledujúcom programe je logická chyba, viete ju nájsť?

```
program MAXIMUM_3;
uses crt;
var A, B, C : integer;
begin
clrscr;
write('Zadaj tri cele cisla: ');
readln(A,B,C);
write('Z cisel ',A,', ',B,', ',C,' je najvacsie ');
if (A>B) and (A>C)        { preco sme presunuli vystup (v nom nie je chyba!)? }
  then writeln(A);
if (B>A) and (B>C)
  then writeln(B);
if (C>A) and (C>B)
  then writeln(C);
readln
end.
```

Takto to je už dobre?

```
program MAXIMUM_4;
uses crt;
var A, B, C : integer;
begin
clrscr;
write('Zadaj tri cele cisla: ');
readln(A,B,C);
write('Z cisel ',A,', ',B,', ',C,' je najvacsie ');
if (A>B) and (A>C)
  then writeln(A)
  else if (B>A) and (B>C)
         then writeln(B)
         else writeln(C);
readln
end.
```

Programátorský štýl

Ak sa pozriete na zápisy príkazov if aj celých programov, vidíte v nich určitú štruktúru, ktorou sa snažíme vyjadriť podriadenosť niektorých častí nad inými. Rovnocenné príkazy sú na jednej úrovni, podriadené časti sa snažíme posunúť napríklad o tri medzery vpravo. Zároveň takýto zápis sprehľadňuje celý algoritmus alebo program. Veľmi vám odporúčame používať určité logické schémy pri zápisoch programov a algoritmov. Budú pre všetkých prehľadnejšie, zrozumiteľnejšie, pri tvorbe spravíte menej chýb a ľahšie sa v nich budú hľadať chyby. Odporúčame dve schémy:

1. zápis if podmienka
   then príkaz1
   else príkaz2

   dobre graficky znázorňuje, že ide o úplné binárne vetvenie s oboma vetvami na samostatných riadkoch; riadky s then a else sú podriadené riadku if, kde príkaz začína, ďalší príkaz by bol už na úrovni if;

2. zápis if podmienka then príkaz1
   else príkaz2

   zdôrazňuje prítomnosť vetvy else, t.j. úplného binárneho vetvenia, vetva then je „schovaná“, keďže sa musí vyskytovať v každom príkaze if.

Pri písaní programov v editore TP si všimnite, že podporuje štruktúrovaný zápis.

Prakticky prekladaču TP je jedno, ako je daný program napísaný. Pre neho je rozhodujúca len prítomnosť bodkočiarok medzi jednotlivými príkazmi. Celý program by mohol byť napísaný v jednom riadku (ak by sa tam zmestil). Súčasťou dobrého programátora je aj dobrý programátorský štýl, aj čo sa týka zápisu programov, ktorý sa určite neprezentuje programom v jednom stĺpci alebo riadku!

A propos bodkočiarky. Platí: príkazy sa od seba oddeľujú bodkočiarkami. Preto za begin, pred end a inými vyhradenými slovami bodkočiarky nemusia byť, neoddeľovali by príkazy. V niektorých prípadoch sú dokonca chybou (uvidíme pri cykloch). My v programoch na miestach, kde nemusia byť, nebudeme dávať bodkočiarky, aj keď väčšinou by tam mohli byť.

Zložený príkaz

Ak si dobre všimnete príkaz if, za then aj else je dovolený len jeden (!) príkaz. Ak potrebujeme na týchto miestach použiť viacej príkazov, musíme vytvoriť tzv. zložený príkaz.

Zložený príkaz použijeme na mieste, kde je dovolený len jeden príkaz a my potrebujeme, aby došlo k vykonaniu viacerých príkazov. Zložený príkaz vytvára zo skupiny príkazov jeden príkaz.

Má tvar:	začiatok	v TP:	**begin**	
	p1;		**p1;**	
	p2;		**p2;**	
	...		**...**	
	pn;		**pn;**	
	koniec		**end**	kde p1 až pn sú príkazy

Vykonanie: Z príkazov p1, p2 až pn je vytvorený jeden (zložený) príkaz.

```
Napríklad: if A > B                    if  X<=0
             then begin                  then writeln('Pre X<=0 neviem vypočítať!')
                  POM := A;              else  begin
                  A := B;                      XnaN := exp( N*ln(X) );
                  B := POM;                    writeln( X, ' na ' , N , ' = ' , XnaN )
                  end                         end
           { po skončení A ≤ B }       { výpočet mocniny x^n }
```

Poradie vyhodnocovania operácií (priorita operátorov)

Všimnite si, že v programoch MAXIMUM 3 a 4 sme v podmienkach príkazov if použili zátvorky. Otázka znie: Kedy musíme použiť zátvorky v zápise výrazov a kedy nie? O tom rozhoduje priorita jednotlivých operátorov. Podobne ako v matematike má aj v TP násobenie a delenie „prednosť“ pred sčítaním a odčítaním a pod. Všeobecne platí, že najprv sa vyhodnotia funkcie (abs, sqr, sqrt, odd,...).

Priorita operátorov je nasledovná:

1. unárne mínus (zmena znamienka), not
2. *, / , div, mod, and
3. +, - , or, xor
4. <, <=, =, <>, >=, >, in

pri rovnosti operátorov sa postupuje zľava doprava; prvé sa vyhodnocujú výrazy v zátvorkách, od vnútorných k vonkajším.

Príklady: (I = N) or (ODP = 'K')
boolean boolean

Bez zátvoriek by došlo ku chybe, lebo or má vyššiu prioritu ako = a preto by sa najprv vykonalo N or ODP pričom N je číselný typ a ODP je typu char, čo nie sú typy boolean očakávané „okolo“ or.

Výraz $\frac{a+b}{c}$ musí byť zapísaný pomocou zátvoriek (A+B)/C, inak by došlo najprv k deleniu B/C (delenie má vyššiu prioritu ako sčítanie) a až potom k sčítaniu, čo nezodpovedá pôvodnému výrazu.

Výraz $\frac{a}{b.c}$ musí byť zapísaný A/(B*C), ináč by nezodpovedal pôvodnému výrazu, pretože zápis A/B*C (pravidlo „zľava doprava“) zodpovedá výrazu $\frac{a}{b}.c$ (čo nie je pôvodný výraz). Pôvodnému výrazu však zodpovedá aj zápis A/B/C!

Výraz $\frac{a.b}{c}$ môže byť zapísaný bez zátvoriek A*B/C (pravidlo „zľava doprava“; najprv sa vykoná násobenie a potom delenie).

Znova sme trochu múdrejší v teórii, vráťme sa však k príkladom využívajúcim vetvenie a príkaz if.

Príklad VIF-2: Vytvorte program na usporiadanie troch celých čísel (zostupne).

Analýza: Podľa potreby budeme vymieňať hodnoty premenných A, B, C, kým nebude A<=B<=C.

```
program USPORIADAJ;
uses crt;
var A, B, C, POM : integer;
begin
clrscr;
write('Zadaj tri cele cisla: ');
readln(A,B,C);
if A>B
  then begin POM:=A; A:=B; B:=POM end;
if B>C
  then begin POM:=B; B:=C; C:=POM end;
if A>B
  then begin POM:=A; A:=B; B:=POM end;
writeln(A,' <= ',B,' <= ',C);
readln
end.
```

Venujme sa ešte jednej „teoretickej“ otázke. V príkaze if nič nebráni tomu, aby na mieste príkazu p1 alebo p2 mohol byť znova, podľa potreby, použitý príkaz if, ako to vidieť v programe MAXIMUM_4.

Pri štruktúrach

```
if b1
  then if b2
         then p1
else p2
```

```
if b1
  then  if b2
          then p1
else p2
else p3
```

vzniká otázka, ku ktorému if patria vetvy else?

Platí: Vetva else patrí vždy k najbližšiemu predchádzajúcemu then.

Vyjadrené štruktúrovane:

```
if b1
  then if b2
         then p1
         else  p2
```

```
if b1
  then  if b2
          then p1
          else  p2
  else p3
```

Ak chceme, aby else p2 v prvej štruktúre patrilo prvému then musíme to prekladaču povedať zápisom:

```
if b1
  then begin
         if b2
           then p1
         end
  else  p2
```

Príklad VIF-3: Vytvorte program na výpočet výsledného odporu dvoch rezistorov zapojených sériovo alebo paralelne s presnosťou na stotiny.

Analýza: Z fyziky treba vedieť, že pri sériovom zapojení rezistorov výsledný odpor $R=R_1+R_2$, pri paralelnom zapojení $R = \frac{R_1 . R_2}{R_1 + R_2}$.

Doriešiť treba ešte zadanie spôsobu zapojenia. Rozhodli sme sa pre vloženie prvého písmena (typ char), presnejšie, pri vložení znaku p alebo P je zapojenie paralelné, inak sériové.

```
program VYSLEDNY_ODPOR;
uses crt;
var R1, R2, R : real;
        ZAPOJ : char;
begin
clrscr;
write('Zadaj hodnoty odporov R1 a R2: '); readln(R1,R2);
write('Sposob zapojenia (P - paralelne, iny znak - seriovo): ');
readln(ZAPOJ);
if (ZAPOJ='P') or (ZAPOJ='p')
  then R:=R1*R2/(R1+R2)
  else R:=R1+R2;
writeln('Vysledny odpor R = ',R:4:2);
readln
end.
```

Využitie funkcie upcase a upravenie výstupu:

```
program VYSLEDNY_ODPOR_2;
uses crt;
var R1, R2, R : real;
        ZAPOJ : char;
begin
clrscr;
write('Zadaj hodnoty odporov R1 a R2 v ohmoch: '); readln(R1,R2);
write('Sposob zapojenia (P - paralelne, iny znak - seriovo): ');
readln(ZAPOJ);
ZAPOJ:=upcase(ZAPOJ);
if ZAPOJ='P'
  then R:=R1*R2/(R1+R2)
  else R:=R1+R2;
writeln;
write('Vysledny odpor pri ');
if ZAPOJ='P'
  then write('paralelnom')
  else write('seriovom');
writeln(' zapojeni R = ',R:4:2,' ohmov.');
readln
end.
```

Ďalšou možnosťou je ošetrenie zadania spôsobu zapojenia tak, aby program „zobral“ len S alebo P.

```
program VYSLEDNY_ODPOR_3;
uses crt;
var R1, R2 : real;
     ZAPOJ : char;
begin
clrscr;
write('Zadaj hodnoty odporov R1 a R2 v ohmoch: '); readln(R1,R2);
write('Sposob zapojenia (S - seriovo, P - paralelne): '); readln(ZAPOJ);
if (ZAPOJ='S') or (ZAPOJ='s')
  then writeln('Vysledny odpor R = ',R1+R2:3:1,' ohmov')
  else if (ZAPOJ='P') or (ZAPOJ='p')
         then writeln('Vysledny odpor R = ',R1*R2/(R1+R2):3:1,' ohmov')
         else writeln('Nebolo stlacene S alebo P!');
readln
end.
```

Úloha: Program VYSLEDNY_ODPOR môžete upraviť aj tak, aby reagoval na vloženie celého slova seriovo alebo paralelne (zrejme premenná ZAPOJ bude typu string[9]).

Príklad VIF-4: Vytvorte program, ktorý po zadaní pH oznámi: prostredie kyslé, neutrálne alebo zásadité.

Analýza: Trochu chémie nezaškodí. Tuším nás učili: ak je pH < 7, tak je prostredie kyslé; ak je pH = 7, tak je neutrálne a ak je pH > 7, tak je prostredie zásadité. Použitie podmienených príkazov je zrejmé.

```
program PH_PROSTREDIA;
uses crt;
var PH:real;
begin
clrscr;
write('Zadaj pH: '); readln(PH);
writeln;
write('Prostredie je ');
if PH<7 then writeln('kysle.');
if PH=7 then writeln('neutralne.');
if PH>7 then writeln('zasadite.');
readln
end.
```

Príklad VIF-5: Vytvorte program analogický programu S-5 (prepočíta čas v sekundách na hodiny : minúty : sekundy s výstupom HH : MM : SS (dvojčísla).

Analýza: Rozdiel od príkladu S-5 je v tom, že ak je číselný údaj (hodiny, minúty alebo sekundy) jednociferný, tak treba „predložiť“ nulu.

```
program CAS_V_SEKUNDACH;
uses crt;
var  H,M,CAS:longint;
begin
clrscr;
write('Zadaj cas v sekundach: '); readln(CAS);
H:=CAS div 3600; CAS:=CAS mod 3600;
M:=CAS div 60;   CAS:=CAS mod 60;
if H<10 then write('0');
write(H,' : ');
if M<10 then write('0');
write(M,' : ');
if CAS<10 then write('0');
```

```
writeln(CAS);
readln
end.
```

Príklad VIF-6: Vytvorte program na určenie, či zadaný rok je priestupný.

Analýza: Platí, že rok je priestupný, ak je deliteľný 4 a nie je deliteľný 100, okrem rokov deliteľných 400. Napríklad rok 1600 je priestupný ale rok 1700 nie. Ešte sa „s priestupným rokom“ stretneme.

```
program PRIESTUPNY_ROK;
uses crt;
var ROK:integer;
begin
clrscr;
write('Zadaj rok: '); readln(ROK);
if (ROK mod 4=0) and ((ROK mod 100<>0) or (ROK mod 400=0))
  then writeln('Rok ',ROK,' - priestupny')
  else writeln('Rok ',ROK,' - nepriestupny');
readln
end.
```

Ešte jedno riešenie (toto riešenie ešte použijeme neskôr):

```
program PRIESTUPNY_ROK_2;
uses crt;
var ROK:integer;
    PRIESTUPNY:boolean;
begin
clrscr;
write('Zadaj rok: '); readln(ROK);
PRIESTUPNY:=(ROK mod 100<>0) and (ROK mod 4=0) or (ROK mod 400=0);
if PRIESTUPNY
  then writeln('Rok ',ROK,' - priestupny')
  else writeln('Rok ',ROK,' - nepriestupny');
readln
end.
```

Upravte program tak, aby oznamoval aktuálne: Rok ... bol / je / bude ... Použite príkaz GetDate z unitu Dos (pozri pomocou Ctrl+F1), ktorého výsledkom je aktuálny rok, mesiac, deň a poradové číslo dňa v týždni, pričom poradové číslo nedele je 0, pondelka 1 atď. Použité premenné sú typu word (pozri aj príklad VCS-1: program POCET_DNI).

Príklad VIF-7: Vytvorte program na výpočet doby splácania - počtu rokov a mesiacov, bezúročnej pôžičky po zadaní požičanej sumy a výšky mesačnej splátky.

Analýza: Zovšeobecnite situáciu, keď máte požičaných 12 000 Sk a mesačne splácate po 100 Sk. Ako dlho budete platiť? Najprv nech nás nezaujímajú „zvyšky“ (rozumej mesiace), len celé roky, ktoré treba splácať. Tie určí delenie bezo zvyšku, t.j. funkcia div. Ak ste našli odpoveď, môžeme ísť ďalej. Ak by ste mali požičaných 13 000 Sk, koľko mesiacov by ste museli ešte po desiatich rokoch platiť? Zovšeobecnite!

Myslite aj na možnosť, že splácať nebude treba ani jeden celý rok, potom položka *počet rokov splácania* by sa vôbec nemala zobraziť, podobne s mesiacmi, ak sa všetko splatí za celý násobok roka.

```
program SPLATKY;
uses crt;
var SUMA, MESACNE, ROKOV, MESIACOV, ZOSTAVASK : longint;
begin
clrscr;
```

```
write('Pozicana suma: '); readln(SUMA);
write('Vyska mesacnej splatky: '); readln(MESACNE);
writeln;
writeln('Treba splacat: ');
ROKOV:=SUMA div (12*MESACNE);
if ROKOV>0
  then writeln('- rokov: ',ROKOV);
ZOSTAVASK:=SUMA-ROKOV*12*MESACNE;
MESIACOV:=ZOSTAVASK div MESACNE;
if MESIACOV>0
  then writeln('- mesiacov: ',MESIACOV);
ZOSTAVASK:=SUMA-(ROKOV*12*MESACNE+MESIACOV*MESACNE);
if ZOSTAVASK>0
  then writeln('- dalsi mesiac este splatit: ',ZOSTAVASK,' korun.');
readln
end.
```

Doplňujúca úloha: Výstup s koncovkami v správnom tvare:

Treba splácať:

1 rok alebo 2, 3, 4 roky alebo 5, 6, ... rokov

1 mesiac alebo 2, 3, 4 mesiace alebo 5, 6, ... mesiacov

... splatiť 1 korunu alebo 2, 3, 4 koruny alebo 5,6, ...korún.

Príklad VIF-8: Na pobavenie a doplnenie vôbec prvého programu v skriptách. Program, ktorý po zadaní mena pozdraví, opýta sa "Ako sa máš?" a primerane zareaguje.

```
program POZDRAV;
uses Crt;
var  MENO, ODPOVED:string;
begin
clrscr;
write('Ako sa volas ? ');
readln(MENO);
{ nasledujuce zmeni prve pismeno v mene na velke - pozri Help pomocou Ctrl+F1 }
MENO:=upcase(MENO[1])+copy(MENO,2,length(MENO));
writeln;
writeln('Ahoj, ',MENO,'!');
writeln;
write('Ako sa mas? ');
readln(ODPOVED);
writeln;
if (ODPOVED='dobre') or (ODPOVED='vyborne') or (ODPOVED='OK')
  then writeln('Som rad, ze sa mas ',ODPOVED,'!')
  else if (ODPOVED='zle') or (ODPOVED='nanic') or (ODPOVED='na...')
         then writeln('Mrzi ma, ze sa mas ',ODPOVED,'!')
         else writeln('"',ODPOVED,'" nepoznam!');
writeln;
gotoxy(25,12);
writeln('A h o j , musim koncit!');
delay(3500);     { delay(CISLO) zastavi beh programu na CISLO milisekund }
clrscr
end.
```

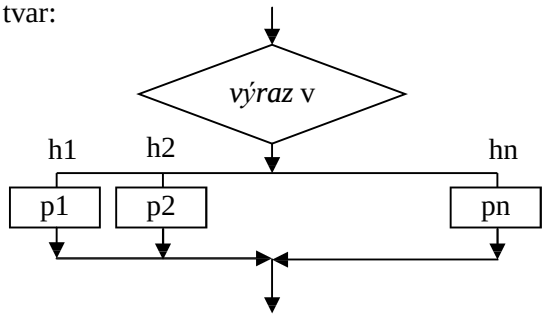
Príklad VIF-9: Už sme mali fyziku, chémiu, bankovníctvo, slušné správanie, tak znova trocha matematiky. Vytvorte program, ktorý zistí, či zadané číslo je deliteľné druhým zadaným číslom.

Analýza: Napríklad číslo 21 je deliteľné 7, lebo zvyšok po delení čísla 21 siedmimi je nula! Ale zvyšok po celočíselnom delení dáva funkcia mod, preto ak...

```
program DELI;
uses crt;
var DELENEC, DELITEL : integer;
begin
clrscr;
write('Zadaj delenca a delitela: ');
readln(DELENEC, DELITEL);
if DELENEC mod DELITEL=0
  then writeln('Cislo ',DELENEC,' je delitelne ',DELITEL)
  else writeln('Cislo ',DELENEC,' nie je delitelne ',DELITEL);
readln
end.
```

N-árne vetvenie, podmienený príkaz case

Má tvar:



h1, h2 až hn sú hodnoty, ktoré môže nadobudnúť výraz v

v slovnom zápise nemá ekvivalent,

realizuje sa zápisom:

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

kde p1 až pn sú príkazy

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

Má tvar:	**case v of**	kde v je tzv. výberový výraz ordinálneho typu,
	h1 : p1;	h1, h2 až hn sú hodnoty rovnakého typu ako výberový výraz
	h2 : p2;	a p, p1, p2 až pn sú príkazy
	...	
	hn : pn	
	[else p]	do hranatých zátvoriek sa umiestňuje nepovinná časť
	end	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.

Príklad:
```
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
end          { do premennej DNI priradí počet dní v mesiaci }
```

Dovolené sú aj zápisy s intervalmi (medzi minimálnou a maximálnou hodnotou sú dve bodky):

```
case ZNAK of
  'A'..'V' , 'a'..'v' : write( succ(ZNAK) );
             'Z' : write( 'A' ) ;
              'z' : write( 'a' )
   else write(ZNAK)
end      { príkaz zakóduje písmeno na nasledujúce  (Z, z na A, a) a zobrazí ho }
         { iné znaky nemení, zobrazí pôvodné }
```

Príklad VCS-1: Program na určenie aktuálneho dňa v týždni pomocou príkazu GetDate (unit Dos), na určenie počtu dní do konca mesiaca a do konca roka.

Analýza: Výsledkom príkazu GetDate (pozri pomocou Ctrl+F1) je aktuálny rok, mesiac, deň a poradové číslo dňa v týždni, pričom poradové číslo nedele je 0, pondelka 1 atď. Využili sme aj tvrdenie, že rok je priestupný (február má 29 dní), ak je deliteľný 4 a pritom nie je deliteľný 100 alebo je deliteľný 400. Rok 2000 by mal byť priestupný, lebo je deliteľný 4, nemal by byť priestupný, lebo je deliteľný 100, ale je priestupný, pretože je deliteľný 400. Zaujímavé. Ak rok nie je deliteľný 4, určite je nepriestupný.

Dali sme si záležať aj na koncovke slova deň, dni, dní (slovenská diakritika sa do TP dá dostať).

Riešenie výpočtu dní do konca roka nie je veľmi elegantné, pokúste sa o krajšie riešenie .

```
program POCET_DNI;
uses crt,dos;
var
  d, m, r, por : word;
  pdm, dkm: byte;    { pdm - pocet dni v mesiaci, dkm - do konca mesiaca }
  dkr:integer;       { dkr - do konca roka }
  priestupny:boolean;{ vysledkom je true, ak je rok priestupny, inak false }
begin
clrscr;
GetDate(r,m,d,por);
write('Dnes je ');
case por of
  0: write('nedela');
  1: write('pondelok');
  2: write('utorok');
  3: write('streda');
  4: write('stvrtok');
  5: write('piatok');
  6: write('sobota');
end;
writeln(' ', d, '.', m, '.', r, '.');
writeln;
priestupny:=(r mod 4=0) and ((r mod 100<>0) or (r mod 400=0));
case m of
  1,3,5,7,8,10,12: pdm:=31;
                2: if priestupny
                     then pdm:=29
                     else pdm:=28;
         4,6,9,11: pdm:=30;
end;
dkm:=pdm-d;
write('Do konca mesiaca ');
case dkm of
      1: writeln('zostáva 1 deň.');
  2,3,4: writeln('zostávajú ', dkm,' dni.');
  else writeln('zostáva ', dkm,' dní.')
  end;
writeln;
case m of
  1: if priestupny
       then dkr:=366-d
       else dkr:=365-d;
  2: if priestupny
       then dkr:=366-31-d
       else dkr:=365-31-d;
  3: dkr:=dkm+30+31+30+31+31+30+31+30+31;
  4: dkr:=dkm+31+30+31+31+30+31+30+31;
  5: dkr:=dkm+30+31+31+30+31+30+31;
  6: dkr:=dkm+31+31+30+31+30+31;
  7: dkr:=dkm+31+30+31+30+31;
```

```
  8: dkr:=dkm+30+31+30+31;
  9: dkr:=dkm+31+30+31;
 10: dkr:=dkm+30+31;
 11: dkr:=dkm+31;
 12: dkr:=dkm;
end;
write('Do konca roka ');
case dkr of
      1: writeln('zostáva 1 deň.');
  2,3,4: writeln('zostávajú ', dkr,' dni.');
  else writeln('zostáva ', dkr,' dní.')
end;
readln
end.
```

Neriešené úlohy na vetvenie:

1. Vytvorte program na riešenie rovnice ax + b = 0, a, b reálne.
2. Vytvorte program na riešenie kvadratickej rovnice $ax^2 + bx + c = 0$, a, b, c reálne, $a \neq 0$.

   Návod: $D = b^2 - 4.a.c$; pre $D \geq 0$ platí $x_{1,2} = (-b \pm \sqrt{D}) / (2.a)$.
3. Vytvorte program na riešenie rovnice $ax^2 + bx + c = 0$, a, b, c reálne.

   Návod: Pozri úlohy č. 1 a 2.
4. Vytvorte program na výpočet neznámej strany pravouhlého trojuholníka po zadaní usporiadanej trojice [a, b, c] (a, b odvesny, c prepona), kde neznáma strana má pri zadaní veľkosť 0.

   Návod: Použi Pythagorovu vetu, napríklad pre trojicu [3,0,5] sa počíta $b = (5^2 - 3^2) = 4$.
5. Vytvorte program na výpočet výšky poštovného pri podaní peňažnej poukážky (tarifná tabuľka je na druhej strane poukážky).
6. Vytvorte program na výpočet povrchu a obvodu zvoleného rovinného útvaru. Ponuka napr.: štvorec, obdĺžnik, trojuholník, lichobežník, kruh a pod. (selekcia príkazom case!).
7. Vytvorte program, ktorý zistí, či zadané tri čísla môžu byť veľkosti strán trojuholníka.

   Návod: Použi trojuholníkovú nerovnosť: súčet dvoch strán v trojuholníku je vždy väčší ako tretia strana.
8. Ukážte, že tri podmienky umožňujú rozlíšiť 8 situácií.
9. Osoby majú hmotnosť x , y a z. Nosnosť výťahu je w. Vytvorte program na zistenie, koľko jázd je treba, aby sa všetci dostali výťahom z prízemia na 13. poschodie.
10. Je daný počet dní v mesiaci a informácia, na ktorý deň v týždni pripadá prvý deň v mesiaci (pondelok - 1, utorok - 2, atď.). Zistite, koľko je v danom mesiaci piatkov.
11. Je daný počet dní v mesiaci a informácia, na ktorý deň v týždni pripadá prvý deň v mesiaci (pondelok - 1, utorok - 2, atď.). Zistite, koľko je v danom mesiaci pracovných dní.
12. Vytvorte program, ktorý po zadaní postupnosti celých čísel oznámi, či je postupnosť rastúca, neklesajúca, klesajúca, nerastúca, konštantná alebo nemá žiadnu z týchto vlastností.

CYKLUS

Použijeme, ak nejaký príkaz alebo skupina príkazov sa má opakovane vykonávať, pokiaľ je splnená alebo nesplnená daná podmienka.

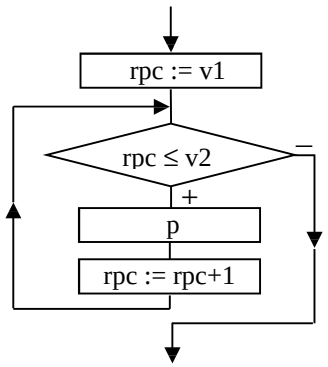
Cykly rozdeľujeme:

1. podľa umiestnenia podmienky na:
    a) cyklus s podmienkou na začiatku
    b) cyklus s podmienkou na konci
    c) úplný cyklus (s podmienkou v strede)
2. podľa toho, či je známy alebo neznámy počet opakovaní v cykle na:
    a) cyklus s explicitne daným počtom opakovaní
    b) cyklus s implicitne daným počtom opakovaní.

Cyklus s explicitne daným počtom opakovaní, príkaz for

Cyklus s explicitne („zvonka“) daným 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

V TP cyklu so známym 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 succ(rpc) (pred(rpc)).

Poznámky:

- Riadiaca premenná cyklu je ordinálneho typu, t.j. môže byť len typu integer, boolean alebo char. Z toho vyplýva, že nemôže nadobúdať hodnoty meniace sa o ľubovoľný krok, napr. 0,5; 0,1 a pod. Programátor s dobrým programátorským štýlom zásadne nemení hodnotu rpc ani o krok napr. 2, 10 a pod., len na nasledovníka resp. predchodcu. Ak potrebujeme zrealizovať cyklus s „nevhodným“ krokom pre príkaz for, použijeme cyklus s príkazom while alebo repeat.

- Začiatočná a koncová hodnota riadiacej premennej cyklu sa získa vyhodnotením výrazov v1 a v2 na začiatku vykonávania príkazu for a počas cyklu ich nemožno meniť.
- Z historických dôvodov (z čias programovacieho jazyka fortran) sa zaužívalo najčastejšie označiť riadiacu premennú cyklu písmenom I, nie je to však pravidlo.

Príklady: for I := 1 to 100 do write(I : 4)

príkaz zobrazí čísla 1 2 3 4 ... 99 100 v piatich riadkoch po 20 čísel (80 znakov v riadku delené 4 znaky na číslo = 20 čísel v riadku)

for I := 100 downto 1 do write (I : 4)

príkaz zobrazí čísla 100 99 98 ... 2 1, t.j. opačne ako v predchádzajúcom príkaze for

SUCET := 0;

for CISLO := 1 to POCET do SUCET := SUCET + CISLO

príkaz sčíta čísla 0 + 1 + 2 + 3 + ... + POCET, 0 je počiatočná hodnota premennej SUCET

MOCNINA := 1;

for I := 1 to N do MOCNINA := MOCNINA * X

príkaz vykoná 1.X.X. ... .X = X^N, 1 je počiatočná hodnota premennej MOCNINA

for Z := 'A' to 'Z' do writeln(Z : 5, ord(Z) : 3)

príkaz zobrazí v riadkoch znaky A až Z a ich poradové čísla z ASCII tabuľky

for J := -10 to 10 do p

príkaz for vykoná príkaz p 21-krát (J bude postupne -10, -9, ... , 0, ... , 10)

for K := 0 to (downto) 0 do p

príkaz for vykoná príkaz p jedenkrát

for NIC := 100 to –100 do p

príkaz for nevykoná príkaz p ani raz (začiatočná hodnota rpc nie je ani raz menšia alebo rovná koncovej hodnote)

for NIC := 0 downto 10 do p

príkaz for nevykoná príkaz p ani raz (začiatočná hodnota rpc nie je ani raz väčšia alebo rovná koncovej hodnote)

for NIC := 'z' to 'A' do p

príkaz for nevykoná príkaz p ani raz, pretože 'z' > 'A' (znak z má v ASCII tabuľke väčšie poradové číslo)

for I := - trunc(sqrt(N)) to trunc(sqrt(N)) do p

Výrazy v1 a v2 nemusia mať pri zápise len konkrétne hodnoty, môžu to byť aj výrazy.

Príklad CFR-1: Vytvorte program na zobrazenie hodnôt riadiacej premennej cyklu zo zadaného intervalu pre typ integer aj char. Uvedomte si, že ak variant „to“ zobrazí viac ako jednu hodnotu, variant „downto“ nemôže zobraziť ani jednu hodnotu a naopak.

```
program HODNOTY_RPC;
uses crt;
var  ZAC, KON, I : integer;
     ZACZ, KONZ, Z : char;
begin
```

```
clrscr;
write('Zadaj zaciatocnu a koncovu hodnotu typu integer: ');
readln(ZAC,KON);
writeln('Hodnoty rpc pre "to":');
for I:=ZAC to KON do
  write(I:8);
writeln;
writeln('Hodnoty rpc pre "downto":');
for I:=ZAC downto KON do
  write(I:8);
writeln; writeln;
write('Zadaj zaciatocnu hodnotu typu char: ');
readln(ZACZ);
write('Zadaj  koncovu  hodnotu  typu char: ');
readln(KONZ);
writeln('Hodnoty rpc pre "to":');
for Z:=ZACZ to KONZ do
  write(Z:4);
writeln;
writeln('Hodnoty rpc pre "downto":');
for Z:=ZACZ downto KONZ do
  write(Z:4);
writeln;
readln
end.
```

Neuviedli sme analýzu, radšej upozorníme na „problém“ s načítaním viacerých hodnôt typu char v jednom príkaze readln. Keby sme, tak ako pri type integer, umiestnili premenné ZACZ a KONZ do jedného príkazu readln a hodnoty vložili oddelené medzerou, premennej KONZ by vždy priradilo medzeru (druhý znak je predsa medzera)! Preto nečíselné hodnoty načítavame z klávesnice väčšinou v samostatných príkazoch readln. Ak chceme načítať obe hodnoty v jednom príkaze readln, musíme ich vložiť za sebou bez oddelenia.

Príklad CFR-2: Vytvorte program na výpočet súčtu všetkých prirodzených čísel od 1 po zadané N pomocou cyklu.

Analýza:

1. Akú algoritmickú konštrukciu treba použiť?

   Napríklad pre N = 100 treba sčítať 1 + 2 + 3 + ... + 99 + 100 = 5050. Vidíme, že sa opakuje sčítavanie. Keďže opakovane treba vykonávať nejaký príkaz, použijeme cyklus. Poznáme aj počet opakovaní, N-krát (aj 1 treba pričítať k pôvodnej hodnote v pamäti), preto to je cyklus so známym počtom opakovaní a ten sa realizuje príkazom for: for CISLO := 1 to N do

2. Aký príkaz sa má v cykle opakovať?

   Vyzerá to tak, že vždy k čiastočnému súčtu, t.j. k súčtu, kým nie je všetko sčítané, treba pripočítať vhodné číslo. Vhodné číslo je pri prvom prechode cyklom 1, pri druhom 2, pri treťom 3 atď., až pri n-tom N. Ak sme pozorní, uvedomíme si, že práve tieto hodnoty nadobúda riadiaca premenná cyklu for. Preto: nový čiastočný súčet = predchádzajúci čiastočný súčet + CISLO

   Ak dobre poznáme vykonanie príkazu priradenia, uvedomíme si, že ten presne pracuje s „novým a predchádzajúcim“, ak na jeho ľavej aj pravej strane je tá istá premenná (to isté pamäťové miesto)! Preto v cykle treba opakovať príkaz: SUCET := SUCET + CISLO

   V príkaze takéhoto typu musíte neustále vidieť: nový SUCET = starý SUCET + ...

   alebo všeobecne: nová HODNOTA = stará HODNOTA +,*,-,/...

3. Teraz je už všetko v poriadku?

   Ešte nie. Ak začneme simulovať výpočet v cykle, zistíme, že nepoznáme hodnotu premennej SUCET na pravej strane pri prvom prechode cyklom. Nepoznáme počiatočný súčet resp. nenastavili sme ešte počiatočnú hodnotu premennej SUCET. Správne vykonanie príkazu priradenia, v ktorom je rovnaká premenná na ľavej aj pravej strane, si vyžaduje pred príkazom cyklu nastavenie počiatočnej hodnoty takejto premennej. Pre sčítanie to býva (ale vo všeobecnosti nemusí byť) 0, pre násobenie 1. Preto pred príkaz for vložíme ešte príkaz: SUCET := 0;

   Už počujeme hlasy, že váš program počíta správne aj bez tohto priradenia (spomínali sme, že TP 7.0 vloží do všetkých číselných premenných počiatočnú hodnotu 0). Čo však, ak program nespustíte v TP 7.0 alebo ho vložíte do cyklu, aby vám viackrát umožnil zopakovať výpočet? Potom určite nebude počítať správne (počiatočnou hodnotou bude náhodné číslo alebo hodnota predchádzajúceho súčtu).

   Teraz je už všetko v poriadku a tu je program:

```
program SucetPrvychN;
uses crt;
var  N, CISLO, SUCET : integer; { pripadne typ word alebo longint }
begin
clrscr;
write('Scitat prirodzene cisla po ');
readln(N);
SUCET:=0;
for CISLO:=1 to N do
  SUCET:=SUCET+CISLO;
writeln('Sucet prvych ',N,' cisel je ',SUCET);
readln
end.
```

Poznámka: Na výpočet súčtu prvých n prirodzených čísel možno použiť aj vzorec SUCET = $\frac{N}{2}$.(1+N), preto sme v zadaní úlohy uviedli, že chceme výpočet pomocou cyklu. Zároveň si uvedomte, že výrazne sa efektivitou výpočtu líšiacich algoritmov môže byť (a väčšinou aj býva) viacej!

Program upravte tak, aby:

a) počítal súčet prvých N prirodzených čísel od N po 1,
b) počítal súčet všetkých prirodzených čísel od zadaného A po zadané B vrátane,
c) počítal súčet všetkých celých čísel od zadaného A po zadané B vrátane.

Príklad CFR-3: Vytvorte program na výpočet mocniny x na n-tú, x reálne, n prirodzene číslo.

Analýza: Začneme „sedliackym“ rozumom. Ak máme vypočítať napríklad 5^3, násobíme 5.5.5 = 125. Ďalej by sme mohli vedieť, že 0^0 nie je definované (asi preto, že 0 na „čokoľvek“ je 0 a „čokoľvek“ na 0-tú je 1, preto 0^0 by malo byť 0 a zároveň 1 a to nejde).

1. Opakovane sa vykonáva násobenie, preto treba použiť cyklus; násobenie treba vykonať n-krát, preto cyklus so známym počtom opakovaní, teda príkaz for.
2. Vždy nový súčin sa rovná predchádzajúci súčin krát x, preto sa v cykle bude opakovať príkaz SUCIN := SUCIN * X.
3. Nesmieme zabudnúť na počiatočnú hodnotu premennej SUCIN, ktorá môže byť 1. Keby sme zvolili nulu?

4. Je tu však ešte jedno obmedzenie, ak zadáme N aj X nula, tak má program oznámiť: „0 na 0-tú nedefinovane!“, inak má „normálne“ počítať – teda vetvenie.

```
program MOCNINA;
uses crt;
var X, MOCNINA : real;
     N, I, PDM : integer;
begin
clrscr;
write('Zadaj zaklad mocniny a exponent: ');
readln(X,N);
write('Zadaj pocet desatinnych miest vystupu: ');
readln(PDM);
if (X=0) and (N=0)
  then writeln('0 ma 0-tu nedefinovane!')
  else begin
       MOCNINA:=1;
       for I:=1 to N do
         MOCNINA:=MOCNINA*X;
       writeln(X:PDM+2:PDM,' na ',N,' = ',MOCNINA:PDM+2:PDM)
       end;
readln
end.
```

Príklad CFR-4: Vytvorte program na výpočet aritmetického priemeru n reálnych čísel.

Analýza: Snáď všetci vieme, že aritmetický priemer sa vypočíta ako súčet daných n čísel delený počtom všetkých čísel, t.j. n. Program si má vypýtať n reálnych čísel a sčítať ich, preto cyklus s „príkazmi“: ***zadaj číslo*** a ***k čiastočnému súčtu pripočítaj nové číslo***. Toto sa má zopakovať pre n čísel, teda n-krát. Takže cyklus so známym počtom opakovaní – príkaz for.

V programe sme namiesto premennej n použili premennú POCET a ošetrili sme aj prípad, keď bude zadaný počet čísel 0 - program by bez ošetrenia skončil chybovým hlásením: Delenie nulou!

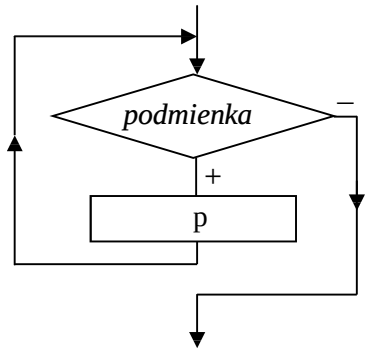
```
program PRIEMER;
uses crt;
var  I, POCET : integer;
     X, SUCET : real;
begin
clrscr;
write('Zadaj pocet cisel: ');
readln(POCET);
SUCET:=0;
for I:=1 to POCET do
  begin
  write(I:2,'. cislo: ');
  readln(X);
  SUCET:=SUCET+X
  end;
if POCET>0
  then writeln('Priemer zaokruhleny na stotiny: ',SUCET/POCET:4:2);
readln
end.
```

Ďalšie úlohy aj na príkaz for sú uvedené v kapitole Ďalšie zaujímavé úlohy.

Cyklus s podmienkou na začiatku, príkaz while

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

Má tvar:



v slovnom zápise:

pokiaľ *podmienka* opakuj
p

kde p je príkaz

V TP 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.

Príklady:

```
I := 1;
while I <= 10 do
   I := I + 1;                    príkaz v cykle sa vykoná 10-krát

SUCET := 0;
while N > 0 do
   begin
   SUCET := SUCET + N;
   N := N - 1
   end                            sčíta všetky prirodzené čísla od N po 1

DELITEL := 2;
while CISLO mod DELITEL <> 0  do
    inc ( DELITEL )               nájde deliteľa zadaného čísla ( CISLO > 1 )
```

Príklad CWH-1: Príkaz for je viac-menej špeciálnym prípadom príkazu while. Demonštrujeme to v nasledujúcom programe.

```
program AKO_FOR;
uses crt;
var  N, I : integer;
begin
clrscr;
write('Zadaj pocet opakovani: ');
readln(N);
writeln('Ako prikaz for s "to" od 1 po ',N);
I:=1;
while I<=N do
  begin
  writeln(I:6,'. opakovanie');
  I:=I+1
  end;
readln;
```

```
writeln('Ako prikaz for s "downto" od ',N,' po 1');
while N>0 do
  begin
  writeln(N:6,'. opakovanie');
  N:=N-1
  end;
readln
end.
```

Príklad CWH-2: Vytvorte program na výpočet ciferného súčtu zadaného prirodzeného čísla.

Analýza: Ciferný súčet napríklad čísla 123 je 6, lebo 1 + 2 + 3 = 6. K prvej cifre (3) sa dostaneme predelením čísla 123 desiatimi: 123 : 10 = 12, zvyšok 3. Pokúsme sa vystačiť s celými číslami. Ak spravíme 123 mod 10, dostaneme rovno číslo 3. Ak spravíme 123 div 10 dostaneme 12, čo sa nám tiež hodí, lebo ďalej potrebujeme to isté (mod a div) zopakovať s 12-kou (12 mod 10 = 2 a 12 div 10 = 1) a nakoniec s 1-kou (1 mod 10 = 1 a 1 div 10 = 0). Tým sme prešli všetky cifry čísla. Opakovane spracúvam výsledky po funkciách mod a div s deliteľom 10, preto treba použiť cyklus. Zadané číslo môže byť jedno ale aj troj (to naše) prípadne desaťciferné a na začiatku cyklu nevieme, kedy spracujeme poslednú cifru. Preto cyklus s neznámym počtom opakovaní príkazov v cykle – príkaz while. Podmienkou ukončenia cyklu je, aby sme nemali už čo „spracovať“, t.j. aby aktuálnym číslom bola nula (CISLO div 10 = 0).

```
program CIFERNY_SUCET;
uses crt;
var  CISLO    : longint;
     CIFSUCET : byte;
begin
clrscr;
write('Zadaj prirodzene cislo: ');
readln(CISLO);
write('Sucet cifier cisla ',CISLO,' je ');
CIFSUCET:=0;
while CISLO>0 do
  begin
  CIFSUCET:=CIFSUCET+CISLO mod 10;
  CISLO:=CISLO div 10
  end;
writeln(CIFSUCET);
readln
end.
```

Prečo sme výstup museli posunúť hore? Ak sa vám funkcie div a mod nepáčia, tu je riešenie bez nich:

```
program CIFERNY_SUCET_2;
uses crt;
var  CISLO    : longint;
     CIFSUCET : byte;
begin
clrscr;
write('Zadaj prirodzene cislo: ');
readln(CISLO);
write('Sucet cifier cisla ',CISLO,' je ');
CIFSUCET:=0;
while CISLO>0 do
  begin
  CIFSUCET:=CIFSUCET+trunc(frac(CISLO/10)*10);
  CISLO:=trunc(CISLO/10)
  end;
writeln(CIFSUCET);
readln
end.
```

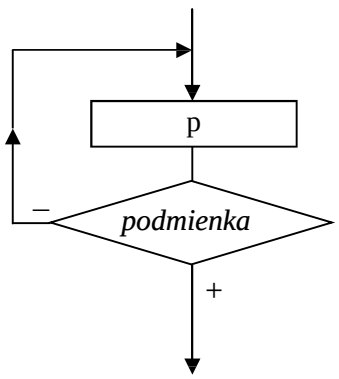
Príklad CWH-3: Vytvorte program na zobrazenie prevodnej tabuľky medzi slov. korunami a inou menou. Kurz je zadaný v konštante. Tabuľka bude zobrazovať hodnoty od 0,- Sk po zadanú sumu (premenná KoncovaSuma) so zadaným prírastkom (premenná Krok). Napríklad pre koncovú sumu 100 s prírastkom 10 má vypočítať a zobraziť hodnoty pre 0, 10, 20, 30, 40, ... , 100 Sk. Výpočet hodnoty v cudzej mene je jednoduchý: suma v Sk krát kurz. Keďže sa opakovane počíta hodnota v cudzej mene, treba použiť cyklus. Riadiaca premenná cyklu sa však môže zväčšovať o teoreticky ľubovoľné kladné reálne číslo, preto použijeme príkaz while.

Program sme doplnili o jednoduchú hlavičku tabuľky a hodnoty sú zobrazované s presnosťou na desatiny, na miestach stotín sa zobrazujú nuly. Doplňte tabuľku o zvislé „čiary".

```
program TABULKA;
uses  crt;
const KURZ=1.24;
var   KoncovaSuma, Krok, RPC : real;
begin
clrscr;
write('Zadaj koncovu sumu a prirastok: ');
readln(KoncovaSuma, Krok);
clrscr;
writeln('----------------------':48);
writeln('      SK         MENA ':48);
writeln('----------------------':48);
RPC:=0;
while RPC<=KoncovaSuma do
  begin
  writeln(RPC:33:1,'0',KURZ*RPC:12:1,'0');
  RPC:=RPC+KROK
  end;
writeln('----------------------':48);
readln
end.
```

Cyklus s podmienkou 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 TP 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.

Príklady:

```
I := 0;
repeat
  I := I + 1
until I = 10
```
príkaz v cykle sa vykoná 10-krát

```
repeat
  write('Zadaj číslo: ');
  readln(X)
until X = 0
```
príkazy v cykle sa budú opakovať, kým nevložíme 0

```
repeat
  p1; ...
  pn
until false
```
nekonečný cyklus ukončí sa Ctrl+Break

Príklad CRP-1: Vytvorte programovú schému na zabezpečenie opakovania behu programu ľubovoľný počet krát.

Analýza: Už vás nudí neustále spúšťať ten istý program, najmä keď ho testujete, či pracuje správne? Riešením je príkaz repeat. Chceme dosiahnuť, aby sa po spustení program vykonal a na konci sa opýtal: „Opakovať?“. Ak odpovieme N = nie, program skončí, inak sa vykoná znova.

Máme viacej možností, najprv tá najjednoduchšia:

```
...
begin
repeat
  { príkazy programu }
  write('Opakovať (nie = N) ?');
  readln(ODPOVED)
until (ODPOVED = 'N') or (ODPOVED = 'n')
end.
```

Elegantnejším riešením je použiť funkciu unitu crt readkey. Výsledkom funkcie readkey je stlačený kláves, ušetríme premennú ODPOVED aj stlačenie klávesu Enter. Readkey v kombinácii s upcase:

```
begin
repeat
  { príkazy programu }
  writeln('Opakovať (nie = N) ?')
until  upcase(readkey) = 'N'
end.
```

Na ukončenie programov sa používa aj kláves Esc. Pomocou programu na nasledujúcej strane zistíme, že jeho poradové číslo je 27. A tu je schéma:

```
begin
repeat
  { príkazy programu }
  writeln('Koniec – stlač Esc!':45)
until  readkey = chr(27)
end.
```

```
const Esc = #27;
...
begin
{ príkazy programu }
writeln('Koniec – stlač Esc!':45)
until  readkey = Esc
end.
```

Zápis chr(X) je rovnocenný so zápisom #X.

Aj na zistenie poradového čísla stlačeného znaku môže poslúžiť malý program s cyklom repeat:

```
program KOD_STLACENEHO_KLAVESU;
uses crt;
var  stlaceny_klaves:char;
begin
clrscr;
writeln('Koniec – stlac C':45);
repeat
  stlaceny_klaves:=readkey;
  writeln(ord(stlaceny_klaves));
until upcase(stlaceny_klaves)='C';
end.
```

Príklad CRP-2: Vytvorte programovú schému na ošetrenie správnosti vstupných hodnôt. Ak vstupné hodnoty nespĺňajú vstupné podmienky, majú sa vyžiadať nové vstupné hodnoty.

Analýza: Príkaz repeat je vhodný aj na ošetrenie vstupných hodnôt, či spĺňajú vstupné podmienky. Najprv predsa musíme zadať vstupné hodnoty a až potom môže podmienka rozhodnúť, či treba vstup opakovať alebo môže program pokračovať. Ako ukážku ošetríme výpočet mocniny x^n pre x = 0 a n = 0 (pozri príklad CFR-3):

```
repeat
  write('Zadaj zaklad mocniny a exponent: ');
  readln(X,N)
until (X<>0) or (N<>0)
```

Správnejšie je nie len „zacykliť“ vstup ale aj doplniť výpisom, prečo sa príkazy vstupu opakujú. Teda:

```
repeat
  write('Zadaj zaklad mocniny a exponent: ');
  readln(X,N);
  if (X=0) and (N=0) then writeln('0 na 0-tú nedefinovanú!')
until (X<>0) or (N<>0);
```

Pod chybou vstupu môžeme rozumieť aj „preklep“, napríklad vložením písmena namiesto číslice (Error 106: Invalid numeric format.). K vyriešeniu tohto problému treba poznať funkciu IOResult, ktorá po každej vstupno/výstupnej operácii nadobúda buď hodnotu 0, ak nedošlo ku chybe, alebo číslo chyby. Keďže sme zapli v Options – Compiler... prepínač I/O Checking – kontrolu všetkých vstupno/výstupných operácií, program by hneď skončil s chybovým hlásením. Túto kontrolu však možno vypnúť umiestnením direktívy {$I - } pred príkaz vstupu a následne ju zapnúť príkazom {$I +}. Schéma má tvar:

```
...
var CHYBA : boolean;
            X : integer;   { X akýkoľvek číselný typ }
...
begin
repeat
  write('Vlož číslo: ');
  {$I-} readln(X); {$I+}
  CHYBA := IOResult <> 0;
  if CHYBA then writeln('Chyba vstupu!')
until not CHYBA;
napr. writeln(X)
```

Ak dôjde ku chybe, IOResult bude rôzny od nuly a preto premenná CHYBA nadobudne hodnotu TRUE. Na obrazovke sa zobrazí: Chyba vstupu! a znova Vlož číslo:, pretože CHYBA má hodnotu true a negácia (not) true je false – vtedy sa príkazy v repeat opakujú. Otestujte program pre dobré aj zlé vstupné hodnoty

A teraz už „normálny“ príklad:

Príklad CRP-3: Vytvorte program – hru na uhádnutie zadaného čísla, keď po vložení čísla program oznámi „Veľa, uber!“, „Málo, pridaj!“ alebo „Výborne, uhádol si!“. Nech treba uhádnuť celé číslo z intervalu od 0 po 100.

Analýza: Pre získanie čísla, ktoré bude treba uhádnuť sú dve možnosti:

- buď máme protihráča a ten zadá číslo tak, aby to súper nevidel (napríklad čiernym písmom – nastavuje sa príkazom textcolor na čiernom podklade – nastavuje sa príkazom textbackground), alebo
- číslo pomocou funkcie random vyberie počítač (random(x) - výsledkom je náhodné celé číslo z intervalu <0,x); pred príkazom random musí byť použitý na začiatku programu príkaz randomize).

Druhá časť programu je vlastné hádanie. „Sedliacky“ rozum hovorí: vložíme číslo, počítač ho porovná s číslom, ktoré treba uhádnuť a oznámi „Veľa, uber!“ alebo „Málo, pridaj!“ a znova sa celé opakuje, až kým neuhádneme, vtedy hádanie končí. Keďže nevieme, na koľký pokus sa nám podarí uhádnuť hľadané číslo, je to cyklus s neznámym počtom opakovaní. Najprv musíme vložiť číslo a porovnať a až potom vieme rozhodnúť, či cyklus končí – preto cyklus s podmienkou na konci. A tu je hotový program:

```
program HADAJ;
uses  crt;
const MAX=100;
var   CISLO,UHCISLO:integer;
      ODP:char;
begin
randomize;
repeat
  clrscr;
  gotoxy(1,12); writeln('Cislo vyberie pocitac ....... P':50);
  writeln;      writeln('Cislo zada protihrac ........ H':50);
  ODP:=upcase(readkey);
  clrscr;
  case ODP of
    'P': UHCISLO:=random(MAX+1);
    'H': begin
         gotoxy(15,12);
         write('Zadaj cislo na uhadnutie < ',MAX,': ');
         textcolor(black);
         readln(UHCISLO);
         textcolor(white);
         clrscr;
         end;
  end;
  writeln('H A D A J  C I S L O !');
  repeat
    readln(CISLO);
    if CISLO <> UHCISLO
      then if CISLO<UHCISLO then writeln('Malo, pridaj!')
                            else writeln('Vela, uber!');
  until CISLO=UHCISLO;
  clrscr; textcolor(yellow+blink);
  gotoxy(30,12); write('B  I  N  G  O  ! ! ! ');
  delay(2500);
  normvideo;
  gotoxy(30,24); write('Este raz? (nie = N) ');
until upcase(readkey)='N';
end.
```

Príkaz normvideo by mal nastaviť štandardné atribúty pre farby textu aj pozadia. Program môžete doplniť aj o zvukové efekty pomocou príkazov sound(*frekvencia*) a nosound – vypína zvuk, tento nesmiete zabudnúť uviesť po príkaze sound. Dĺžka trvania tónu sa realizuje príkazom delay(*milisekúnd*).

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 ASCII), 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 situácii, 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.

Sem-tam sa u začiatočníkov stáva, v snahe nezabudnúť bodkočiarku, že ju dajú aj za slovo „do“ v príkaze for alebo while. Tým vznikol tzv. **prázdny príkaz** a cyklus zrejme pracuje celkom ináč, ako to bolo pôvodne plánované.

Organizácia programu

Poslednou teoretickou časťou je organizácia programu v Turbo Pascale.

Program sa skladá:

program *meno programu;*	hlavička programu (môže chýbať)		
uses *unity oddelené čiarkami***;**	deklarácia unitov		
const *meno konštanty* **=** *hodnota***;**	definícia konštánt		
type *meno typu* = ...;	definícia typov	úsek definícií	
var *premenné oddelené čiarkami* **:** *typ premenných*;	deklarácia premenných	a deklarácií	
procedure ...	deklarácia		
function ...	procedúr a funkcií		blok
begin			
príkaz1;	príkazová časť		
príkaz2;	začína slovom begin		
...	a končí slovom end		
príkazn			
end.	bodka – koniec programu (nepatrí do bloku)		

Poznámky:

- Niektoré objekty z predchádzajúcej schémy ešte nepoznáme.
- Príkazy sa v príkazovej časti oddeľujú bodkočiarkou.
- Všetky mená objektov sú identifikátory.

EXE verzie programov

Turbo Pascal umožňuje vytvárať aj od prostredia TP nezávislé verzie programov, priamo programy s príponou exe. Keď chceme získať exe verziu programu, musíme jeho kompiláciu (preklad) presmerovať z pamäte na disk. Robí sa to prepnutím položky Destination v Compile z hodnoty Memory na hodnotu Disk. Program s rovnakým názvom ako bol pôvodný súbor, len s príponou exe, sa uloží do aktuálneho adresára.

Ďalšie riešené úlohy

Príklad CWH-4: Vytvorte program na krátenie zlomkov.

Analýza: Napr. v zlomku 15/12 sa dá čitateľ aj menovateľ deliť číslom 3 a zlomok sa dá upraviť do základného tvaru 5/4. Číslo 3, ktorým sme krátili zlomok, je najväčší spoločný deliteľ (nsd) čitateľa a menovateľa zlomku. Nsd(15,12) sa vypočíta takto: nsd(15,12) = nsd(15-12,12) = nsd(3,12) = nsd(3,12-3) = nsd(3,9) = nsd(3,9-6) = nsd(3,3) = 3. Opakujeme odčítanie menšieho čísla od väčšieho dovtedy, kým sa nerovnajú. Rovnaké číslo je najväčší spoločný deliteľ pôvodných čísel.

V slovnom zápise:
```
pokiaľ je A<>B opakuj
    ak A>B
        tak A := A-B
        inak B := B-A;
NSD := A  { alebo B, keďže sú rovnaké }
```

Výstup programu nech je v tvare: 15 / 12 = 5 / 4, ak je podiel celé číslo, zlomková čiara a menovateľ 1 nech sa nezobrazia, napr. 12 / 6 = 2.

```
program ZLOMOK;
uses crt;
var CIT,MEN,A,B:integer;
begin
clrscr;
repeat
  write('Zadaj citatela a menovatela zlomku: ' );
  readln(CIT,MEN);
  write(CIT,' / ',MEN,' = ');
  A:=CIT;
  B:=MEN;
  while A<>B do
    if A>B
      then A:=A-B
      else B:=B-A;
  CIT:=CIT div A;
  MEN:=MEN div B;
  if MEN=1
    then writeln(CIT)
    else writeln(CIT,' / ',MEN);
  writeln('Koniec - stlac Esc!')
until readkey=chr(27)
end.
```

Príklad CRP-4: Vytvorte program na výpočet aritmetického priemeru vopred neznámeho počtu kladných reálnych čísel.

Analýza: Úloha je podobná úlohe CFR-4, v nej však bol známy počet čísel a preto sme mohli použiť príkaz for. Pri neznámom počte čísel musíme použiť cyklus s neznámym počtom opakovaní. Ukončenie cyklu je v takomto prípade na tzv. koncovú hodnotu. V našom príklade sú všetky zadávané reálne čísla kladné a preto môžeme zvoliť za koncovú hodnotu nulu. Znamená to, že príkazy v cykle sa budú opakovať, pokiaľ nevložíme 0, vtedy sa cyklus ukončí. Najprv vložíme číslo a potom môžeme rozhodnúť, či cyklus končí, teda podmienka je na konci, preto príkaz repeat. Ošetrite aj prípad, keď hneď prvým vloženým číslom bude 0.

```
program PRIEMER2;
uses crt;
var     POCET : integer;
     X, SUCET : real;
begin
clrscr;
POCET:=0;
SUCET:=0;
repeat
  write('Vloz cislo: ');
  readln(X);
  if X>0
    then begin
         inc(POCET);
         SUCET:=SUCET+X
         end
until X=0;
if POCET>0
  then writeln('Priemer zaokruhleny na stotiny: ',SUCET/POCET:4:2)
  else writeln('Bola zadana len koncova hodnota!');
readln
end.
```

Program CFR-5: Program na prevod malých písmen textu na veľké.

Analýza: Prakticky si treba len osvojiť zápis napr. TEXT[1] – prvý znak reťazca v premennej TEXT, TEXT[5] – piaty znak reťazca, TEXT[I] – I-ty znak reťazca v premennej TEXT. Ak teda chceme zmeniť všetky písmená v reťazci na veľké, musíme ísť od prvého znaku až po posledný. Počet znakov v reťazci nám zistí funkcia length (vracia číslo – dĺžku reťazca). Funkcia upcase zmení malé písmeno na veľké, ostatné znaky nemení.

```
program PREVOD;
uses crt;
var TEXT : string;
    DLZKA, I : integer;
begin
clrscr;
write('Text: ');
readln(TEXT);
DLZKA:=length(TEXT);
for I:=1 to DLZKA do
  TEXT[I]:=upcase(TEXT[I]);
writeln(TEXT);
readln
end.
```

Program CFR-6: Vytvorte program na kódovanie textu posunutím o jeden znak vpravo (Zz -> Aa) a dekódovanie textu posunutím o jeden znak vľavo (Aa -> Zz).

Analýza: Podstatou algoritmu je brať znak po znaku text a pokiaľ je znakom písmeno, zobraziť jeho nasledovníka – pri kódovaní, resp. predchodcu – pri dekódovaní. Situáciu trochu komplikujú okrajové písmená, pretože veľké Z a malé z sa majú posunúť na veľké A a malé a resp. Aa na Zz pri dekódovaní. Keďže je viacej možností, a na každú sa má ináč kódovať, vhodné je použiť príkaz case s časťou else (každý iný znak ako písmeno sa nemá zmeniť). Prejsť treba celý text, od prvého po posledný znak, čo nám zaručí príkaz for (v2 = length(TEXT)). Program musí umožňovať vybrať si z dvoch možností: Kódovať alebo Dekódovať.

```
program KODOVANIE_a_DEKODOVANIE_posunutim_o_znak;
uses crt;
var TEXT:string;
    I:byte;
    ODPOVED:char;
begin
clrscr;
write('Text: ');
readln(TEXT);
writeln('[K]odovat alebo [D]ekodovat?');
ODPOVED:=upcase(readkey);
if ODPOVED='K'
  then begin
       for I:=1 to length(TEXT) do
         case TEXT[i] of
           'A'..'V','a'..'v': write(succ(TEXT[I]));
                        'Z': write('A');
                        'z': write('a');
           else write(TEXT[I])
         end;
       writeln
       end
   else if ODPOVED='D'
          then begin
               for I:=1 to length(TEXT) do
                 case TEXT[i] of
                   'B'..'Z','b'..'z': write(pred(TEXT[I]));
                                'A': write('Z');
                                'a': write('z');
                   else write(TEXT[I])
                 end;
               writeln
               end
          else writeln('Nebolo stlacene K alebo D!');
readln
end.
```

Príklad CFR-7*: Príklad je určený pre zdatnejších alebo húževnatejších programátorov. Je principiálne odlišný od predchádzajúceho. Program má kódovať a dekódovať písmená textu posunutím o POSUN písmen vpravo (vľavo).

Analýza: Programy s hviezdičkou (nie sú zakázané) sú určené na prácu s hotovým programom. Ujasnite si jednotlivé časti programu a vyskúšajte, ako pracujú. Program si pamätá predchádzajúci text (pri prvom spustení samozrejme nie), aby ste ho mohli po zakódovaní hneď dekódovať a opačne. Dekódovanie je v skutočnosti kódovanie o 26-POSUN znakov (neustále vpravo). Číslo 26 je počet písmen (A až Z alebo malé a až z), pričom poradové číslo A je 65 a malého a 97. Pri kódovaní sa pohybujeme v uzavretom kruhu.

```
program KODOVANIE_a_DEKODOVANIE_posunutim_o_POSUN_znakov;
uses crt;
var TEXT:string;
    POSUN:integer;
    DLZKA,i,ZacAbc,PozVAbc,RelPozKodu:byte;
    ODPOVED:char;
begin
clrscr;
repeat
  writeln('Novy text (ano - A)?');
  if upcase(readkey)='A'
    then begin
         writeln('Text:');
         readln(TEXT);
         DLZKA:=length(TEXT);
         end;
  repeat
    write('[K]odovat alebo [D]ekodovat? ');
    readln(ODPOVED);
    ODPOVED:=upcase(ODPOVED);
  until (ODPOVED='K') or (ODPOVED='D');
  write('Posun o znakov: ');
  readln(POSUN);
  POSUN:=POSUN mod 26;
  if POSUN<0 then POSUN:=POSUN+26;
  if ODPOVED='D' then POSUN:=26-POSUN;
  for i:=1 to DLZKA do
    if TEXT[i] in ['A'..'Z','a'..'z']
      then begin
           case TEXT[i] of
             'A'..'Z':ZacAbc:=65;
             'a'..'z':ZacAbc:=97;
           end;
           PozVAbc:=ord(TEXT[i])-ZacAbc;
           RelPozKodu:=(PozVAbc+POSUN) mod 26;
           TEXT[i]:=chr(ZacAbc+RelPozKodu)
           end;
  writeln(TEXT);
  writeln('KONIEC - stlac Esc':45)
until readkey=#27
end.
```

Ďalšou skupinou úloh sú „grafické“ úlohy v textovom režime obrazovky.

Príklad CFR-8: Program ROZSVECOVANIE_V_OKNACH náhodným generovaním polohy postupne vypĺňa plochu obrazovky „rozsvietenými“ obdĺžnikmi (znak s poradovým číslo 219) .

Analýza: Treba opakovane „rozsvecovať okná“, preto cyklus. My sme ho postavili na známom počte opakovaní a preto sa program opýta, koľko okien má rozsvietiť (to je počet opakovaní príkazov v cykle). No a potom už stačí len použiť funkciu random a nechať náhodne vygenerovať číslo od 1 do 80 – stĺpec a číslo od 1 do 24 – riadok, v ktorom sa „rozsvieti“. Požadovaný stĺpec a riadok sa nastaví príkazom gotoxy.

```
program ROZSVECOVANIE_V_OKNACH;
uses crt;
var  I, POCET : word;
begin
randomize;
clrscr;
write('Pocet rozsvietenych okien do 65 tis.: ');
readln(POCET);
for I:=1 to POCET do
  begin
```

```
  gotoxy(random(80)+1,random(24)+1);
  write(chr(219));
  end;
readln
end.
```

Predchádzajúca úloha je ešte zaujímavejšia, ak chceme, aby program generoval „rozsvecovanie“, až kým nestlačíme ľubovoľný kláves. Funkcia keypressed unitu crt má hodnotu true, ak bol stlačený kláves, inak má hodnotu false. Celý problém je vo vytvorení tentoraz cyklu s neznámym počtom opakovaní, ktorý skončí na stlačenie klávesu. Ak chceme „rozsvecovanie“ spomaliť, stačí vložiť do cyklu príkaz delay(*milisekúnd*).

```
program ROZSVECOVANIE_V_OKNACH2;
uses crt;
begin
randomize;
clrscr;
repeat
  gotoxy(random(80)+1,random(24)+1);
  write(chr(219));
until keypressed;
readln
end.
```

V ďalšom príklade sa pokúsime usmerniť pohyb.

Príklad CRP-5: Vytvorte program na pád „vajíčka“ (písmena O) z hora nadol v strede riadkov.

Analýza: Kto si poctivo ujasnil predchádzajúce dva programy, nemôže mať s nasledujúcimi dvoma problémy. Mali by sme použiť príkaz for RIA:=1 to 25 do ... ale chceme od vás, aby ste použili príkaz repeat. Program sme doplnili aj o zvukový efekt. Ďalšie zvukové efekty môžete doprogramovať pre let vajíčka. Ak sa chcete pohrať, môžete pridať aj „gravitáciu“, aby vajíčko pri svojom páde zrýchľovalo.

```
program pad_vajca;
uses crt;
var  ria:byte;
begin
clrscr;
ria:=1;
repeat
  gotoxy(40,ria); write('.');
  gotoxy(40,ria+1); write('O');
  delay(150); inc(ria)
until ria=25;
sound(60);
gotoxy(36,12); write('B U M !!!');
gotoxy(37,ria); write('---^---');
delay(200);
nosound;
readln
end.
```

V ďalšej verzii programu PAD_VAJCA sa ho pokúsime zachrániť pred rozbitím. Nech program vygeneruje náhodne veľké písmeno (veľké písmená začínajú „na“ 65 a končia na 90) a zobrazí ho v ľavom hornom rohu obrazovky. Ak stihneme do dopadnutia vajíčka „do 25 riadku“ nájsť na klávesnici a stlačiť hľadaný kláves, pád sa zastaví.

```
program zachran_vajce;
uses crt;
var  RIA:byte;
     PISMENO, STLACIL:char;
begin
randomize;
clrscr;
PISMENO:=chr(random(26)+65);
gotoxy(1,5); write('Najdi ',PISMENO);
RIA:=1;
repeat
  gotoxy(40,RIA); write('.');
  gotoxy(40,RIA+1); write('O');
  if keypressed then STLACIL:=upcase(readkey);
  delay(100); inc(RIA)
until (STLACIL=PISMENO) or (RIA=25);
if RIA<25
  then begin gotoxy(33,12); write('V Y B O R N E !') end
  else begin
       sound(60);
       gotoxy(36,12); write('B U M !!!');
       gotoxy(37,RIA); write('---^---');
       delay(200); nosound
       end;
readln
end.
```

Ak vás pohyb na obrazovke zaujal, môžeme skúšať ďalej.

Príklad CRP-6: Vytvorte program na pohyb znaku po obrazovke. Na ovládanie pohybu použite šípky na ovládanie pohybu kurzora. Testujte aj „vybehnutie“ znaku z obrazovky.

Analýza: Tento program uvedieme bez hlbšej analýzy. Necháme ho na podrobné štúdium a experimentovanie. Ku kódom šípok sme sa dostali pomocou programu uvedeného nad príkladom CRP-2 (0 si nevšímajte, sú tam preto, že sa nejedná o obyčajné klávesy). Všimnite si cyklus repeat until keypressed, ktorý zamedzuje neustálemu prepisovaniu znaku na mieste a jeho blikaniu (prakticky zacyklí program na danom mieste až do stlačenia ľubovoľného klávesu). Program možno násilne kedykoľvek ukončiť stlačením klávesu Esc (príkaz halt – je to len ukážka, nenavykajte si naň!).

Programy začínajú byť zložitejšie a preto sme sa rozhodli začať nenápadne používať podprogramy (procedúry). Podprogram rieši nejaký čiastkový problém a je veľmi podobný programu (namiesto slova program používa slovo procedure). V príkazovej časti hlavného programu stačí uviesť meno procedúry a automaticky sa vykoná všetko, čo v nej je naprogramované.

```
program pohyb_znaku_pri_pridrzani_sipky_bez_blikania;
uses  crt;

const Esc=chr(27);
      sipkaVPRAVO=chr(77);     {M}
      sipkaVLAVO =chr(75);     {K}
      sipkaHORE  =chr(72);     {H}
      sipkaDOLE  =chr(80);     {P}

var   stl,ria:byte;

procedure inicializacia;
  begin
  clrscr;
  stl:=40; ria:=12; gotoxy(stl,ria); write('*');
  end;
```

```
procedure pohyb;
  var stlznak:char;
  begin
  repeat
    repeat until keypressed;
    stlznak:=upcase(readkey);
    gotoxy(stl,ria); write(' ');
    case stlznak of
      sipkaVPRAVO : inc(stl);
      sipkaVLAVO  : dec(stl);
      sipkaHORE   : dec(ria);
      sipkaDOLE   : inc(ria);
      Esc         : halt;
      end;
    gotoxy(stl,ria); write('*');
  until (stl=1)or(stl=80)or(ria=1)or(ria=25);
  gotoxy(25,13); writeln('B U M ! ! !  O H R A D A ! ! !');
  end;
BEGIN
inicializacia;
pohyb;
readln
END.
```

Pohyb je zaujímavejší, ak sa znak pohybuje určeným smerom, až kým nezvolíme iný smer. Čiže, kým nezvolíme iný smer (nestlačíme inú šípku), znak sa má uberať „starým“ smerom.

```
program pohyb_znaku_bez_podrzania_sipky;
uses  crt;
const Esc=chr(27);
      sipkaVPRAVO=chr(77);     {M}
      sipkaVLAVO =chr(75);     {K}
      sipkaHORE  =chr(72);     {H}
      sipkaDOLE  =chr(80);     {P}
var   stl,ria:byte;

procedure inicializacia;
  begin
  clrscr;
  stl:=40; ria:=12; gotoxy(stl,ria); write('*');
  end;

procedure pohyb;
  var stlznak:char;
  begin
  repeat
    if keypressed then stlznak:=upcase(readkey);
    gotoxy(stl,ria); write(' ');
    case stlznak of
      sipkaVPRAVO : inc(stl);
      sipkaVLAVO  : dec(stl);
      sipkaHORE   : dec(ria);
      sipkaDOLE   : inc(ria);
      Esc         : halt;
      end;
    gotoxy(stl,ria); write('*'); delay(70);
  until (stl=1)or(stl=80)or(ria=1)or(ria=25);
  gotoxy(25,13); writeln('B U M ! ! !  O H R A D A ! ! !');
  end;
BEGIN
inicializacia;
pohyb;
readln
END.
```

Spravme v programe ešte zopár úprav krásy. Po prvé, vyrobme okolo obrazovky ohradu (v dolnej časti po riadok 24). Potrebné kódy znakov sme si našli po zobrazení ASCII tabuľky. Môžete postupovať aj tak, že v konštantách budete mať priamo zobrazené príslušné semigrafické znaky v apostrofoch. druhá úprava krásy spočíva vo vypnutí zobrazovania kurzora, aby nám „nekazil celkový dojem“ (príkazy a potrebné premenné – registre pozná unit dos).

```
program pohyb_znaku_bez_podrzania_sipky2;
uses  crt,dos;
const ESC=chr(27);
      sipkaVPRAVO=chr(77);     {M}
      sipkaVLAVO =chr(75);     {K}
      sipkaHORE  =chr(72);     {H}
      sipkaDOLE  =chr(80);     {P}
var   stl,ria:byte;

procedure ohrada;
  var i:integer;
  begin
  write(chr(201)); for i:=2 to 79 do write(chr(205)); write(chr(187));
  for i:=2 to 23 do begin write(chr(186)); write(chr(186):79) end;
  write(chr(200)); for i:=2 to 79 do write(chr(205)); write(chr(188));
  end;

procedure skry_kurzor;
  var reg:registers;
  begin                {ukaz}
  reg.ah:=1;           {   1}
  reg.ch:=32;          {  13}
  reg.cl:=0;           {  40}
  intr($10,reg)
  end;

procedure inicializacia;
  begin
  clrscr;
  ohrada;
  skry_kurzor;
  stl:=40; ria:=12; gotoxy(stl,ria); write('*');
  end;

procedure pohyb;
  var stlznak:char;
  begin
  repeat
    if keypressed then stlznak:=readkey;
    gotoxy(stl,ria); write(' ');
    case stlznak of
      sipkaVPRAVO : inc(stl);
      sipkaVLAVO  : dec(stl);
      sipkaHORE   : dec(ria);
      sipkaDOLE   : inc(ria);
      Esc         : halt;
      end;
    gotoxy(stl,ria); write('*'); delay(70);
  until (stl=1)or(stl=80)or(ria=1)or(ria=24);
  gotoxy(25,13); write('B U M ! ! !  O H R A D A ! ! !');
  end;
BEGIN
inicializacia;
pohyb;
readln
END.
```

Predchádzajúci pohyb doplníme do hry. Nech program vygeneruje na náhodnom, ale vhodnom(!) mieste napríklad O, ktoré treba pohybujúcim sa znakom trafiť. Princíp je jednoduchý, pribudne testovanie, či sa nenachádza pohybujúci sa znak na pozícii znaku O.

```
program traf_znak;
uses  crt,dos;

const Esc=chr(27);
      sipkaVPRAVO=chr(77);     {M}
      sipkaVLAVO =chr(75);     {K}
      sipkaHORE  =chr(72);     {H}
      sipkaDOLE  =chr(80);     {P}

var   stl,ria,x,y:byte;

procedure skry_kurzor;
  var reg:registers;
  begin
  reg.ah:=1;
  reg.ch:=32;
  reg.cl:=0;
  intr($10,reg)
  end;

procedure ohrada;
  var i:integer;
  begin
  textcolor(white);
  write(chr(201)); for i:=2 to 79 do write(chr(205)); write(chr(187));
  for i:=2 to 23 do begin write(chr(186)); write(chr(186):79) end;
  write(chr(200)); for i:=2 to 79 do write(chr(205)); write(chr(188));
  end;

procedure inicializacia;
  begin
  randomize;
  skry_kurzor;
  clrscr;
  ohrada;
  x:=random(78)+2; y:=random(22)+2; gotoxy(x,y); write('O');
  stl:=40; ria:=12; gotoxy(stl,ria); write('*');
  end;

procedure pohyb;
  var stlznak:char;
  begin
  repeat
    if keypressed then stlznak:=readkey;
    gotoxy(stl,ria); write(' ');
    case stlznak of
      sipkaVPRAVO : inc(stl);
      sipkaVLAVO  : dec(stl);
      sipkaHORE   : dec(ria);
      sipkaDOLE   : inc(ria);
      Esc         : halt;
      end;
    gotoxy(stl,ria); write('*'); delay(100);
  until (stl=1)or(stl=80)or(ria=1)or(ria=24)or((stl=x)and(ria=y));
  if (stl=x)and(ria=y)
    then begin gotoxy(32,12); write('Z A S A H ! ! !') end
    else begin gotoxy(25,12); write('B U M ! ! !  O H R A D A ! ! !') end;
  end;
```

```
BEGIN
inicializacia;
pohyb;
readln
END.
```

Program môžete ďalej vylepšovať, napríklad nech sa náhodne pohybuje aj znak O. Ale to už nechávame na vašu šikovnosť.

Okrem dynamických (pohybujúcich sa) obrazcov je zaujímavé programovať aj statické obrazce. Väčšinou ide o požitie príkazov for, pričom

- ak obrazec ne je plný, cykly bývajú susedné
- ak je obrazec plný, cykly bývajú vnorené - vonkajší cyklus nastavuje riadok a vnútorný cyklus zabezpečuje pohyb v danom riadku.

Nasledujúci program demonštruje uvedené tvrdenia. Susedné cykly sú v procedúrach obdlznik1 a 2, stvorec a trojuholnik1 a 2; vnorené cykly sú v procedúrach obdlznik3 a trojuholnik3.

```
program OBRAZCE;
uses crt;
var n,i:integer;

procedure obdlznik1;
  begin
  clrscr;
  write('Strana obdlznika: '); readln(n);
  for i:=1 to n do write('*');
  writeln;
  for i:=2 to n-1 do writeln('*','*':n-1);
  for i:=1 to n do write('*');
  writeln;
  readkey
  end;

procedure obdlznik2;
  begin
  clrscr;
  write('Strana obdlznika: '); readln(n);
  for i:=1 to n do write('*');
  writeln;
  for i:=2 to n-1 do writeln('*','*':i-1,'*':n-i);
  for i:=1 to n do write('*');
  writeln;
  readkey
  end;

procedure stvorec;
  begin
  clrscr;
  write('Strana stvorca: '); readln(n);
  for i:=1 to n do write('* ');
  writeln;
  for i:=2 to n-1 do writeln('*','*':2*(n-1));
  for i:=1 to n do write('* ');
  writeln;
  readkey
  end;

procedure trojuholnik1;
  begin
  clrscr;
  write('Strana trojuholnika: '); readln(n);
  writeln('*');
```

```
  for i:=2 to n-1 do writeln('*','*':i-1);
  for i:=1 to n do write('*');
  writeln;
  readkey
  end;

procedure trojuholnik2;
  begin
  clrscr;
  write('Strana trojuholnika: '); readln(n);
  for i:=1 to n do write('*');
  writeln;
  for i:=2 to n-1 do writeln('*','*':n-i);
  writeln('*');
  readkey
  end;

procedure obdlznik3;
  var j:integer;
  begin
  clrscr;
  write('Strana obdlznika: '); readln(n);
  for i:=1 to n do
    begin
    for j:=1 to n do
      if i<>j then write('*') else write(' ');
    writeln
    end;
  readkey
  end;

procedure trojuholnik3;
  var j:integer;
  begin
  clrscr;
  write('Strana trojuholnika: '); readln(n);
  for i:=1 to n do
    begin
    for j:=1 to i do write('*');
    writeln
    end;
  readkey
  end;

begin
obdlznik1;
obdlznik2;
stvorec;
trojuholnik1;
trojuholnik2;
obdlznik3;
trojuholnik3;
end.
```

Pokúste sa vytvoriť program ktorý nakreslí rovnoramenný trojuholník s podstavou šírky n znakov, obrátenú pyramídu, kosoštvorec, štyri štvorce vedľa seba so stranou n znakov, štvorec s uhlopriečkami, plný obdĺžnik bez znakov na opačnej uhlopriečke, ako to je v procedúre obdlznik3.

Obrazce sú nekonečnou témou. Môžete sa pokúsiť naprogramovať aj pohyb napr. obdĺžnika (medzi jednotlivými polohami netreba mazať celú obrazovku).

Teraz trochu na oddýchnutie, aby sme však dlho neoddychovali, pobežia nám hodiny.

Príklad CRP-7: Vytvorte program simulujúci beh digitálnych hodín v strede obrazovky v tvare HH : MM : SS. Využite príkaz GetTime z unitu Dos.

Analýza: Program má vlastne do nekonečna (kým nestlačíme ľubovoľný kláves) čítať hodnoty systémového času pomocou príkazu gettime unitu dos a zobrazovať ich v strede obrazovky v predpísanom formáte. Problémom môže byť prechod z dvojciferného údaja na jednociferný, keď cifra z desiatok nebude premazaná – na to nesmieme zabudnúť.

```
program HODINY;
uses crt,dos;
var h,m,s,s100:word;
           reg:registers;
begin
{skry kurzor}
reg.ah:=1;
reg.ch:=32;
reg.cl:=0;
intr($10,reg);
clrscr;
repeat
  gettime(h,m,s,s100);
  gotoxy(33,12);
  if h<10 then write('0');
  write(h,' : ');
  if m<10 then write('0');
  write(m,' : ');
  if s<10 then write('0');
  write(s);
until keypressed;
end.
```

Doplňte program tak, aby hodiny pracovali ako budík („zazvonia“ v zadanom čase) prípadne časovač („cinknú“ po uplynutí zadaného času).

Na záver si ešte zopakujeme prácu s textom.

Príklad CWH-5: Vytvorte program, ktorý zistí, či zadané slovo je symetrické, t.j. či sa rovnako číta zľava aj sprava. Symetrické sú napríklad slová RADAR a ABBA.

Analýza: Zistiť, či zadané slovo je symetrické, znamená porovnať jeho prvý znak s posledným, ak sa rovnajú, porovnať jeho druhý znak s predposledným, ak sa rovnajú, porovnať..., až kým nenarazíme na dva rôzne znaky, alebo nie sme v strede slova (ľavá polovica slova sa rovná pravej polovici). Opakovane porovnávame vhodné znaky, preto treba použiť príkaz cyklu. Nevieme, kedy sa znaky už prestanú rovnať, je to cyklus s neznámym počtom opakovaní. Úloha je riešiteľná s príkazom while aj repeat, my sme sa rozhodli pre while. Keď sa po skončení cyklu s dvoma podmienkami ukončenia rozhodujete, pomocou ktorej testovať, prečo vlastne cyklus skončil, vyberte si vždy tú dôležitejšiu. V našom prípade to znamená testovať, či po skončení cyklu sa posledne testované písmená rovnajú alebo nie (if SLOVO[i]=SLOVO[DLZKA-i+1]), či sme už v strede slova, nie je natoľko dôležité.

```
program SYMETRICKE_SLOVO;
uses crt;
var SLOVO:string[80];
    i,DLZKA:byte;
```

```
begin
clrscr;
repeat
  write('Vloz slovo: '); readln(SLOVO);
  DLZKA:=length(SLOVO);
  i:=1;
  while (SLOVO[i]=SLOVO[DLZKA-i+1]) and (i<DLZKA div 2) do
    inc(i);
  if SLOVO[i]=SLOVO[DLZKA-i+1]
    then writeln('Text je symetricky')
    else writeln('Text nie je symetricky');
  writeln('KONIEC - stlac Esc')
until readkey=#27;
end.
```

Uvedený program možno zdokonaliť tým, že nebude rozlišovať medzi veľkými a malými písmenami a všetky ostatné znaky bude „ignorovať“. Potom aj text: „Jelenovi pivo nelej“ bude symetrický!

Analýza: Prvú požiadavku, nerozlišovať medzi veľkými a malými písmenami, možno vyriešiť pomocou funkcie upcase (známa vec). Druhú požiadavku, aby všetky ostatné znaky boli ignorované, možno dosiahnuť vytvorením nového textu, ktorý bude obsahovať len písmená. Využili sme množinový operátor in – je prvkom množiny (prvky množín sa uvádzajú v hranatých zátvorkách).

```
program SYMETRICKY_TEXT_2;
uses crt;
var TEXT,novyTEXT:string[80];
    i,j,DLZKA:byte;
begin
clrscr;
repeat
  write('Zadaj text: '); readln(TEXT);
  DLZKA:=length(TEXT);
  j:=0;
  for i:=1 to DLZKA do
    if TEXT[i] in ['A'..'Z','a'..'z']
      then begin
           inc(j);
           novyTEXT[j]:=upcase(TEXT[i])
           end;
  for i:=1 to j do
    TEXT[i]:=novyTEXT[i];
  DLZKA:=j;
  i:=1;
  while (TEXT[i]=TEXT[DLZKA-i+1]) and (i<DLZKA div 2) do
    inc(i);
  if TEXT[i]=TEXT[DLZKA-i+1]
    then writeln('Text je symetricky')
    else writeln('Text nie je symetricky');
  writeln('KONIEC - stlac Esc':45)
until readkey=#27;
end.
```

Neriešené úlohy na cykly:

1. Vytvorte program na výpočet súčinu A.B dvoch prirodzených čísel pomocou súčtu A + A + ...+ A (A ≥ B).
2. Vytvorte program výpočet mocniny s krokom x^2 (napr. $x^7 = x^2 . x^2 . x^2 . x$).
3. Vytvorte program na zistenie, či zadané číslo je prvočíslo.
4. Vytvorte program na vypísanie všetkých prvočísiel po zadané prirodzené číslo väčšie ako 1.
5. Vytvorte program na zistenie, či prirodzené číslo N < 341 je prvočíslo, ak viete, že platí veta:

   „N je prvočíslom práve vtedy , ak je deliteľom čísla 2^N-2 a N < 341“.
6. Vytvorte program na nájdenie všetkých prirodzených čísel po zadanú hranicu, ktorých ciferný súčet sa rovná zadanému číslu.
7. Vytvorte program, ktorý ako učebnú pomôcku vygeneruje tabuľku násobilky zadaného násobiteľa. Obmedzte hodnotu násobiteľa do 100 a násobenec nech má hodnoty z množiny {1,2,...,10 }.
8. Vytvorte program na prevod čísla z desiatkovej do dvojkovej sústavy.
9. Vytvorte program na prevod čísla z dvojkovej do desiatkovej sústavy.
10. Vytvorte program na prevod čísla z desiatkovej do šestnástkovej sústavy.
11. Vytvorte program na prevod čísla zo šestnástkovej do desiatkovej sústavy.
12. *Vytvorte program na zobrazenie podielu dvoch prirodzených čísel v tvare: {celá časť},{predperióda}{(perióda)}.

    Napr.: 3 / 2 = 1,5(0); 7 / 11 = 0,(63); 11 / 7 = 1,(571428) 31 / 60 = 0,51(6)

    (Platí veta: Predperióda je tvorená toľkými ciframi, koľkokrát je deliteľ deliteľný číslami 10, 5 a 2; napríklad deliteľ 60 je deliteľný 10 a 2).
13. Vytvorte program, ktorý zobrazí prevodnú tabuľku stupňov Celzia na Kelviny. Tabuľka začína 0^0C, končí hodnotou 100^0C s krokom po 10^0C. Tabuľka má mať hlavičku s nadpismi a pokúste sa o rámček zo semigrafických znakov (write(chr(...))).
14. Analogicky ako v predchádzajúcom príklade vytvorte prevodnú tabuľku uhlové stupne - radiány.
15. Vytvorte program , ktorý vypočíta dĺžku lomenej čiary určenej bodmi so súradnicami [x,y]. Na výpočet dĺžky úsečky (vzdialenosti dvoch bodov v rovine) použite Pythagorovu vetu.
16. Vytvorte program na nájdenie najväčšieho spoločného deliteľa troch prirodzených čísel.
17. Vytvorte program na vykreslenie grafu napr. funkcie sínus s osou x zvisle.
18. Vytvorte program na zistenie , či zadané prirodzené číslo je dokonalé. Číslo N je dokonalé, ak súčet všetkých jeho deliteľov menších ako N sa rovná N. Dokonalé je napríklad číslo 6.
19. Vytvorte obmenu programu CRP-5, keď budú postupne padať náhodne generované veľké písmená od prvého po 80. stĺpec. Na konci hry nech sa zobrazí úspešnosť v percentách (počet „zachytených“ písmen). Ďalšie varianty hry:
    - padanie sa bude postupne zrýchľovať, napr. po „zachytení“ 5 písmen za sebou
    - pád doplňte zvukovými efektami
    - hru doplňte ponukou: začiatočník – pokročilý – expert – koniec

- hru nech možno kedykoľvek ukončiť stlačením klávesu Esc
- vytvorte „nekonečný“ cyklus, keď po 80. stlpci pokračuje hra automaticky v 1. stĺpci
- písmená nech padajú v náhodne generovaných stĺpcoch
- padajúce písmená nech menia farbu
- atď.

Procedúry

Každý problém (okrem najjednoduchších) možno rozložiť na čiastkové podproblémy. Ich vyriešením a vykonaním v správnom poradí, dostávame riešenie daného problému. Programovacie jazyky na riešenie „podúloh“ ponúkajú podprogramy. 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 (každý poriadny program sa „rozpadá“ minimálne na zadanie vstupných hodnôt, výpočet a výstup získaných výsledkov) alebo
- ak potrebujeme vykonať rovnaký „výpočet“[1] viackrát s rôznymi vstupnými hodnotami (toto je praktický dôvod, aby sme neopisovali alebo nekopírovali tie isté príkazy viackrát v programe všade tam, kde potrebujeme vykonať rovnakú postupnosť príkazov).

Programovací jazyk TP má dva druhy podprogramov, procedúry a funkcie. V tejto kapitole sa budeme venovať procedúram.

Procedúry môžu byť:

- bez lokálnych objektov a bez parametrov,
- s lokálnymi objektami 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
   write(´Zadaj dve čísla: ´);
   readln(A,B);
   end;

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

procedure VYSTUP;
   begin
   writeln(A,' ',B)
   end;
```

[1] V súčasnosti sa už počítače na klasické výpočty používajú minimálne, častejšie na vyhľadávanie alebo triedenie dát.

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 hlavného programu:

```
BEGIN
clrscr;
VSTUP;
VYMENA;
VYSTUP;
readln
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 hlavného programu. Ako v celom TP, aj pri podprogramoch platí, že každý objekt musí byť najprv definovaný alebo deklarovaný a až potom ho môžeme použiť.

Procedúry s lokálnymi objektami 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 objektami má tvar:

procedure mp;
úsek definícií a deklarácií
príkazová časť
} blok

kde mp je meno procedúry.

Pamäťovo efektívnejší zápis procedúry VYMENA (s lokálnou premennou POM):

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

Volanie procedúry s lokálnymi objektami 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.

Procedúry s parametrami

Ak by sme procedúru POM 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 NAJVACSIE (A , B , C : integer; var MAX : integer);
procedure ZASIFRUJ (TEXT : string; POSUN : integer; var KOD : string);

Ak potrebujeme hodnoty do procedúry len doviesť, špecifikujeme 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, TEXT a POSUN.

Ak potrebujeme hodnoty z procedúry aj vyviesť, špecifikujeme 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, MAX a KOD.

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);
NAJVACSIE (5 , -3 , 7 , MAXIMUM); NAJVACSIE (X , Y , Z , Q);
ZASIFRUJ (´CEZAR´, 1 , POSLI); ZASIFRUJ (SPRAVA , N , KOD);

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.

Praktické použitie procedúr si ukážeme po prebratí jednorozmerného poľa.

Funkcie

Funkcia je špeciálnym prípadom procedúry. Procedúru môžeme zapísať ako funkciu, keď jej výsledkom je jedna jednoduchá hodnota alebo typ 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 MOCNINA (X : real; N : integer) : real;
function NACHADZA_SA : boolean;
function MAXIMUM (A , B , C : integer) : integer;

Príklad: Vytvorte podprogram na výpočet mocniny x^n, kde x je reálne číslo a n prirodzené číslo.

Analýza: Keďže výsledkom výpočtu je jedna jednoduchá hodnota, môžeme použiť aj funkciu. Na výpočet mocniny použijeme cyklus so známym počtom opakovaní: x^n = x.x.x. ... .x.

```
function MOCNINA (X : real; N : integer ) : real;
  var MOC : real;
      I : integer;
  begin
  MOC := 1;
  for I:=1 to N do
      MOC := MOC * I;
  MOCNINA := MOC
  end;
```

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: writeln (´Výsledok výpočtu mocniny: ´, MOCNINA (ZAKLAD , EXPONENT) : 12 : 10);
VYSLEDOK := MOCNINA (X , 3);

Príklad FNK-1: Vytvorte program na výpočet počtu kombinácií C(k,n) k-tej triedy z n prvkov $0 \le k \le n$.

Analýza: Platí vzorec $C(k,n) = \frac{n!}{(n-k)!.k!}$. Vidíme, že vo vzorci sa trikrát opakuje ten istý výpočet – výpočet faktoriálu (n! = 1.2.3. ... .n), len s rôznymi vstupnými hodnotami (n, n-k a k). Preto je výhodné použiť podprogram na výpočet faktoriálu a keďže výsledkom je jedna jednoduchá hodnota, konkrétne funkciu.

```
program KOMBINACIE;
uses  crt;
const Esc=chr(27);
var   N,K:integer;

function KOMB(N,K:integer):integer;

  function FAKT(N:integer):longint;       {lokalna funkcia}
    var I:integer;
        F:longint;
```

```
    begin
    F:=1;
    for I:=1 to N do
      F:=F*I;
    FAKT:=F
    end;

  begin     {príkazová časť funkcie KOMB }
  KOMB:=FAKT(N) div FAKT(N-K) div FAKT(K)
  end;

BEGIN
clrscr;
writeln('KOMBINACIE, KONIEC - Esc':50);
repeat
  write('Zadaj n a k ( n>=k>=0 ): ');readln(N,K);
  writeln('C (',N,',',K,') = ',KOMB(N,K));
until readkey=Esc;
END.
```

Program je skôr akademický, pretože hodnota n! už pre n = 13 prekročí maximálnu hodnotu typu longint (13! = 6 227 020 800).

Tak ako procedúry, aj funkcie si precvičíme až v ďalších kapitolách.

Rekurzia

Niektoré programovacie jazyky, vrátane TP, umožňujú používať veľmi elegantnú programovaciu techniku, tzv. rekurziu. Ak procedúra alebo funkcia vo svojom tele volá samu seba, hovoríme o priamej rekurzii; ak napríklad procedúra A volá procedúru B a naopak, hovoríme o nepriamej rekurzii.

Príklad REK-1: Vytvorte rekurzívnu funkciu na výpočet n!

Analýza: Vyjdeme z tzv. rekurentnej definície n-faktoriálu:

1. 0! = 1
2. n! = n.(n-1)! pre n>0

Ako vidieť z bodu 2, na vyčíslenie n! potrebujeme vykonať súčin n krát (n-1)!, pričom výpočet (n-1)! znamená rovnaký výpočet ako n!, len s hodnotou n zmenšenou o 1.

```
function FAKT ( N : integer) : longint;
   begin
   if N = 0
      then FAKT := 1                    { nerekurzívna vetva }
      else FAKT := N * FAKT ( N-1 )     { rekurzívna vetva }
   end;
```

Keďže z príkladu FNC-1 vieme, že klasický výpočet n! obsahuje cyklus, musí byť aj v príklade REK-1 niekde „schovaný“. K cyklu nás privedie poznanie, že počítač, ak nevie nejaký výraz, pre jeho zložitosť, vyčísliť, odkladá požiadavky na operácie do zásobníka a vracia sa k nim, keď ich už vie finalizovať. Preto pri výpočte napríklad 5! počítač odloží najprv požiadavku 5 krát „čosi“ (4! nepozná), potom 4 krát, 3 krát, 2 krát, 1 krát „čosi“, ale tu už „čosi“ je 0! a to je 1 (nerekurzívna vetva). Preto sa počítač vracia a vykonáva odložené operácie, t.j. 1.1.2.3.4.5 čím získa výsledok 120. Preto musíme vždy myslieť aj na ukončenie rekurzie, ukončenie volania danej procedúry alebo funkcie, čo zabezpečí nerekurzívna vetva.

Rekurzívny výpočet je časovo aj pamäťovo náročnejší ako iteračný výpočet (pomocou cyklu), niektoré problémy však rieši veľmi elegantne a relatívne jednoducho.

Príklad REK-2: Vytvorte rekurzívnu funkciu na výpočet mocniny x^n, x reálne číslo, n prirodzené číslo vrátane nuly.

Analýza: Potrebujeme znova poznať z matematiky rekurentnú definíciu mocniny, ktorá hovorí:

Pre $x \neq 0$:

1. $x^0 = 1$
2. $x^n = x \cdot x^{n-1}$ pre n>0

Vidíme, že v bode 2 máme na ľavej aj pravej strane rovnaký „problém“ – výpočet mocniny.

```
function MOCNINA ( X : real; N : word ): real;
  begin
  if N = 0
     then MOCNINA := 1
     else MOCNINA := X * MOCNINA ( X , N-1 )
  end;
```

Parameter X nie je nevyhnutný, X môže byť aj globálna premenná. Nezabudnite, že 0^0 nie je definované (ošetriť hneď na vstupe hodnôt X a N).

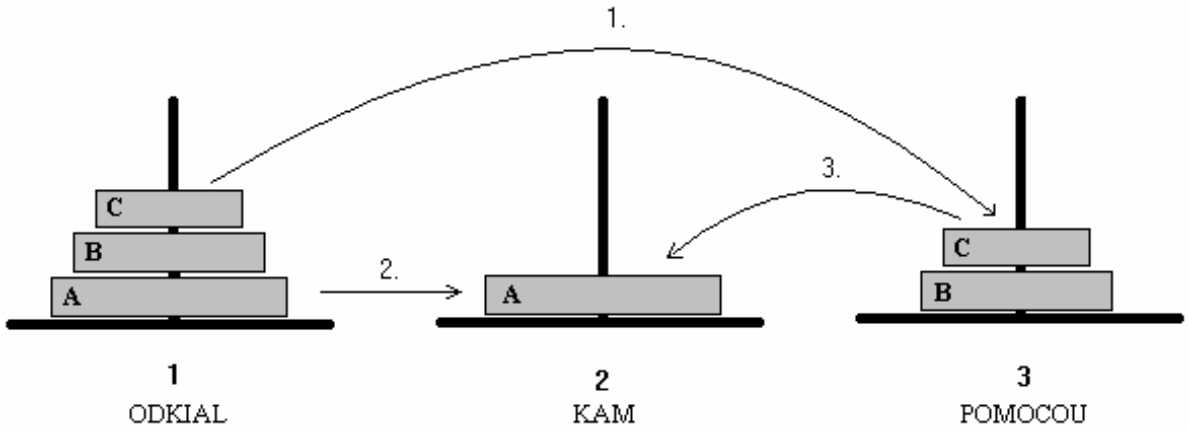
Príklad REK-3*: Vytvorte program riešiaci problém Hanojské veže.

Analýza: Ako ukážku sily a princípu práce rekurzie uvedieme známy problém nazvaný Hanojské veže. V starobylom kláštore ukrytom v ďalekom kúte Ázie vraj 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 platí:

- 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 (kolíky) sú označené 1, 2 a 3 (skutočné parametre). Pre označenie kolíkov sme zvolili premenné ODKIAL, KAM a POMOCOU (formálne parametre).



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,
2. potom môžeme disk A premiestniť z kolíka 1 na kolík 2 a
3. nakoniec presunúť vežu zvyšných N-1 diskov z kolíka 3 na kolík 2 pomocou kolíka 1.

Myšlienka použiť rekurziu by nám mohla napadnúť, ak si uvedomíme, že problém preniesť vežu s N diskami obsahuje v sebe problém preniesť vežu s N-1 diskami (ďalej už „len“ preložiť disk a naň vrátiť vežu s N-1 diskami).

Tu je program:

```
program HANOJ;
uses crt;
var  POCET:integer;

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

  procedure PRENES_DISK(Z,NA:integer);
    begin
    writeln(Z,' -> ',NA)
    end;

begin
if N>0
  then begin
       PRENES_VEZU(N-1,ODKIAL,POMOCOU,KAM);     {1}
       PRENES_DISK(ODKIAL,KAM);
       PRENES_VEZU(N-1,POMOCOU,KAM,ODKIAL);     {2}
       end;
end;

BEGIN
clrscr;
write('POCET DISKOV: ');
readln(POCET);
PRENES_VEZU(POCET,1,2,3);
readln;
END.
```

Jeho jednoduchosť asi prekvapí každého. Úspešnosť rekurzie spočíva v tom, čo sme uviedli v príklade REK-1, totiž, že ak počítač „nevie niečo vypočítať“, „odloží to“ do zásobníka. Tak počítač odkladá PRENES_VEZU 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 PRENES_DISK atď. Zložitosť výpočtu by mala byť zrejmá z výpisu, ktorý sme získali po doplnení vyššie uvedeného programu príkazmi na výpis, pričom prvé volanie procedúry PRENES_VEZU vo vetve then sme označili číslom 1 a druhé číslom 2.

```
POCET DISKOV = 3
Prenes vezu s 3 disk. pomocou volania 0  (prvé volanie z hlavného programu)
Prenes vezu s 2 disk. pomocou volania 1
Prenes vezu s 1 disk. pomocou volania 1
Prenes vezu s 0 disk. pomocou volania 1
1 -> 2
Prenes vezu s 0 disk. pomocou volania 2
1 -> 3
Prenes vezu s 1 disk. pomocou volania 2
Prenes vezu s 0 disk. pomocou volania 1
2 -> 3
Prenes vezu s 0 disk. pomocou volania 2
1 -> 2
Prenes vezu s 2 disk. pomocou volania 2
Prenes vezu s 1 disk. pomocou volania 1
Prenes vezu s 0 disk. pomocou volania 1
3 -> 1
Prenes vezu s 0 disk. pomocou volania 2
3 -> 2
Prenes vezu s 1 disk. pomocou volania 2
```

```
Prenes vezu s 0 disk. pomocou volania 1
1 -> 2
Prenes vezu s 0 disk. pomocou volania 2
```

Neriešené úlohy na rekurziu:

4. Vytvorte program na výpočet najväčšieho spoločného deliteľa (NSD) dvoch prirodzených čísel pomocou rekurzie, ak platí:

$$NSD(A,B) = \begin{cases} A & \text{ak } A=B \\ NSD(A-B,B) & \text{ak } A>B \\ NSD(A,B-A) & \text{ak } B>A \end{cases}$$

5. Vytvorte program na výpočet najväčšieho spoločného deliteľa (NSD) dvoch prirodzených čísel pomocou rekurzie, ak platí:

$$NSD(A,B) = \begin{cases} A+B & \text{ak } A=0 \text{ alebo } B=0 \\ NSD(A \bmod B,B) & \text{ak } A>B \\ NSD(A,B \bmod A) & \text{ak } B>A \end{cases}$$

6. Objasnite prácu rekurzívnej procedúry OTOC:

```
program ZRKADLO;
uses crt;
procedure OTOC;
  var z:char;
  begin
  read(z);
  if z<>'.'
    then OTOC
    else writeln;
  write(z)
  end;
BEGIN
clrscr;
writeln('Veta konci bodkou !':48);write(' ');
OTOC;
readkey
END.
```

7. Vytvorte program na výpočet ľubovoľného Fibonacciho čísla pomocou rekurzívnej a nerekurzívnej (iteračnej) funkcie a porovnajte časy výpočtu - urobte záver.

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

   vzorcami: $F_0 = 0$, $F_1 = 1$

   $F_i = F_{i-1} + F_{i-2}$ pre $i \geq 2$

8. Vytvorte podprogram na výpočet ľubovoľného člena Pascalovho trojuholníka, po zadaní nezáporných celých čísel n a k , ak platí:

   $B(n, 0) = B(n, n) = 1$

   $B(n, k) = B(n-1, k-1) + B(n-1, k)$ pre $1 \leq k \leq n$

9. Vytvorte funkciu CISLICA(n,k), ktorej výsledkom je k-ta číslica sprava prirodzeného čísla n.
10. *Vytvorte program, ktorý zadané prirodzené číslo n rozloží na prvočinitele, t.j. na prvočísla c_1, c_2, ...,c_k , pričom platí: $n = c_1 . c_2 . ... . c_k$.
11. *Vytvorte program, ktorý zistí, na koľko častí sa rozpadne štvorčekový papier po vystrihnutí niektorých štvorčekov.
12. *Vytvorte program na zistenie všetkých možných spôsobov vyplatenia zadanej sumy pri známych hodnotách použitých mincí (počet mincí rôznych hodnôt je neobmedzený).

Jednorozmerné pole

Predstavme si, že chceme vypočítať napríklad aritmetický priemer z veľkého počtu čísel a výstup má byť v tvare: Aritmetický priemer z čísel: *vypísané všetky čísla* je .... Na uchovanie väčšieho počtu údajov počas vykonávania programu s doteraz prebranými údajovými typmi nevystačíme. Pomôže nám nový údajový typ pole.

Konečnú skupinu rovnakých údajov (čísel, reťazcov a pod.) môžeme v programovaní výhodne uložiť do jednej štruktúrovanej premennej, do poľa. Pole sa skladá z pevného počtu zložiek rovnakého typu, pričom každú zložku možno sprístupniť pomocou indexu. Rôzne polia môžu obsahovať rôzne počty prvkov a aj ich hodnoty môžu byť rôznych typov (raz sú to čísla typu integer, druhýkrát reťazce typu string atď.). Preto rôzne polia predstavujú rôzne typy poľa, ktoré sa líšia počtom prvkov, typom zložky alebo typom indexu. Aby počítač vedel, akú veľkú pamäť má vyhradiť pre hodnoty premennej typu pole (skrátene pre pole), musíme v programe najprv typ pole definovať.

Definícia typu pole má tvar:

type mt = *array* [ti] *of* tz

kde mt je meno typu pole – identifikátor, ti je typ indexu – musí byť ordinálny typ a tz je typ zložky poľa.
Ako typ indexu sa najčastejšie používa tzv. interval, definovaný: min .. max, kde min a max sú hodnoty toho istého ordinálneho typu (medzi min a max sú práve dve bodky!).

Napríklad:	Pole vľavo uvedeného typu bude obsahovať:
type POLE1 = array [1..20] of integer;	20 čísel typu integer s indexami 1, 2, ..., 20
type POLE2 = array [-100..100] of boolean;	201 hodnôt false alebo true s indexami -100, -99, ...
type POCET = array [´A´..´Z´] of byte;	26 celých čísel od 0 do 255 s indexmi ´A´, ´B´,...,´Z´
type tVELA = array [byte] of string;	256 reťazcov s indexami 0, 1, 2 až 255
type HLUPOST = array [boolean] of boolean;	dva prvky s hodnotami false alebo true s indexami false a true

Uvedomte si, že súčasťou definície typu pole je aj pomenovanie daného typu. Ide o údajový typ s nami definovaným rozsahom, obsahom, indexami aj menom.

V úseku deklarácií premenných deklarujeme premenné typu pole rovnakým spôsobom ako iné premenné. Napríklad zápis: var A : POLE1 zavádza pole s názvom A typu POLE1. Sprístupnenie jednotlivých zložiek poľa A sa realizuje zápisom A [index], kde **index** je indexový výraz a musí byť rovnakého typu, ako je typ indexu z definície typu pole. Zápisu A [index] hovoríme **indexovaná premenná** a jej typ je určený typom zložky daného poľa.

Napríklad:

v poli A typu POLE1 znamená zápis:	A [1]	prvú zložku poľa A s hodnotou napríklad -7
	A [2]	druhú zložku poľa A
	A [I]	i-tu zložku poľa A
	A [20]	poslednú zložku poľa A
v poli B typu POLE2 znamená zápis:	B [-100]	prvú zložku poľa B s hodnotou napríklad true
	B [0]	101. zložku poľa B

v poli C typu POCET znamená zápis: C [´A´] prvú zložku poľa C s hodnotou napríklad 0

C [´Z´] poslednú zložku poľa C

Ak sú dve polia rovnakého typu, napríklad označené A a B, možno ich hodnoty odovzdávať naraz, napríklad príkazom B := A.

Premenná typu pole môže byť deklarovaná aj bez definície príslušného typu. Pre pole s názvom A by deklarácia mala tvar:

var A : *array* [ti] *of* tz

kde ti a tz majú rovnaký význam ako v definícii typu pole. Najprv definovať typ poľa a až potom deklarovať premennú daného typu má niekoľko výhod a preto odporúčame typ poľa najprv definovať. Napríklad v špecifikácii formálneho parametra v procedúre je dovolené na mieste typu parametra uviesť len meno poľa a nie zápis *array* [ti] *of* tz.

Pre programy s typom pole odporúčame používať podobný úsek definícií a deklarácií ako uvádzame nižšie:

```
program ...;
uses   crt;
const MAXPP = 20;              { definujeme konštantu udávajúcu maximálny počet zložiek - prvkov poľa }
type   POLE = array [ 1..MAXPP ] of integer;
var    A : POLE;
       N, I : integer;         { N bude aktuálny počet prvkov poľa A }
```

Načítanie prvkov poľa je čiastkový problém, ktorý je vhodné vyriešiť v procedúre. Ak v celom programe pracujeme len s jedným poľom, môžeme použiť procedúru bez parametrov, t.j. pracovať s globálnymi premennými. Ak v programe pracujeme s viacerými poliami, treba použiť parametre (ukážeme si v príklade neskôr). Princípom vstupu prvkov poľa je zapísať prvú zadanú hodnotu ako 1. prvok poľa, druhú ako 2. prvok poľa, atď. až po zápis posledného N-tého prvku poľa. N-krát sa opakuje rovnaká činnosť – zápis do poľa, preto použijeme cyklus so známym počtom opakovaní – príkaz for.

Príklad procedúry riešiacej načítanie prvkov globálneho poľa A s počtom prvkov N:

```
procedure VSTUP;       { bez parametrov }
  begin
  write(´Počet prvkov poľa: ´);
  readln(N);
  for I := 1 to N do          { Premenná I bude postupne nadobúdať hodnoty 1, 2, 3, ..., N }
    begin
    write(I:2,´. prvok: ´);    { Zápis I:2 spôsobí, že jednotky budú pod jednotkami a desiatky pod desiatkami }
    readln(A[I])
    end
  end;
```

Aj zobrazenie prvkov poľa je čiastkový problém riešiteľný v procedúre. Jednotlivé hodnoty poľa môžu byť na samostatných riadkoch (nevýhodné), oddelené medzerami alebo čiarkami.

Príklad procedúry slúžiacej na zobrazenie prvkov globálneho poľa A oddelených medzerami (predpokladáme, že výstupné hodnoty majú najviac 7 znakov):

procedure VYSTUP;

```
begin
for I := 1 to N do
  write(A[I]:8);          { keďže je v riadku 80 znakov, v každom bude zobrazených do desať hodnôt }
writeln
end;
```

Príklad procedúry slúžiacej na zobrazenie prvkov globálneho poľa A oddelených čiarkami:

procedure VYSTUP;

```
begin
for I := 1 to N-1 do              { čiarky majú byť za prvou až predposlednou hodnotou }
  write(A[I],´, ´);
writeln(A[N])
end;
```

Príklad POLE-1: Vytvorte program, ktorý načíta najviac 30 celých čísel z intervalu 0 až 999 a zobrazí ich.

Analýza: Požadovaný program stačí doslova poskladať z predchádzajúcich procedúr, len s malými úpravami: MAXPP treba zväčšiť na 30 a typ zložky poľa môže byť byte (potom v prvej procedúre VYSTUP vystačíme aj s formátom A[I]:4). Neznáma je už len príkazová časť hlavného programu:

```
BEGIN
clrscr;
VSTUP;                          { podľa „chuti“ môžete doplniť príkaz clrscr; }
writeln(´Prvky pola:´);
VYSTUP;
readln
END.
```

Odporúčame uložiť si tento program do súboru s názvom POLE a máme základ pre mnoho programov pracujúcich s poľom – každý by predsa mal mať vstup a výstup prvkov poľa. Po otvorení súboru POLE.PAS ho treba cez položku File – Save as... premenovať na aktuálny názov a doplniť o procedúru riešiacu zadanú úlohu.

Často prvky poľa môžu byť aj náhodne počítačom vygenerované čísla, stačí, ak zadáme len ich počet. Procedúra VSTUP_NAHODNE za nás „vyberie“ prvky poľa:

procedure VSTUP_NAHODNE;

```
begin
write(´Počet prvkov poľa: ´);
readln(N);
for I := 1 to N do
  A[I]:=random(10)              { pole A bude obsahovať celé čísla od 0 po 9 }
end;
```

V hlavnom programe za BEGIN treba vložiť príkaz randomize, aby sme znáhodnili výber prvkov poľa po každom spustení programu.

Prácu s poľom si precvičíme najmä na činnosti nazvanej vyhľadávanie. **Vyhľadávanie** v poli je činnosť s cieľom zistiť, či zložka so zadanými vlastnosťami sa nachádza v poli. Cieľom môže byť aj zistenie, koľkokrát sa zložka so zadanými vlastnosťami nachádza v poli, na ktorých miestach, prípadne požiadavka, ak sa zložka v poli nenachádza, vložiť ju za posledný prvok poľa a pod.

Príklad POLE-2: Vytvorte podprogram na zistenie najväčšieho prvku daného poľa.

Analýza: Maximum hľadajme v poli celých čísel. Na začiatku hľadania je najväčším prvkom prvý prvok poľa (jeho hodnotu uložíme do premennej MAX). Potom musíme zistiť, či druhý prvok poľa nie je väčší, ak áno, našli sme nové maximum (do premennej MAX musíme zapísať hodnotu druhého prvku poľa). Podobne porovnáme tretí prvok poľa s MAX a ak treba aktualizujeme MAX (zmeníme hodnotu MAX na hodnotu tretieho prvku poľa), potom štvrtý prvok, atď., až kým neprejdeme celé pole. Prejsť celé pole od prvého resp. druhého po posledný prvok znamená použiť príkaz for.

```
procedure MAXIMUM;
  var MAX:integer;           { lokálna premenná MAX }
  begin
  MAX:=A[1];                 { priradenie počiatočnej hodnoty premennej MAX }
  for I:=2 to N do           { zabezpečí prejdenie celého poľa od druhého prvku}
    if A[I]>MAX then MAX:=A[I];
  writeln('Z cisel:');
  VYSTUP;                          {zobrazí prvky poľa }
  writeln('je najvacsie ',MAX);
  end;

BEGIN
clrscr;
VSTUP;
MAXIMUM;
readln
END.
```

Procedúru MAXIMUM môžeme napísať aj ako funkciu, keďže výsledkom danej procedúry je jedna jednoduchá hodnota. Funkciu je výhodné použiť, ak chceme maximum vyviesť.

Program od funkcie MAXIMUM:

```
function MAXIMUM:integer;
  var MAX:integer;
  begin
  MAX:=A[1];
  for I:=2 to N do
    if A[I]>MAX then MAX:=A[I];
  MAXIMUM:=MAX;                    {priradenie najväčšej hodnoty menu funkcie,}
  end;                             {cez ktoré sa vyvezie daná hodnota}

BEGIN
clrscr;
VSTUP;
VYSTUP;
writeln('je najvacsie ',MAXIMUM);  {zápis funkcie na mieste,kde chceme zobraziť}
readln                             {výsledok funkcie}
END.
```

Príklad POLE-3: Vytvorte podprogram na zistenie, koľkokrát sa zadaný prvok nachádza v poli naviac 100 celých čísel.

Analýza: Zistiť, koľkokrát sa zadaný prvok nachádza v poli znamená prejsť celé pole a vždy keď sa prezeraný prvok poľa rovná hľadanému, zväčšiť počet výskytov o jednotku.

```
procedure ZISTI_POCET;
  var HLPRVOK,POCET:integer;
  begin
  write('Zadaj prvok, ktoreho pocet vyskytov mam zistit: ');
  readln(HLPRVOK);
  POCET:=0;
```

```
  for I:=1 to N do
    if A[I]=HLPRVOK then POCET:=POCET+1;      { analogicky inc(POCET); }
  if POCET=0
    then writeln('Hladany prvok sa v poli nevyskytuje!')
    else writeln('Hladany prvok sa v poli vyskytuje ', POCET,'-krat.')
  end;
BEGIN
clrscr;
VSTUP;
ZISTI_POCET;
readln
END.
```

Procedúra môže byť opäť napísaná aj ako funkcia, jej použitie vzhľadom na požiadavku zadať hľadaný prvok je trochu komplikovanejšia:

```
function ZISTI_POCET(HLPRVOK:integer):integer; {parameter nahradzovaný hodnotou}
  var POCET:integer;
  begin
  POCET:=0;
  for I:=1 to N do
    if A[I]=HLPRVOK then inc(POCET);
  ZISTI_POCET:=POCET
  end;

BEGIN
clrscr;
VSTUP;
write('Zadaj prvok, ktoreho pocet vyskytov mam zistit: ');
readln(HLADANY_PRVOK);              {HLADANY_PRVOK musí byť globálna premenná}
writeln('Hladany prvok sa v poli vyskytuje ', ZISTI_POCET(HLADANY_PRVOK),'-
krat.');
readln
END.
```

Príklad POLE-4: Vytvorte podprogram na zistenie, či sa prvok s danou hodnotou vyskytuje v poli.

Analýza: Kým doteraz sme museli prejsť celé pole, v tejto úlohe môžeme prehľadávanie poľa ukončiť po nájdení hľadanej hodnoty v poli. Z algoritmického hľadiska to znamená použiť časovo efektívnejší cyklus s neznámym počtom opakovaní. Výsledkom prehľadávania má byť len zistenie, či sa prvok v poli nachádza (vyvezie sa hodnota true) alebo nenachádza (vyvezie sa hodnota false).

```
function NACHADZA_SA:boolean;
  begin
  I:=0;
  repeat
    inc(I)
  until (A[I]=HLPRVOK) or (I=N);
  NACHADZA_SA:=A[I]=HLPRVOK              { Tento zápis nahrádza príkaz if }
  end;

BEGIN
clrscr;
VSTUP;
write('Zistit vyskyt prvku s hodnotou: ');
readln(HLPRVOK);
writeln('Prvok s hodnotou ',HLPRVOK,' sa v poli:');
VYSTUP;
if NACHADZA_SA
  then writeln('nachadza.')
  else writeln('nenachadza.');
readln
END.
```

Príklad POLE-4b: Ako zaujímavosť uvádzame rovnaký problém – zistiť, či sa prvok s danou vlastnosťou vyskytuje v poli, pričom však vieme, že pole je utriedené. Pod utriedeným poľom rozumieme vzostupné utriedenie, t.j. najmenší prvok je prvý a najväčší prvok je posledný, preto platí: A[I]≤A[I+1] pre všetky prípustné hodnoty I. Použiť by sme mohli aj predchádzajúcu funkciu NACHADZA_SA, ktorá používa tzv. lineárne vyhľadávanie – pole prehľadáva od prvého prvku prvok za prvkom, my však chceme časovo efektívnejší algoritmus (ak sa v tisíc prvkovom poli nachádza hľadané číslo napr. na 999 mieste, lineárne vyhľadávanie musí prezrieť 999 prvkov, nasledujúca funkcia však najviac 10!).

Analýza: Princípom binárneho vyhľadávanie, lebo tak je toto vyhľadávanie pomenované, je určenie stredu prehľadávanej oblasti a zistenie, či sme našli hľadaný prvok a ak nie, či sa nachádza v jeho dolnej (ľavej) časti alebo v hornej (pravej) časti. Tento postup sa opakuje dovtedy, kým sa nenájde hľadaný prvok, alebo keď už nie je čo prehľadávať (dolná hranica prehľadávanej oblasti je nad jej hornou hranicou).

```
function NACHADZA_SA:boolean;
  var DH,HH,S:integer;
  begin
  DH:=1; HH:=N;
  repeat
    S:=(DH+HH) div 2;
    if A[S]<>HLPRVOK
      then if A[S]<HLPRVOK
             then DH:=S+1
             else HH:=S-1
  until (A[S]=HLPRVOK) or (DH>HH);
  NACHADZA_SA:=A[S]=HLPRVOK
  end;
```

Ďalším typom úloh môžu byť úlohy zistiť, na ktorých miestach v poli sa vyskytuje prvok so zadanou vlastnosťou. Úloha môže byť formulovaná: zistiť prvý výskyt, všetky výskyty, posledný výskyt a pod.

Príklad POLE-5: Vytvorte podprogram na nájdenie prvého výskytu prvku so zadanou vlastnosťou.

Analýza: Nájsť miesto výskytu znamená vlastne určiť index prvku, ktorý má požadovanú vlastnosť (ak sú indexy poľa 1, 2, ..., N). Ak sa hľadaná hodnota v poli nevyskytuje, nech sa vyvezie 0. Keďže ide o jednu jednoduchú hodnotu, môžeme použiť funkciu. My sme sa rozhodli pre procedúru, aby sme ukázali použitie procedúry namiesto funkcie a aj preto, že funkčný zápis by sme museli použiť dvakrát (všade tam v hlavnom programe, kde sme použili premennú MIESTO), čo nie je efektívne. Ak sa chceme vyhnúť viacnásobnému použitiu funkčného zápisu s nezmenenými hodnotami, môžeme výsledok funkcie uložiť v hlavnom programe do premennej a ďalej pracovať len s touto premennou. V nasledujúcej časti programu sú premenné HLPRVOK a MIESTO globálne, nezabudnite ich deklarovať.

```
procedure PRVY_VYSKYT;
  begin
  I:=0;
  repeat
    inc(I)
  until (A[I]=HLPRVOK) or (I=N);
  if A[I]=HLPRVOK
    then MIESTO:=I
    else MIESTO:=0
  end;
```

```
BEGIN
clrscr;
VSTUP;
write('Zistit miesto prveho vyskytu prvku s hodnotou: ');
readln(HLPRVOK);
clrscr;
writeln('Prvok s hodnotou ',HLPRVOK,' sa v poli:');
VYSTUP;
PRVY_VYSKYT;
if MIESTO>0
  then writeln('nachadza na ',MIESTO,'. mieste.')
  else writeln('nenachadza.');
readln
END.
```

Veľmi zaujímavé sú aj kombinácie predchádzajúcich úloh, ako napríklad zistiť, či sa prvok s požadovanou vlastnosťou nachádza v poli a ak áno, koľkokrát, prípadne na ktorých miestach, prípadne aj koľkokrát a na ktorých miestach. Tieto kombinované úlohy väčšinou neznamenajú riešiť každú úlohu v samostatnom cykle (neefektívne), ale pri jednom prechode poľom možno „vyriešiť" všetky úlohy. Úlohy podobného typu sme dali medzi neriešené úlohy, teraz si uveďme aspoň jednu zložitejšiu.

Príklad POLE-6: Vytvorte program na zistenie najmenšieho prvku v poli, koľkokrát sa vyskytuje a na ktorých miestach v poli.

Analýza: Program nechávame na „samoštúdium". Všimnite si priradenie počiatočných hodnôt v procedúre MINIMUM a ako sme sa „pohrali" s výstupom (VYSTUP2).

```
program POLE_6;
uses  crt;
const MAXPP=20;
type  POLE=array[1..MAXPP] of integer;
var   A,VYSKYT:POLE;
      N,I,MIN,POCET:integer;

procedure VSTUP;
  begin
  write('Pocet prvkov pola: ');
  readln(N);
  for I:=1 to N do
    begin
    write(I:2,'. prvok: ');
    readln(A[I]);
    end;
  end;

procedure VYSTUP1;
  begin
  for I:=1 to N-1 do
    write(A[I],', ');
  writeln(A[N]);
  end;

procedure MINIMUM;
  begin
  MIN:=A[1]; POCET:=1; VYSKYT[1]:=1;      { priradenie počiatočných hodnôt }
  for I:=2 to N do
    if A[I]<=MIN
      then if A[I]=MIN        { prvok s hodnotou rovnou momentálnemu minimu }
             then begin
                  inc(POCET);
                  VYSKYT[POCET]:=I
                  end
```

```
            else begin            { nové minimum }
                 MIN:=A[I];
                 POCET:=1;
                 VYSKYT[1]:=I
                 end
  end;

procedure VYSTUP2;
  begin
  writeln('je najmensim prvok ',MIN);
  write('a vyskytuje sa ',POCET,'-krat ');
  if POCET=1
    then write('na mieste: ',VYSKYT[1])
    else begin
         write('na miestach: ');
         for I:=1 to POCET do write(VYSKYT[I],'  ')
         end
   end;

BEGIN
clrscr;
VSTUP;
clrscr;
writeln('V poli:');
VYSTUP1;
MINIMUM;
VYSTUP2;
readln
END.
```

Ďalšiu techniku, alebo ak chcete „fintu“, ktorú by mal ovládať každý programátor, je použitie tzv. nárazníka. Jeho princíp a použitie si ukážeme na nasledujúcom príklade:

Príklad POLE-7: Vytvorte podprogram, ktorý zistí, či sa hľadaný prvok nachádza v poli, a ak nie, vloží ho za posledný prvok.

Analýza: Princíp použitia nárazníka spočíva vo vložení hľadaného prvku do poľa, v tomto prípade za posledný prvok. To zabezpečí zastavenie prehľadávania (prvok sa určite v takto upravenom poli nachádza) najneskôr na nárazníku. Ak sa prehľadávanie poľa zastaví skôr, hľadaný prvok sa nachádza už v pôvodnom poli. Ak sa hľadaný prvok v poli nenachádza, už ho máme vložený za posledný prvok, len nesmieme zabudnúť zväčšiť počet prvkov poľa (hodnotu N).

```
procedure NARAZNIK;
  begin
  A[N+1]:=HLPRVOK;       { vlozili sme naraznik }
  I:=0;
  repeat
    I:=I+1
  until A[I]=HLPRVOK;
  if I=N+1 then N:=N+1   { prvok sa v poli nenachadzal, pridany dalsi prvok }
  end;

BEGIN
clrscr;
VSTUP;
write('Zistit vyskyt prvku s hodnotou: ');
readln(HLPRVOK);
writeln('Povodne pole:'); VYSTUP;
NARAZNIK;
writeln('Nove pole:'); VYSTUP;
readln
END.
```

Na nasledujúcom príklade „z trochu iného súdka“ si ukážeme, aké je dôležité dobre si zvoliť typ pole.

Príklad POLE-8: Vytvorte program, ktorý po zadaní textu prekonvertujte malé písmená v texte na veľké a zobrazí tabuľku počtu výskytov jednotlivých písmen abecedy v texte.

Analýza: Pole zvolíme tak, aby indexami poľa boli postupne všetky veľké písmená abecedy a hodnotami daného poľa budú čísla udávajúce počet výskytov jednotlivých písmen v texte.

```
program POLE_8;
uses  crt;
type  POLE=array['A'..'Z'] of byte;
var   TEXT:string;
      POCET:POLE;

procedure VSTUP;
  procedure KONVERZIA;
    var I:byte;
    begin
    for I:=1 to length(TEXT) do
      TEXT[I]:=upcase(TEXT[I])           { malé písmená v TEXTe zmení na veľké }
    end;
  begin
  write('Zadaj text: '); readln(TEXT);
  KONVERZIA
end;

procedure ZISTI_POCTY;
  var I:byte;
  procedure VYNULUJ;
    var ZNAK:char;
    begin
    for ZNAK:='A' to 'Z' do
      POCET[ZNAK]:=0
    end;
  begin
  VYNULUJ;
  for I:=1 to length(TEXT) do
    if TEXT[I] in ['A'..'Z']
      then inc(POCET[TEXT[I]]);   { TEXT[I] predstavuje konkrétne písmeno textu }
  end;

procedure VYSTUP;
  var ZNAK:char;
  begin
  writeln('Znak          Pocetnost':45);
  for ZNAK:='A' to 'Z' do
    begin
    writeln(ZNAK:24,POCET[ZNAK]:17);
    if ZNAK='W' then begin readkey; clrscr end
    end
  end;

BEGIN
clrscr;
VSTUP;
ZISTI_POCTY;
VYSTUP;
readln
END.
```

Typové konštanty

Typové konštanty môžeme prirovnať k premenným, ktorých počiatočné hodnoty sú uvedené už v úseku definícií a deklarácií. Môžeme ich aj používať rovnako ako premenné takého istého typu.

Deklarácia typovej konštanty má tvar:

const identifikátor_premennej : identifikátor_typu = typová_konštanta

Napríklad:

```
const   SYM : boolean = TRUE;
        stringANO : string[3] = ´ANO´ ;
        A : array[1..10] of integer = (15,2,-4,6,9,12,0,-7,6,11);
        CISLICE : array[0..9] of char = (´0´,´1´,´2´,´3´,´4´,´5´,´6´,´7´,´8´,´9´);
        CISLICE : array[0..9] of char = ´0123456789´;      { táto deklarácia je totožná s predchádzajúcou }
```

Príklad POLE-9: V príklade VCS-1 sme v programe použili na výpočet počtu dní do konca roka (dkr) „otrasný“ zápis v príkaze case. Pri použití typovej konštanty poľa sa zápis podstatne zjednoduší. Do ukážky sme pridali aj výpočet počtu dní od začiatku roka (ozr).

```
const dvm:array[1..12] of 1..31=(31,28,31,30,31,30,31,31,30,31,30,31);
      { dvm – počet dní v mesiaci }
...
writeln;
dkr:=dkm;
for i:=12 downto m+1 do
  dkr:=dkr+dvm[i];
if (m<2) and priestupny then inc(dkr);
write('Do konca roka ');
case dkr of
      1: writeln('zostáva 1 deň.');
  2,3,4: writeln('zostávajú ', dkr,' dni.');
  else writeln('zostáva ', dkr,' dní.')
end;
{ výpočet počtu dní od začiatku roka }
writeln;
ozr:=d;
for i:=1 to m-1 do
  ozr:=ozr+dvm[i];
if (m>2) and priestupny then inc(ozr);
write('Od zaciatku roka ');
case ozr of
      1: writeln('ubehol 1 deň.');
  2,3,4: writeln('ubehli ', ozr,' dni.');
  else writeln('ubehlo ', ozr,' dní.')
end;
readln
end.
```

Samozrejme, ak poznáme počet dní do konca roka, počet dní od začiatku roka sa dá vypočítať aj jednoduchšie.

Príklad POLE-10: Vytvorte program na prevod správy – textu do morseovej abecedy.

Analýza: Príklad je typický pre použitie typovej konštanty pole. Doplnili sme ho aj o zvukový výstup. Program ako „nepovinný“ ponechávame na samoštúdium.

```
program POLE_10;
uses  crt;
const TAB:array['A'..'Z'] of string[4] = ('.-','-...','-.-.','-..','.','..-.',
'--.','....','..','.---','-.-','.-..','--','-.','---','.--.','--.-','.-.','...',
' ','...-', '...-','.--','-..-','-.--','--..');
      takt=100;
      bodka=takt;
      ciarka=3*takt;
      medzi_znakmi=takt;
      medzi_bc=takt;
      Hz=1000;
var   s:string;
      i:byte;

procedure zvuk(MSznak:string);
  var i:byte;
  begin
  write(MSznak);
  for i:=1 to length(MSznak) do
    begin
    if MSznak[i]='|'
      then Delay(medzi_znakmi)
      else begin
           Sound(Hz);
           if MSznak[i]='.'
             then Delay(bodka)
             else Delay(ciarka);
           NoSound
           end;
    Delay(medzi_bc)
    end
  end;

BEGIN
clrscr;
write('Zadaj retazec: '); readln(s);
for i:=1 to length(s) do
  case s[i] of
    'A'..'Z','a'..'z': zvuk( TAB[ UpCase(s[i]) ] + '|' );
    ' '              : writeln
  end;
readln
END.
```

Okrem vyhľadávania sa často údaje aj triedia. Utriediť prvky poľa znamená vykonať činnosť, po končení ktorej pre prvky poľa napr. A platí: A[I] ≤ A[I+1] pre všetky dovolené hodnoty I. Triediacich algoritmov je veľa. Každý by mal ovládať aspoň ten najjednoduchší, ktorým môže byť napríklad bubblesort.

Príklad POLE-11: Vytvorte podprogram na utriedenie číselného poľa bubblesortom.

Analýza: Princípom práce bubblesortu je prechádzať poľom a porovnávať dva vedľa seba stojace prvky; ak je ľavý prvok väčší ako pravý, vymeniť ich. Ak si predstavíte túto činnosť, malo by byť zrejmé, že po prvom prechode poľom je najväčší prvok na svojom miesta, t.j. na konci poľa. Po druhom prechode poľom je na svojom mieste druhý najväčší prvok, atď. Pri n-prvkovom poli musí byť po n-1 prechodoch pole utriedené. Podstatou algoritmu sú dva vnorené cykly: vonkajší cyklus zabezpečuje pri n-prvkovom poli n-1 prechodov daným poľom (pp – počet prechodov) a vnútorný cyklus zabezpečuje pohyb v poli od prvého po n-tý prvok (premenná i). Vnútorný cyklus možno ešte zefektívniť, ak zohľadníme aj to, že po prvom prechode poľom je najväčší prvok na svojom mieste a preto pri ďalších prechodoch poľom už nemusíme ísť až na koniec poľa;

analogicky po druhom prechode poľom je už aj druhý najväčší prvok poľa na svojom mieste a preto v nasledujúcich prechodoch poľom nemusíme porovnávať už ani predposledný prvok, atď.

1. prechod	treba porovnať n prvkov	t.j. ísť po n-1. prvok
2. prechod	treba porovnať n-1 prvkov	t.j. ísť po n-2. prvok
3. prechod	treba porovnať n-2 prvkov	t.j. ísť po n-3. prvok
...		
pp. prechod		treba ísť po n-pp. prvok

```
procedure bubblesort;
  var pp:integer;
  procedure vymena;
    var pom:real;
    begin
    pom:=a[i]; a[i]:=a[i+1]; a[i+1]:=pom;
    end;
  begin
  for pp:=1 to n-1 do         { počet prechodov }
    for i:=1 to n-pp do       {pohyb v poli }
      if a[i]>a[i+1] then vymena;
  end;

BEGIN
clrscr;
vstup;
writeln('Povodne pole:'); vystup;
bubblesort;
writeln('Utriedene pole:'); vystup;
readln
END.
```

Aspoň jedným príkladom si pripomenieme, že aj údajový typ string (reťazec) je špeciálnym prípadom jednorozmerného poľa. Teoreticky ide o pole typu array [0..255] of char, ktorého prvky môžeme načítať „naraz“ príkazom read a zobraziť tiež „naraz“ bez cyklu for príkazom write (znak s indexom 0 má osobitnú funkciu – je v ňom uložená aktuálna dĺžka reťazca). Môžeme však sprístupniť i-ty znak reťazca indexovanou premennou ako u poľa.

Príklad POLE-12: Vytvorte program, ktorý zistí, či zadané dva reťazce sa skladajú z tých istých písmen a v rovnakom počte. Takéto reťazce nazývame anagramy.

Analýza: Najľahšie sa zistí, či majú zadané reťazce rovnakú dĺžku. Ak ich dĺžky sú rôzne, nemôžu byť anagramy. Ak sa ich dĺžky rovnajú, musíme zistiť, či sa skladajú z rovnakých písmen a v rovnakom počte. Znamená to zobrať prvý znak v prvom reťazci a hľadať jeho výskyt v druhom reťazci, ak je všetko „v poriadku“, zobrať druhý znak z prvého reťazca atď. Teda cyklus v cykle, kde vonkajší cyklus nastavuje prvý, druhý,... znak prvého reťazca a vnorený cyklus „prechádza“ druhým reťazcom a hľadá v ňom zodpovedajúci znak. Najefektívnejšie sú cykly s neznámym počtom opakovaní. Ďalším problémom je zabezpečiť, aby sme zistili, či sa písmená zhodujú aj v počtoch. Najjednoduchšie je asi „vyrábať“ pri prechode druhým reťazcom postupne reťazec totožný s prvým. Pre ozrejmenie programu si spravte pomocný výpis upravovaného druhého reťazca príkazom write(y), ako to je v poznámke programu.

```
program POLE_12;
uses crt;
var  slovo1,slovo2:string[80];

function anagram(x,y:string):boolean;
  var i,j,dlzkax,dlzkay:byte;
      anag:boolean;
      pom:char;
  begin
  dlzkax:=length(x); dlzkay:=length(y);
  if dlzkax=dlzkay
    then begin
         anag:=true;
         i:=1;
         while anag and (i<=dlzkax) do              { vonkajší cyklus }
           begin
           j:=i;
           while (x[i]<>y[j]) and (j<dlzkay) do     { vnútorný cyklus }
             j:=j+1;
           anag:=x[i]=y[j]; pom:=y[i]; y[i]:=y[j]; y[j]:=pom;
           i:=i+1;                          { doplňte príkaz writeln(y) }
           end;
         anagram:=anag
         end
    else anagram:=false
  end;

BEGIN
clrscr;
repeat
  write('Prve  slovo: '); readln(slovo1);
  write('Druhe slovo: '); readln(slovo2);
  if anagram(slovo1,slovo2)
    then writeln('JE TO ANAGRAM':44)
    else writeln('NIE JE TO ANAGRAM':46);
  writeln('KONIEC - stlac Esc');
  writeln
until readkey=chr(27);
END.
```

Ak sa niekomu zapáčil pohyb znaku po obrazovke, môže pokračovať v príklade CRP-6, a pokúsiť sa naprogramovať jednu z prvých hier na počítačoch – červíka Wurmi.

Príklad POLE-13: Vytvorte program – hru, v ktorej budú na obrazovke náhodne vygenerované krížiky (x) - jed a listy - potrava (♠ chr(6)). Šípkami na riadenie pohybu kurzora nech sa ovláda pohyb červíka Wurmi (znak chr(1)), ktorého úlohou je „požierať“ listy a vyhýbať sa jedu. Ak narazí na jed, hra končí. Hra končí aj „narazením do okrajov obrazovky“.

Analýza: Pri programovaní tejto hry použite príklad CRP-6, v ktorom máme vyriešené ovládanie pohybu znaku na obrazovke vrátane ošetrenia „trafenia“ znaku na obrazovke a „narazenie do okraja obrazovky“. Ďalšie zlepšenie hry si vyžaduje použitie polí, aby bolo možné zapamätať si pozície (x-ové a y-nové súradnice) jedu a potravy. Po každom premiestnení Wurmiho treba testovať, či „nestojí“ na jede alebo potrave. Ak stojí na potrave, treba započítať, že „zjedol“ ďalší list. Hra končí, keď:

- „zožerie“ všetky vygenerované listy
- „zožerie“ jed
- „narazí“ do okrajov obrazovky.

```
program POLE_13;
uses  crt,dos;

const pocet=5;                { počet vygenerovaných jedov a listov }
      ESC=chr(27);
      sipkaVPRAVO=chr(77);
      sipkaVLAVO =chr(75);
      sipkaHORE  =chr(72);
      sipkaDOLE  =chr(80);
      list       =chr(6);

var   stl,ria,body:integer;
      px,py,jx,jy:array[1..pocet] of integer;

procedure skry_kurzor;
  var reg:registers;
  begin
  reg.ah:=1;
  reg.ch:=32;
  reg.cl:=0;
  intr($10,reg)
  end;

procedure ohrada;
  var i:integer;
  begin
  textcolor(white);
  write(chr(201)); for i:=2 to 79 do write(chr(205)); write(chr(187));
  for i:=2 to 23 do begin write(chr(186)); write(chr(186):79) end;
  write(chr(200)); for i:=2 to 79 do write(chr(205)); write(chr(188));
  end;

procedure generuj_potravu_jed;
  var i:integer;
  begin
  for i:=1 to pocet do
    begin
    px[i]:=random(78)+2; py[i]:=random(22)+2;  { generuje potravu }
    jx[i]:=random(78)+2; jy[i]:=random(22)+2;  { generuje jed }
    end
  end;

procedure vykresli_potravu_jed;
  var i:integer;
  begin
  for i:=1 to pocet do
    begin
    gotoxy(px[i],py[i]); write(list);
    gotoxy(jx[i],jy[i]); write('x');
    end;
  end;

procedure inicializacia;
  var limit:integer;
  begin
  randomize;
  generuj_potravu_jed;
  skry_kurzor;
  clrscr;
  ohrada;
  vykresli_potravu_jed;
  stl:=40; ria:=12; gotoxy(stl,ria); write(chr(1));
  body:=0
  end;
```

```
function jed:boolean;   { testuje, či Wurmi nie je na poličku jed }
  var i:integer;
  begin
  i:=0;
  repeat
    inc(i)
  until ((jx[i]=stl) and (jy[i]=ria)) or (i=pocet);
  jed:=(jx[i]=stl) and (jy[i]=ria)
  end;

procedure potrava;      { testuje, či Wurmi nie je na políčku list }
  var i:integer;
  begin
  i:=0;
  repeat
    inc(i)
  until ((px[i]=stl) and (py[i]=ria)) or (i=pocet);
  if (px[i]=stl) and (py[i]=ria)
    then begin inc(body); px[i]:=0; py[i]:=0 end;
  gotoxy(3,2); write(body);
  end;

procedure pohyb;
  var stlznak:char;
  begin
  repeat
    if keypressed then stlznak:=readkey;
    gotoxy(stl,ria); write(' ');
    case stlznak of
      ESC:halt;
      sipkaVPRAVO:inc(stl);
      sipkaVLAVO :dec(stl);
      sipkaHORE  :dec(ria);
      sipkaDOLE  :inc(ria);
      end;
    gotoxy(stl,ria); write(chr(1)); potrava; delay(150);
  until (stl=1) or (stl=80) or (ria=1) or (ria=24) or jed or (body=pocet);
  if jed
    then begin gotoxy(33,12); write(' J E D ! ! ! ') end
    else if body=pocet
           then begin gotoxy(20,12); write('V Y B O R N E !!! VYHRAL SI!!!') end
           else begin gotoxy(25,12); write('B U M !!! OHRADA!!!') end
  end;

BEGIN
inicializacia;
pohyb;
readln
END.
```

Neriešené úlohy na jednorozmerné pole a typ string:

14. Vytvorte program na zistenie najmenšieho a najväčšieho prvku v poli najviac 100 celých čísel a výmenu najmenšieho prvku s prvým a najväčšieho prvku s posledným prvkom v poli.
15. Vytvorte program na nájdenie prvých dvoch najväčších (najmenších) čísel v poli najviac 20 rôznych celých čísel.
16. Vytvorte program na nájdenie k (k≤n) maxím (1. môžu byť aj rovnaké, 2. k rôznych maxím) v poli s n prvkami.

    Návod: využite upravený bubblesort.

17. Vytvorte program na zistenie počtu núl (kladných čísel, párnych čísel, čísel deliteľných zadaným číslom, čísel zo zadaného intervalu a pod.) v zadanom číselnom poli.
18. Vytvorte program na zistenie posledného výskytu (všetkých výskytov) zadaného znaku v texte.
19. Vytvorte program na zistenie polohy maxima a minima v danom poli (ošetrite aj viacnásobný výskyt).
20. Vytvorte program, ktorý prvky poľa napr. 2, 4, 5, 3, 8, 1, 3, 5, 9, 4, 7 zobrazí v tvare:

    2, 4, 5,
    3, 8, (3 < 5)
    1, 3, 5, 9, (1 < 8)
    4, 7 (4 < 9)
21. Vytvorte program, ktorý z pôvodného poľa odstráni všetky prvky so zadanou vlastnosťou (nepárne čísla, záporné čísla, rovné zadanej hodnote, znaky zo zadaného intervalu a pod.).
22. Vytvorte program, ktorý zlúči dve polia s prvkami rovnakého typu (prvky druhého poľa dá za prvky prvého poľa).
23. Vytvorte program, ktorý zlúči dve usporiadané polia s prvkami rovnakého typu tak, že výsledné pole bude tiež usporiadané.
24. Vytvorte program na výpočet a zobrazenie Pascalovho trojuholníka v tvare:

    1
    1 1
    1 2 1
    1 3 3 1
    1 4 6 4 1
    1 1+4 4+6 6+4 4+1 1
    ...

    Návod: Využite vzorce v zadaní REK-8, nepoužite však rekurziu ale pole.
25. Vytvorte program na uloženie prvkov pôvodného poľa opačne, vymeniť prvý prvok s posledným, druhý s predposledným atď.
26. Vytvorte program na posunutie prvkov poľa o k miest doprava (doľava). Posledných k prvkov poľa bude posunutých na začiatok poľa.
27. Vytvorte program na usporiadanie prvkov (čísel, znakov, reťazcov) daného poľa zostupne (od najväčšieho po najmenší).
28. Vytvorte program na výpočet ciferného súčtu prirodzeného čísla využitím typu string.
29. *Vytvorte program na zobrazenie všetkých prvočísiel od 2 po 60 000 metódou tzv. Eratosténovho sita. Návod: Napíšeme si čísla od 2 do 60 000. Dáme vypísať 2 a škrtneme všetky jej násobky po 60 000. Zoberieme a vypíšeme ďalšie číslo: 3 (nie je škrtnuté, preto je prvočíslo) a škrtneme všetky jeho násobky. Zoberieme ďalšie číslo: 4, je škrtnuté a preto nie je prvočíslo. Zoberieme ďalšie číslo: 5, nie je škrtnuté, preto ho vypíšeme ako prvočíslo a škrtneme všetky jeho násobky. Číslo 6 je škrtnuté. Číslo 7 vypíšeme (nie je škrtnuté) a škrtneme všetky jeho násobky (14, 21, 28,...). Takto pokračujeme až po 60 000. Môžeme použiť napríklad typ pole array [2..60000] of boolean, kde true znamená, že dané číslo nie je škrtnuté (je prvočíslo) a false znamená, že je škrtnuté.
30. Zefektívnite bubblesort tým, že ukončíte triedenie (prechody poľom), ak pri poslednom prechode už nedošlo ani k jednej výmene susedných prvkov (pole musí byť už utriedené).
31. *Vytvorte program, ktorý zistí, koľko anagramov sa nachádza v zadanom zozname slov.

Dvojrozmerné pole

V definícii typu pole:

type mt = *array* [ti] *of* tz

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 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 pole a tz je typ zložky.

Dovolený je aj skrátený zápis definície typu pole:

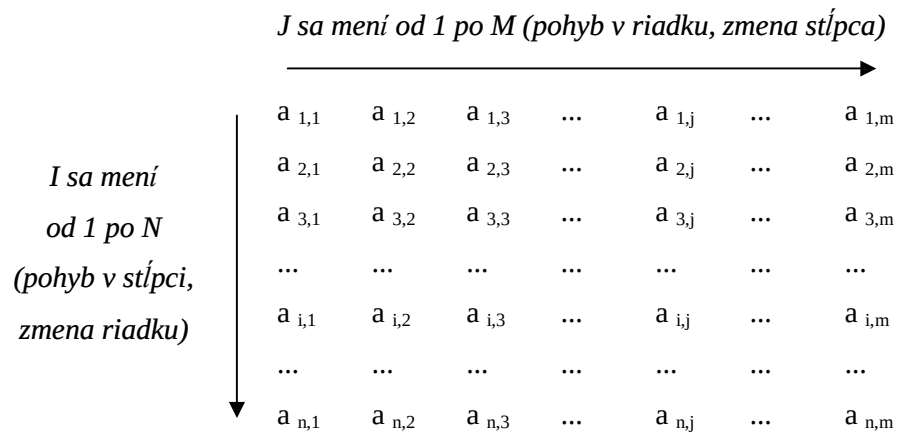
type mt = *array* [ti1 , ti2] *of* tz

Napríklad:

```
type    MATICA1 = array [1..10 ] of array [1..10] of integer;
        MATICA2 = array [1..10,1..10] of integer;       { definícia rovnocenná s MATICA1 }
        SACHOVNICA=array[´a´..´h´,1..8] of boolean;
        NEZMYSEL=array[boolean,boolean] of char;
```

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. Pre ľahšie pochopenie príkladov budeme ďalej používať nasledujúcu symboliku (častejšie ako dvojrozmerné pole budeme používať pojem matica, pretože tá predstavuje model na riešenie nášho problému; až keď je známy model na riešenie, môžeme ho realizovať v počítači dvojrozmerným poľom):

- matica typu N x M („en krát em“) obsahuje N riadkov a M stĺpcov
- pre zmenu riadku, t.j. pohyb v stĺpci budeme používať premennú I
- pre zmenu stĺpca, t.j. pohyb v riadku budeme používať premennú J
- schematicky (matica typu N x M):



Z hľadiska algoritmických konštrukcií pre prácu s maticou väčšinou potrebujeme cyklus v cykle. Vonkajší cyklus zabezpečuje najčastejšie nastavenie príslušného riadku (for I:=1 to N do) a vnútorný cyklus pohyb v nastavenom riadku t.j. zmenu stĺpca od 1 po M (for J:=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 MAT-1: Vytvorte program zabezpečujúci načítanie typu (rozmerov) a prvkov matice a jej zobrazenie na obrazovke monitora.

Analýza: Program obsahuje dva vstupy. V procedúre VSTUP treba zadať okrem rozmerov aj všetky prvky matice z klávesnice, v procedúre VSTUP_NAHODNE po zadaní rozmerov matice program náhodne vygeneruje prvky matice – celé čísla od 0 po 9. Keďže z procedúr vstupu sa majú novozískané hodnoty vyviesť, parametre musia byť nahradzované referenciou. Pri výstupe, hodnoty stačí len doviesť do procedúry, ide o náhradu formálnych parametrov hodnotou.

```
program MATICA_1;
uses  crt;
const MAXPP=10;
type  MATICA=array[1..MAXPP,1..MAXPP] of integer;
var   A:MATICA;
      N,M,I,J:integer;

procedure VSTUP(var N,M:integer;var A:MATICA);
  begin
  write('Zadaj rozmery matice <= ',MAXPP,': ');
  readln(N,M);
  for I:=1 to N do                { nastaví riadok }
    for J:=1 to M do              { nastaví stĺpec }
      begin
      write('Prvok ',I,',',J,': ');
      readln(A[I,J]);
      end;
  end;

procedure VSTUP_NAHODNE(var N,M:integer;var A:MATICA);
  begin
  write('Zadaj rozmery matice <= ',MAXPP,': ');
  readln(N,M);
  randomize;
  for I:=1 to N do
    for J:=1 to M do
      A[I,J]:=random(10);
  end;

procedure VYSTUP(N,M:integer;A:MATICA);
  begin
  for I:=1 to N do           { nastavuje riadok }
    begin
    for J:=1 to M do         { nastavuje stĺpec – pohyb v riadku }
      write(A[I,J]:4);       { hodnoty zobrazené v riadku }
    writeln;                 { presunutie „kurzora" na nový riadok }
    end;
  end;

BEGIN
clrscr;
writeln('Prvky matice nahodne (nie - N)?');
if upcase(readkey)='N'
  then VSTUP(N,M,A)
```

```
  else VSTUP_NAHODNE(N,M,A);
VYSTUP(N,M,A);
readln;
END.
```

Najprv si uvedieme niekoľko matematických príkladov na prácu s maticami.

Príklad MAT-2: Vytvorte program na vynásobenie matice celých čísel celým číslom k.

Analýza: Vynásobiť číselnú maticu číslom znamená vynásobiť každý jej prvok daným číslom, t.j. pre každé dovolené i, j: B[i,j] novej matice sa rovná k∗A[i,j].

```
procedure VYNASOB(N,M:integer;A:MATICA;var B:MATICA);
  var K:integer;
  begin
  write('Maticu vynasobit celym cislom: ');
  readln(K);
  for I:=1 to N do
    for J:=1 to M do
      B[I,J]:=K*A[I,J]
  end;

BEGIN
clrscr;
writeln('Prvky matice nahodne (nie - N)?');
if upcase(readkey)='N'
  then VSTUP(N,M,A)
  else VSTUP_NAHODNE(N,M,A);
VYSTUP(N,M,A);
VYNASOB(N,M,A,B);
VYSTUP(N,M,B);
readln;
END.
```

Príklad MAT-3: Vytvorte program na výpočet súčtu dvoch matíc rovnakých typov.

Analýza: Súčtom dvoch matíc A a B typu NxM je matica C typu NxM, kde C[i,j] = A[i,j] + B[i,j]. V procedúrach vstupu a výstupu je nevyhnutné použiť parametre, keďže procedúry chceme použiť pre matice A, B aj C!

```
program MATICA_3;
uses  crt;
const MAXPP=10;
type  MATICA=array[1..MAXPP,1..MAXPP] of integer;
var   A,B,C:MATICA;
      N,M,I,J:integer;

procedure ROZMERY(var N,M:integer);
  begin
  write('Zadaj rozmery matice <= ',MAXPP,': ');
  readln(N,M);
  end;

procedure VSTUP(N,M:integer;var A:MATICA);
...

procedure VSTUP_NAHODNE(N,M:integer;var A:MATICA);
...

procedure VYSTUP(N,M:integer;A:MATICA);
...

procedure SCITAT(N,M:integer;A,B:MATICA;var C:MATICA);
  begin
  for I:=1 to N do
    for J:=1 to M do
```

```
      C[I,J]:=A[I,J]+B[I,J]
  end;


BEGIN
randomize;
clrscr;
ROZMERY(N,M);
VSTUP_NAHODNE(N,M,A);
VSTUP_NAHODNE(N,M,B);
writeln('Prva matica: ');
VYSTUP(N,M,A);
writeln('Druha matica: ');
VYSTUP(N,M,B);
SCITAT(N,M,A,B,C);
writeln('Sucet prvej a druhej: ');
VYSTUP(N,M,C);
readln;
END.
```

Príklad MAT-4: Pre zasvätených, ktorí vedia, ako sa násobia matice, uvádzame bez komentára procedúru na súčin dvoch matíc. Matica A musí byť typu NxL, matice B typu LxM a výsledná matica C bude typu NxM.

```
procedure SUCIN(N,L,M:integer;A,B:MATICA;var C:MATICA);
  begin
  for I:=1 to N do
    for J:=1 to M do
       begin
       C[I,J]:=0;
       for K:=1 to L do
         C[I,J]:=C[I,J] + A[I,K] * B[K,J]
       end;
  end;
```

Príklad MAT-5: Vytvorte program, ktorý zistí, či zadaná štvorcová matica je symetrická.

Analýza: Štvorcová matica (počet riadkov sa rovná počtu stĺpcov) je symetrická, ak je symetrická podľa hlavnej diagonály (hlavná diagonála je tvorená prvkami A[1,1], A[2,2], A[3,3], ..., A[N,N]). Preto pre každý prvok matice musí platiť: A[I,J] = A[J,I] pre všetky dovolené hodnoty I a J.

Nie najefektívnejší, ale jednoduchší program znamená použitie cyklov for, ktoré "zistia", či sa prvky pod hlavnou diagonálou rovnajú príslušným prvkom nad hlavnou diagonálou (ak sa „dolný trojuholník rovná hornému, musí sa aj horný rovnať dolnému"). Keďže výsledkom podprogramu je hodnota true – ak je matica symetrická alebo hodnota false – ak nie je symetrická, môžeme použiť aj funkciu.

```
program MATICA_5;
uses  crt;
const MAXPP=10;
type  MATICA=array[1..MAXPP,1..MAXPP] of integer;
var   A:MATICA;
      N,I,J:integer;
procedure VSTUP(var N:integer;var A:MATICA);
....
procedure VYSTUP(N:integer;A:MATICA);
...
function SYMETRICKA(N:integer;A:MATICA):boolean;
  var SYM:boolean;
  begin
  SYM:=true;
  for I:=2 to N do
```

```
    for J:=1 to I-1 do
      if A[I,J]<>A[J,I] then SYM:=false; { SYM:=SYM and (A[I,J]=A[J,I]); }
  SYMETRICKA:=SYM
  end;
BEGIN
clrscr;
VSTUP(N,A);
writeln('Matica: ');
VYSTUP(N,A);
if SYMETRICKA(N,A)
  then writeln('je symetricka.')
  else writeln('nie je symetricka.');
readln;
END.
```

Príklad MAT-6: Vytvorte program na nájdenie najmenšej a najväčšej hodnoty v každom riadku matice.

Analýza: Každý riadok matice je jednorozmerné pole „doplnené“ o index riadku, v ktorom sa nachádza. Ak vieme hľadať minimum a maximum v jednorozmernom poli, stačí toto hľadanie vložiť do cyklu, ktorý bude meniť riadky od prvého po posledný.

```
program MATICA_6;
uses  crt;
const MAXPP=10;
type  MATICA=array[1..MAXPP,1..MAXPP] of integer;
      POLE=array[1..MAXPP] of integer;
var   A:MATICA;
      MIN,MAX:POLE;
      N,M,I,J:integer;

procedure VSTUP(var N,M:integer;var A:MATICA);
...
procedure VSTUP_NAHODNE(var N,M:integer;var A:MATICA);
...
procedure VYSTUP(N,M:integer;A:MATICA);
...

procedure MIN_MAX(N,M:integer;A:MATICA;var MIN,MAX:POLE);
  begin
  for I:=1 to N do
    begin
    MIN[I]:=A[I,1];
    MAX[I]:=A[I,1];
    for J:=2 to M do
      begin
      if A[I,J]<MIN[I] then MIN[I]:=A[I,J];
      if A[I,J]>MAX[I] then MAX[I]:=A[I,J]
      end;
    end;
  end;

procedure VYSTUPPOLA(N:integer;A:POLE);
  begin
  for I:=1 to N-1 do
    write(A[I],', ');
  writeln(A[N])
  end;

BEGIN
clrscr;
randomize;
writeln('Prvky matice nahodne (nie - N)?');
if upcase(readkey)='N'
  then VSTUP(N,M,A)
```

```
  else VSTUP_NAHODNE(N,M,A);
MIN_MAX(N,M,A,MIN,MAX);
writeln('V matici: ');
VYSTUP(N,M,A);
writeln('minima v riadkoch: ');
VYSTUPPOLA(N,MIN);
writeln('maxima v riadkoch: ');
VYSTUPPOLA(N,MAX);
readln;
END.
```

Príklad MAT-7: Vytvorte program na utriedenie prvkov v každom riadku matice.

Analýza: Aby sme zvládli tento príklad, musíme vedieť aspoň jeden triediaci algoritmus. Pri jednorozmernom poli sme sa zaoberali bubblesortom, ktorý použijeme aj teraz. Ako v predchádzajúcom príklade, aj tu treba len bubblesort „aplikovať“ na každý riadok matice, čo znamená vložiť bubblesort do cyklu na nastavenie riadku.

```
procedure UTRIED_V_RIADKU(N,M:integer;var A:MATICA);
  var PP:integer;
  procedure VYMENA(var X,Y:integer);
    var POM:integer;
    begin
    POM:=X; X:=Y; Y:=POM;
    end;
  begin
  for I:=1 to N do                  { nastaví riadok }
    for PP:=1 to M-1 do             { počet prechodov riadkom }
      for J:=1 to M-PP do           { pohyb v riadku }
        if A[I,J]>a[I,J+1] then VYMENA(A[I,J],A[I,J+1]);
  end;

BEGIN
clrscr;
writeln('Prvky matice nahodne (nie - N)?');
if upcase(readkey)='N'
  then VSTUP(N,M,A)
  else VSTUP_NAHODNE(N,M,A);
VYSTUP(N,M,A);
UTRIED_V_RIADKU(N,M,A);
writeln('Matica utriedenia v riadkoch: ');
VYSTUP(N,M,A);
readln;
END.
```

Ako treba upraviť program, aby sa zachovala aj pôvodná matica A? Podprogram, ktorý utriedi prvky matice v stĺpcoch, sa len málo líši od podprogramu UTRIED_V_RIADKU. Odlaďte ho.

Príklad MAT-8: Vytvorte program, ktorý utriedi riadky matice podľa hodnoty prvého prvku v riadku, t.j. podľa prvého stĺpca.

Analýza:

Napríklad v matici:	2	4	1	5
	3	5	2	7
	1	2	1	4

bude posledný riadok presunutý do prvého, prvý do stredného a stredný riadok do tretieho.

Utriedená matica:	1	2	1	4
	2	4	1	5

3 5 2 7

Keďže je potrebné vymieňať celé riadky, už v úseku definícií typ dvojrozmerné pole zadefinujeme ako pole riadkov. Tiež si treba uvedomiť, že všetky prvky v prvom stĺpci sú prístupné cez zápis A[I,1], kde I určuje riadok. No a dokonale treba ovládať nejaký triediaci algoritmus, napríklad bubblesort.

```
program MATICA_8;
uses  crt;
const MAXPP=10;
type  RIADOK=array[1..MAXPP] of integer;
      MATICA=array[1..MAXPP] of RIADOK;
var   A:MATICA;
      N,M,I,J:integer;

procedure VSTUP_NAHODNE(var N,M:integer;var A:MATICA);
...
procedure VYSTUP(N,M:integer;A:MATICA);
...
procedure UTRIED_PODLA_PRVEHO(N,M:integer;var A:MATICA);
  var R:RIADOK;
      PP:integer;
  procedure VYMENA(var X,Y:RIADOK);       { typ RIADOK! }
    var POM:RIADOK;
    begin
    POM:=X; X:=Y; Y:=POM
    end;
  begin
  for PP:=1 to N-1 do
    for I:=1 to N-PP do
      if A[I,1]>A[I+1,1] then VYMENA(A[I],A[I+1])
  end;

BEGIN
clrscr;
VSTUP_NAHODNE(N,M,A);
writeln('Povodna matica: ');
VYSTUP(N,M,A);
UTRIED_PODLA_PRVEHO(N,M,A);
writeln('Riadky utriedene podla prveho prvku v riadku: ');
VYSTUP(N,M,A);
readln;
END.
```

Na záver „matematických“ príkladov s maticami uvedieme „hviezdičkový“ príklad.

Príklad MAT-9*: Vytvorte program, ktorý „rozvinie“ riadky a stĺpce matice do jednorozmerného poľa v smere chodu hodinových ručičiek.

Analýza:

Napríklad maticu: 2 4 1 5

3 5 2 7

1 2 1 4

rozvinie program do poľa: 2 4 1 5 7 4 1 2 1 3 5 2

```
program MAT-9;
uses  crt;
const MAXPP=10;
type  MATICA=array[0..MAXPP,0..MAXPP] of integer;
      POLE=array[0..MAXPP*MAXPP] of integer;
var   A:MATICA;
      B:POLE;
      i,j,k,n,m,l,p,h,d:integer;
```

```
procedure VSTUP;
...
procedure VYSTUP;
...
procedure VYTVOR_POLE;
 procedure vpravo;
   begin
   for j:=l to p do begin k:=k+1; B[k]:=A[h,j]; end;
   end;
 procedure dole;
   begin
   for i:=h to d do begin k:=k+1; B[k]:=A[i,p]; end;
   end;
 procedure vlavo;
   begin
   for j:=p downto l do begin k:=k+1; B[k]:=A[d,j]; end;
   end;
 procedure hore;
   begin
   for i:=d downto h do begin k:=k+1; B[k]:=A[i,l]; end;
   end;
 begin
 k:=0; l:=1; p:=m; h:=1; d:=n;
 repeat
   if k<n*m then vpravo; h:=h+1;
   if k<n*m then dole;   p:=p-1;
   if k<n*m then vlavo;  d:=d-1;
   if k<n*m then hore;   l:=l+1;
 until k=n*m;
 end;
procedure VYSTUP_POLA;
 begin
 writeln('"Rozvinuta" matica:');
 for k:=1 to n*m do write(B[k]:4);
 writeln;
 end;
BEGIN
clrscr;
VSTUP;
VYSTUP;
VYTVOR_POLE;
VYSTUP_POLA;
readln
END.
```

Mnohí z nás poznajú hru ŽIVOT, v ktorej sa simuluje život jednobunkových organizmov v rovine. Po vložení počiatočnej generácie program „počíta“ ďalšie generácie. Organizmus prežije do ďalšej generácie, ak ma 2 alebo 3 susedov, ináč zomrie. Živý organizmus vznikne, ak má práve 3 susedov.

Príklad MAT-10: Vytvorte program simulujúci život jednobunkových organizmov za vyššie uvedených podmienok.

Analýza: Potrebné sú dve matice. V jednej „prebieha život“, druhá je pomocná pri výpočte novej generácie. Pre získanie nových poznatkov sme namiesto čísel 0 – prázdna pozícia a 1 – živý organizmus použili programátorom definovaný typ s hodnotami nezivý (ordinálne číslo 0) a zivy (ordinálne číslo 1). Program ponechávame na samoštúdium.

```
program MAT-10;
uses  crt;
const okraj=24;
      plot =23;
```

```
type  torganizmu=(nezivy,zivy);     { programátorom vymenovaný typ }
      tsuradnic=0..okraj;
      tpola=array[tsuradnic,tsuradnic] of torganizmu;
var   a,pom:tpola;
      q:char;

procedure vycisti_pole;
  var i,j:tsuradnic;
  begin
  clrscr;
  for i:=0 to okraj do
    for j:=0 to okraj do
      begin
      a[i,j]:=nezivy;
      pom[i,j]:=nezivy;
      end;
  end;
procedure vystup;
  var i,j:tsuradnic;
  begin
  clrscr;
  for i:=1 to plot do
    begin
    for j:=1 to plot do
      if a[i,j]=zivy
        then write('*')
        else write(' ');
    writeln
    end;
  end;

procedure vstup;
  var i,j:integer;
  begin
  repeat
    write('Zadaj suradnice x,y ziveho organizmu ( KONIEC -> 0 ) ');
    read(i);
    if i>0
      then begin
           readln(j);
           if (i<okraj)and(j>0)and(j<okraj)
             then a[i,j]:=zivy
             else begin writeln('Chybne suradnice');delay(2000) end;
           vystup
           end
  until i=0;
  end;

procedure nova_generacia;
  var i,j:tsuradnic; pocet:0..8;
  begin
  for i:=1 to plot do
    for j:=1 to plot do
      begin
      pocet:=ord(a[i-1,j-1])+      { počíta počet susedov okolo a[i,j] }
             ord(a[i-1,j])+
             ord(a[i-1,j+1])+
             ord(a[i,j-1])+
             ord(a[i,j+1])+
             ord(a[i+1,j-1])+
             ord(a[i+1,j])+
             ord(a[i+1,j+1]);
      if (pocet=3)or((pocet=2)and(a[i,j]=zivy))
        then pom[i,j]:=zivy
        else pom[i,j]:=nezivy;
      end;
```

```
  a:=pom;
  vystup;
  end;
BEGIN
vycisti_pole;
repeat
  vstup;
  repeat
    nova_generacia;
    write('Chces dalsiu generaciu (N) ? ');q:=upcase(readkey);
  until q='N';
  gotoxy(1,24);
  write('Chces doplnit zadanie (A) ? ');q:=upcase(readkey);
until q<>'A';
END.
```

Neriešené úlohy na dvojrozmerné pole:

11. Vytvorte program na výpočet súčtu všetkých prvkov číselnej matice.
12. Vytvorte program, ktorý sčíta všetky prvky na hlavnej diagonále (ide z ľavého horného rohu matice do pravého dolného rohu) štvorcovej matice.
13. Vytvorte program, ktorý sčíta všetky prvky na vedľajšej diagonále (ide z pravého horného rohu matice do ľavého dolného rohu) štvorcovej matice.
14. Vytvorte program na zistenie počtu núl (zadaného čísla, kladných čísel, deliteľných zadaným číslom, zo zadaného intervalu a pod.) v zadanej číselnej matici.
15. Vytvorte program na nájdenie najmenšieho (najväčšieho) prvku matice a jeho výmenu s prvým (posledným) prvkom matice.
16. Vytvorte program na nájdenie najmenšieho a najväčšieho prvku každého riadku matice a jeho výmenu s prvým a posledným prvkom v danom riadku matice.
17. Vytvorte program na nájdenie najmenšieho a najväčšieho prvku v každom stĺpci matice a jeho výmenu s prvým a posledným prvkom v danom stĺpci.
18. Vytvorte program na nájdenie zadanej hodnoty v matici a jej nahradenie novou hodnotou.
19. Vytvorte program na zistenie, či sa zadaný znak nachádza v matici znakov a koľkokrát sa nachádza.
20. Vytvorte program na zistenie miesta výskytu (všetkých, prvého, posledného) prvku so zadanou vlastnosťou v matici.
21. Vytvorte program, ktorý vo štvorcovej matici vymení prvky podľa hlavnej diagonály.
22. Vytvorte program, ktorý v matici vymení prvý stĺpec (riadok) s posledným, druhý s predposledným, atď.
23. Vytvorte program, ktorý zistí, či zadaná štvorcová matica je diagonálna (okrem hlavnej diagonály a[i,j] = 0).
24. Vytvorte program na otočenie štvorcovej matice o 90^0.
25. Vytvorte program, ktorý od miesta „čľupnutia kameňa do matice“ zvýši pôvodne nulové hodnoty matice s narastajúcou vzdialenosťou od miesta „čľupnutia“ o 1.
26. Vytvorte program, ktorý pri pohybe v matici bude meniť hodnoty prechádzaných prvkov.
27. Vytvorte program, ktorý pri pohybe v matici bude meniť hodnoty v okolí prechádzaných prvkov.

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 KARTA = 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;
               Moze : boolean
             end;

type BOD = record
             x,
             y : real;
             farba : 0..14;
             blik : boolean
           end;

type ZLOZKA = record
                HODNOTA : string;
                POCET   : byte
              end;

var KC : record Re, Im: real end;        { deklarovaná premenná typu record }
```

Ak p je premenná typu záznam, k jej položke s názvom pi sa dostaneme zápisom: p.pi. Napríklad, ak K je premenná typu KARTA, možno použiť zápisy: K.OsCislo, K.Meno, K.DatNar.Den, K.DatNar.Rok, K.Adresa.Obec, K.Priemer[1] a pod.

Príklad ZAZ-1: Vytvorte program na určenie vzájomnej polohy bodu a kružnice v rovine s využitím údajového typu záznam.

Analýza: Bod napr. B je určený v rovine dvoma súradnicami B_x a B_y. Kružnica je určená stredom S so súradnicami S_x a S_y a polomerom r. Vzdialenosť medzi bodom B a stredom kružnice k, bodom S, možno

vypočítať pomocou Pythagorovej vety. Bod B leží vo vnútornej oblasti kružnice k (S;r), ak jeho vzdialenosť od stredu kružnice k je menšia ako polomer r. Ak vzdialenosť /BS/ = r, bod B leží na kružnici a ak je táto vzdialenosť väčšia ako polomer r, bod leží mimo oblasť kružnice k.

```
program ZAZ_1;
uses crt;
type BOD=record
           x,y:real;
         end;
     KRUZNICA=record
                s:BOD;
                r:real;
              end;
var A:BOD;
    k:KRUZNICA;
    vzd :real;

procedure VSTUP;
  begin
  writeln('Zadaj suradnice bodu ');
  readln(A.x,A.y);
  writeln('Zadaj stred kruznice a polomer');
  readln(k.s.x,k.s.y,k.r);
  end;

function VZDIALENOST:real;
  begin
  vzdialenost:=sqrt(sqr(A.x-k.s.x)+sqr(A.y-k.s.y));
  end;

BEGIN
clrscr;
VSTUP;
vzd:=VZDIALENOST;
if vzd>k.r
  then writeln('Bod lezi vo vonkajsej oblasti kruznice.')
  else if vzd=k.r
         then writeln('Bod lezi na kruznici.')
         else writeln('Bod lezi vo vnutornej oblasti kruznice.');
readln;
END.
```

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.položka skrátiť na zápis položka.

Napríklad: with k do begin writeln('Zadaj stred kruznice a polomer'); readln(s.x , s.y , r) end;

with Z , DatNar do begin OsCislo:=99; Den:=1; Mes:=1; Rok:=2000 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 ZAZ-2: Vytvorte jednoduchú kartotéku na evidenciu osôb (evidenčné číslo, meno a adresa: miesto, ulica).

Analýza: Na precvičenie údajového typu record vytvoríme jednoduchú kartotéku. Jednotlivé karty budú uložené v poli, čo v skutočnosti nikdy nebýva, lebo ukončením programu by sme prišli o všetky údaje v kartách. Na zapamätanie údajov aj po vypnutí počítača slúžia vonkajšie pamäte (najmä pevný disk). Ukladať údaje na vonkajšie pamäte umožňuje údajový typ súbor, o ktorom si povieme už v nasledujúcej kapitole.

```
program ZAZ_2;
uses  crt;
const pk=10;        { pocet kariet }
      dmena=15;
      dadr=25;
type  karta=record
           ec:byte;
           meno:string[dmena];
           adr:record
               miesto,
               ulica:string[dadr];
               end;
            end;
      kartoteka=array[1..pk] of karta;
var   k:kartoteka;
      apk,i:byte;
      q:char;

procedure vstup_karta;
  begin
  clrscr;
  apk:=apk+1;
  with k[apk],adr do
    begin
    ec:=apk;
    gotoxy(1,8);
    writeln('Evidencne cislo : ':40,ec);
    writeln;
    write('Meno a priezvisko : ':40);readln(meno);
    writeln;
    write('  Adresa - miesto : ':40);readln(miesto);
    writeln;
    write('         -  ulica : ':40);readln(ulica);
    end;
  end;

procedure vystup_kartoteka;
  procedure vystup_karta(i:byte);
    begin
    clrscr;
    writeln('Vystup');
    gotoxy(1,8);
    with k[i],adr do
      begin
      writeln('  Evidencne cislo : ':40,ec);
      writeln;
      writeln('Meno a priezvisko : ':40,meno);
      writeln;
      writeln('  Adresa - miesto : ':40,miesto);
      writeln;
      writeln('          - ulica : ':40,ulica);
      end;
    readln;
    end;
```

```
  begin     {procedura vystup_kartoteka }
  for i:=1 to apk do vystup_karta(i);
  end;

procedure menu;
  begin
  repeat
    clrscr;
    gotoxy(1,10);
    writeln('VLOZIT KARTU..................... +':55);
    writeln;
    writeln('VYPISAT KARTOTEKU.............. Enter':55);
    writeln;
    writeln('KONIEC........................... K':55);
    q:=upcase(readkey);
    case q of
      '+'     : vstup_karta;
      chr(13) : vystup_kartoteka;
    end;
  until q='K';
  end;

BEGIN
apk:=0;
vstup_karta;
menu;
END.
```

Variantný záznam

Pri používaní údajového typu záznam môže vyvstať požiadavka na návrh takej štruktúry záznamu, v ktorej sa na daný prvok nemusí vzťahovať celý zoznam položiek uvedených v zázname, ale iba jeho určitá časť. Takto môže vzniknúť niekoľko alternatívnych typov – variant - jedného typu záznam. Takúto situáciu nám umožňuje riešiť tzv. variantný záznam, ktorý sa skladá z pevnej časti a z variantnej časti.

Variantný záznam má tvar:

```
record
    pevná časť;
    case rozlišovacia_položka : rozlišovací_typ_variantnej_časti of
       rozlišovacia_konštanta1 : ( položka1 : typ1;
                                   položka2 : typ2;
                                   ...          );
       rozlišovacia_konštanta2 : (položka1 : typ1;
                                   položka2 : typ2;
                                   ...          );
       ...
end
```

kde rozlišovací typ variantnej časti môže byť len ordinálny typ.

Napríklad:

```
type    TypPohlavia =(MUZ,ZENA);
        TypMiery=(PRSIA,PAS,BOKY);
```

```
OSOBA = record
            Priezvisko,
            Meno        : string[15];
            DatNar      : string[10];
            case Pohlavie : TypPohlavia of
               MUZ : (Hmotnost : real;
                      Brada    : boolean);
               ZENA : (Miery : array[TypMiery] of  byte)
          end;
```

Príklad ZAZ-3: Program ZAZ-2 doplňte o variantný záznam evidujúci u mužov fúzy a u žien počet detí.

Analýza: Aj keď je program veľmi podobný predchádzajúcemu, uvádzame ho podrobne až po celkom spoločnú časť.

```
program ZAZ_3;
uses  crt;
const pk=10;            { pocet kariet }
      dmena=15;
      dadr=25;
type  karta=record
           ec:byte;
           meno:string[dmena];
           adr:record
               miesto,
               ulica:string[dadr];
               end;
           case pohl:(muz,zena) of
             muz:(fuzy:boolean);
             zena:(deti:integer);
           end;
     kartoteka=array[1..pk] of karta;
var  k:kartoteka;
     apk,i:byte;
     q:char;

procedure vstup_karta;
  var odp:char;
  begin
  clrscr;
  apk:=apk+1;
  with k[apk],adr do
    begin
    ec:=apk;
    gotoxy(1,8);
    writeln('Evidencne cislo : ':40,ec);
    writeln;
    write('Meno a priezvisko : ':40);readln(meno);
    writeln;
    write('  Adresa - miesto : ':40);readln(miesto);
    writeln;
    write('         -  ulica : ':40);readln(ulica);
    writeln;
    write('  Muz (ano - A) ? : ':40);
    if upcase(readkey)='A'
      then begin
           writeln('ano');
           pohl:=muz;
           writeln;
           write('Ma fuzy (ano - A) ? : ':40);
           fuzy:=upcase(readkey)='A';
```

```
           if fuzy then writeln('ano') else writeln('nie')
           end
      else begin
           writeln('nie');
           pohl:=zena;
           writeln;
           write('Pocet deti : ':40);readln(deti);
           end;
    delay(500)
    end;
  end;

procedure vystup_kartoteka;
  procedure vystup_karta(i:byte);
    begin
    clrscr;
    writeln('Vystup');
    gotoxy(1,8);
    with k[i],adr do
      begin
      writeln('  Evidencne cislo : ':40,ec);
      writeln;
      writeln('Meno a priezvisko : ':40,meno);
      writeln;
      writeln('  Adresa - miesto : ':40,miesto);
      writeln;
      writeln('           - ulica : ':40,ulica);
      writeln;
      if pohl=muz
        then if fuzy
               then writeln('Fuzy : ano':43)
               else writeln('Fuzy : nie':43)
        else writeln('Pocet deti : ':40,deti);
      end;
    readln;
    end;
  begin
  for i:=1 to apk do vystup_karta(i);
  end;
...
```

Na záver údajového typu záznam uvádzame „červíka Wurmiho“ (Príklad POLE-13) doplneného o jeho rast, pridanie článku tela, pri každom zožratí potravy (listu).

Príklad ZAZ-4: Program POLE-13 doplňte o „telo“ červíka Wurmiho, ktoré narastie o jeden článok pri každom zožratí potravy.

Analýza: Program POLE-13 treba doplniť o proces zapamätania si polohy článkov tela Wurmiho. Z viacerých možností sme vybrali na zapamätanie polohy článkov tela jednorozmerné pole TELO, v ktorom sú ako položky recordu zapamätané x-ová a y-nová súradnica 1., 2. atď. článku tela.

Program ponechávame na samoštúdium, podprogramy totožné s programom POLE-13 neuvádzame.

```
program ZAZ_4;
uses  crt,dos;
const pocet=5;
...
var   body,dlzka:integer;
      px,py,jx,jy:array[1..pocet] of integer;
      telo:array[0..pocet] of record x,y:integer end;

procedure skry_kurzor;
...
```

```
procedure ohrada;
...
procedure generuj_potravu_jed;
...
procedure vykresli_potravu_jed;
...
procedure inicializacia;
  var limit:integer;
  begin
  randomize;
  generuj_potravu_jed;
  skry_kurzor;
  clrscr;
  ohrada;
  vykresli_potravu_jed;
  dlzka:=0; telo[0].x:=40; telo[0].y:=12;
  gotoxy(telo[0].x,telo[0].y); write(tvaricka);
  body:=0;
  end;

function jed:boolean;
  var i:integer;
  begin
  i:=0;
  repeat
    inc(i)
  until ((jx[i]=telo[0].x) and (jy[i]=telo[0].y)) or (i=pocet);
  jed:=(jx[i]=telo[0].x) and (jy[i]=telo[0].y)
  end;

function potrava:boolean;
  var i:integer;
  begin
  i:=0;
  repeat
    inc(i)
  until ((px[i]=telo[0].x) and (py[i]=telo[0].y)) or (i=pocet);
  if (px[i]=telo[0].x) and (py[i]=telo[0].y)
    then begin potrava:=true; inc(body); px[i]:=0; py[i]:=0 end
    else potrava:=false;
  gotoxy(3,2); write(body);
  end;

procedure pohyb;
  var stlznak:char;
      i:integer;
  begin
  repeat
    if keypressed then stlznak:=readkey;
    if potrava then inc(dlzka);
    gotoxy(telo[dlzka].x,telo[dlzka].y); write(' ');
    for i:=dlzka downto 1 do
      begin
      telo[i]:=telo[i-1];
      gotoxy(telo[i].x,telo[i].y); write('0');
      end;
    case stlznak of
      ESC:halt;
      sipkaVPRAVO:inc(telo[0].x);
      sipkaVLAVO :dec(telo[0].x);
      sipkaHORE  :dec(telo[0].y);
      sipkaDOLE  :inc(telo[0].y);
      end;
    gotoxy(telo[0].x,telo[0].y); write(tvaricka);
    delay(100);
  until  (telo[0].x=1)or(telo[0].x=80)or(telo[0].y=1)or(telo[0].y=24)or  jed  or
```

```
(body=pocet);
  if jed
    then begin gotoxy(33,12); write(' J E D ! ! ! ') end
    else if body=pocet
           then begin gotoxy(20,12); write('V Y B O R N E !!! VYHRAL SI!!!') end
           else begin gotoxy(25,12); write('B U M ! ! !  O H R A D A ! ! !') end
  end;

BEGIN
inicializacia;
pohyb;
readln
END.
```

Neriešené úlohy na záznam:

5. Navrhnite úsek definícií a deklarácií pre evidenčnú kartu knihy a čitateľa v knižnici.
6. Vytvorte program na evidenciu kníh (čitateľov) knižnice.
7. Program z úlohy 6 doplňte o vyradenie čitateľa (knihy) z evidencie.
8. Program z úlohy 6 doplňte o vyhľadanie knihy (čitateľa) po zadaní mena.
9. Vytvorte program na evidenciu motorových vozidiel.
10. Program z úlohy 9 doplňte o variantný záznam, ktorý podľa typu vozidla bude evidovať napr. pri osobnom aute počet miest a nosnosť, pri autobuse počet miest, klimatizáciu a pod., pri nákladnom aute nosnosť, nákladnú plošinu a pod.
11. Vytvorte program pracujúci ako elektronický dvojjazyčný (viacjazyčný) slovník.
12. Vytvorte program, ktorý po zadaní textu zistí počet výskytov jednotlivých slov v texte.
13. Pre typ RETAZEC = record

    POCETZN : 0..255;
    ZNAKY : array[1..255] of char

    end;

    napíšte procedúru, ktorá spojí dva reťazce do tretieho.
14. K údajovému typu z príkladu 13 napíšte funkciu, ktorá zistí, či jeden reťazec je obsiahnutý v druhom (je podreťazcom).

Údajový typ súbor

Ak potrebujeme uchovať údaje aj po vypnutí počítača resp. po ukončení programu, musíme ich uložiť na vonkajšie pamäťové médium – disk, disketu a pod. Zapisovať a čítať údaje z vonkajších pamäťových médií nám v TP umožňuje údajový typ súbor.

Súbor je štruktúrovaný údajový typ, ktorý sa skladá z teoreticky neobmedzeného počtu zložiek, všetky rovnakého typu. Prakticky je počet zložiek súboru obmedzený kapacitou vonkajšej pamäte.

Definícia typu súbor má tvar: *type* mt = *file of* tz

kde mt je meno typu a tz je typ zložky, ktorý nesmie byť typ súbor ani typ obsahujúci ako zložku typ súbor.

Napríklad:	type CELE = file of integer;	typ súbor obsahujúci celé čísla
	type KARTOTEKA = file of KARTA;	typ umožňujúci uloženie kartotéky do súboru
	type BODY = file of record x,y,z:real end;	typ súbor obsahujúci trojice x,y,z
	var F : file of string;	premenná typu súbor obsahujúci reťazce

Súbory môžu byť vnútorné (pracovné) a vonkajšie. Pracovné súbory si môže vytvoriť program pri riešení pamäťovo náročnejších úloh, TP im dáva príponu $$$. Po skončení programu sa môžu z vonkajšej pamäte odstrániť. Ďalej sa budeme zaoberať len vonkajšími súbormi, ktoré existujú aj po skončení programu a umožňujú trvalé uloženie dát.

Základné príkazy pre prácu so súbormi umožňujú len sekvenčné spracovanie súboru, t.j. vytváranie aj čítanie súboru len zložku po zložke (od prvej k druhej atď.). V každom okamihu spracovania súboru je prístupná len jedna zložka súboru, na ktorú „ukazuje“ prístupová premenná súboru. Ak F je premenná typu súbor, k nej prislúchajúca prístupová premenná má označenie F^ a vznikne automaticky pri deklarácii premennej typu súbor.

Deklaráciou premennej typu súbor vznikne len abstraktný vonkajší súbor, sú určené len jeho logické vlastnosti. K spojeniu logického súboru s fyzickým – skutočne existujúcim na konkrétnom vonkajšom pamäťovom médiu, dôjde príkazom *assign* (logické_meno , fyzické_meno) kde logické meno je premenná typu súbor a fyzické meno je reťazec obsahujúci názov súboru na disku, prípadne doplnený aj o cestu. Napríklad: assign(F,´CITATEL.DAT´), assign(SUB,´C:\TP\CELE1.DBF´).

Zápis údajov do súboru (napríklad F):

Zápis údajov do súboru (po stotožnení logického a fyzického mena súboru!) prebieha v dvoch fázach:

1. súbor sa pripraví na zápis príkazom *rewrite*(F). Vytvorí sa prázdny súbor (prázdna hodnota súboru), prístupová premenná F^ sa nastaví na „koniec“ súboru. Ak súbor už obsahuje nejaké zložky, budú odstránené!
2. vlastný zápis do súboru sa realizuje príkazom *write*(F,V), kde V je výraz rovnakého typu, ako je typ zložky súboru. Po vyhodnotení výrazu V, získaní konkrétnej hodnoty, dôjde k jej zápisu do súboru F. Bod 2 možno ľubovoľný počet krát opakovať.

Čítanie údajov zo súboru (napríklad F):

Aj sekvenčné čítanie údajov zo súboru prebieha v dvoch fázach:

1. súbor sa pripraví na čítanie príkazom *reset*(F). Príkaz spôsobí nastavenie prístupovej premennej F^ na začiatok súboru.
2. vlastné čítanie zo súboru sa realizuje príkazom *read*(F,X), kde X je premenná rovnakého typu ako je typ zložky súboru. Po vykonaní príkazu je v premennej X hodnota načítaná zo súboru. Čítanie zo súboru možno opakovať, až kým prístupová premenná nie je za poslednou položkou súboru.

Na zistenie pozície prístupovej premennej v súbore slúži funkcia eof (end of file), ktorá má hodnotu true, ak prístupová premenná je na konci súboru (hodnota F^ nie je definovaná), inak má hodnotu false. To možno výhodne využiť na načítanie všetkých zložiek súboru schémou:

```
reset(F);
while not eof(F) do      { pokiaľ nie je koniec súboru opakuj }
   begin
   read(F,X);            { načíta hodnotu zo súboru do premennej X }
   write(X)              { zobrazí načítanú hodnotu na obrazovke }
   end;
```

Ukončenie práce so súborom (napríklad F):

Po ukončení práce so súborom sa súbor musí zavrieť príkazom *close*(F).

Príklad SUB-1: Vytvorte program na vytvorenie súboru a zobrazenie zložiek súboru obsahujúceho čísla typu integer.

Analýza: Program obsahuje dve samostatné procedúry. Procedúra VSTUP umožňuje vytvoriť, po zadaní mena fyzického súboru, súbor obsahujúci celé čísla a procedúra VYSTUP zobrazuje zložky – celé čísla, zvoleného súboru. Program je pre pohodlie doplnený o menu.

```
program SUB_1;
uses  crt;
var   F:file of integer;
      ODP:char;

procedure VSTUP;
  var FM:string[12];
      X:integer;
  begin
  write('Meno suboru na disku: ');
  readln(FM);
  assign(F,FM);
  rewrite(F);
  repeat
    write('Vlozit cislo: ');
    readln(X);
    write(F,X);
    writeln('Vlozit dalsie cislo (nie - N) ?':52);
  until upcase(readkey)='N';
  close(F);
  end;

procedure VYSTUP;
  var FM:string[12];
      X:integer;
  begin
  write('Meno suboru na disku: ');
  readln(FM);
  assign(F,FM);
  reset(f);
  while not eof(F) do
```

```
    begin
    read(F,X);
    write(X:4);
    end;
  close(F);
  readln
  end;

BEGIN
repeat
  clrscr;
  gotoxy(1,10);
  writeln('Vytvorit subor ............... V':50);
  writeln;
  writeln('Zobrazit subor ............... Z':50);
  writeln;
  writeln('Koniec ....................... K':50);
  repeat ODP:=upcase(readkey); until (ODP='V')or(ODP='Z')or(ODP='K');
  clrscr;
  case ODP of
    'V' : VSTUP;
    'Z' : VYSTUP;
  end
until ODP='K'
END.
```

TP má pre prácu so súbormi ďalšie procedúry a funkcie. Funkcia *FileSize*(F) vracia číslo udávajúce počet zložiek súboru F. Procedúra *Seek*(F,n) nastaví prístupovú premennú za n-tú zložku súboru F. Preto príkaz Seek(F,FileSize(F)) nastaví prístupovú premennú F^ za poslednú zložku súboru a umožní nám napríklad zápis na koniec súboru F. Príkaz Rename(F,nové_fyz._meno) premenuje externý súbor (po príkaze assign). Príkaz Erase(F) zmaže externý súbor (po príkaze assign).
Uvedené príkazy si precvičíme v nasledujúcom príklade.

Príklad SUB-2: Program SUB-1 (predchádzajúci príklad) doplňte o podprogramy na vytvorenie prázdneho súboru, nazistenie počtu zložiek súboru, na pridanie zložky na koniec súboru a na odstránenie zvolenej zložky zo súboru.
Analýza: Program SUB-2 je určený na ozrejmenie základných techník práce s údajovým typom súbor. Ošetrili sme otvorenie súboru funkciou IOResult, ak súbor so zadaným menom na disku neexistuje, vypíše sa správa: Súbor neexistuje! Najkomplikovanejšie je odstránenie zvolenej zložky zo súboru, kde sa musí použiť aj pomocný súbor (zo zdrojového súboru sa číta a do pomocného zapisuje), ktorý po zapísaní zvolených zložiek je premenovaný na pôvodný zdrojový súbor (pôvodný súbor musí byť najprv vymazaný).

```
program SUB_2;
uses  crt;
var   F:file of integer;
      FM:string[12];
      ODP:char;
      CHYBA:boolean;

procedure PRAZDNY;
  var FM:string[12];
  begin
  write('Meno suboru na disku: ');
  readln(FM);
  assign(F,FM);
  rewrite(F);
  close(F)
```

```
  end;

procedure VSTUP;
  var FM:string[12];
      X:integer;
  begin
  write('Meno suboru na disku: ');
  readln(FM);
  assign(F,FM);
  rewrite(F);
  repeat
    write('Vlozit cislo: ');
    readln(X);
    write(F,X);
    writeln('Vlozit dalsie cislo (nie - N) ?':52);
  until upcase(readkey)='N';
  close(F);
  end;

procedure VYSTUP;
  var FM:string[12];
      X:integer;
  begin
  write('Meno suboru na disku: ');
  readln(FM);
  assign(F,FM);
  {$I-} reset(F); {$I+}
  CHYBA:=IOResult<>0;
  if CHYBA
    then writeln('Subor neexistuje!')
    else begin
         if filesize(F)=0 then writeln('Subor je prazdny!');
         while not eof(F) do
           begin
           read(F,X);
           write(X:4)
           end;
         close(F)
         end;
  readln
  end;

procedure POCET;
  var FM:string[12];
  begin
  write('Meno suboru na disku: ');
  readln(FM);
  assign(F,FM);
  {$I-} reset(F); {$I+}
  CHYBA:=IOResult<>0;
  if CHYBA
    then writeln('Subor neexistuje!')
    else begin
         writeln('Pocet zloziek: ',filesize(F));
         close(F);
         end;
  readln
  end;

procedure PRIDAT_NA_KONIEC;
  var FM:string[12];
      X:integer;
  begin
  write('Meno suboru na disku: ');
  readln(FM);
  assign(F,FM);
```

```
  {$I-} reset(F); {$I+}
  CHYBA:=IOResult<>0;
  if CHYBA
    then writeln('Subor neexistuje!')
    else begin
         write('Pridat hodnotu: ');
         readln(X);
         seek(F,filesize(F));
         write(F,X);
         close(F);
         end
  end;

procedure ODSTRANIT;
  var FPom:file of integer;
      FM:string[12];
      X,HlHodnota:integer;
  begin
  write('Meno suboru na disku: ');
  readln(FM);
  assign(F,FM);
  {$I-} reset(F); {$I+}
  CHYBA:=IOResult<>0;
  if CHYBA
    then writeln('Subor neexistuje!')
    else begin
         assign(FPom,'POMOCNY.$$$');
         rewrite(FPom);
         write('Odstranit hodnotu: ');
         readln(HlHodnota);
         while not eof(F) do
           begin
           read(F,X);
           if X<>HlHodnota then write(FPom,X)
           end;
         close(F); close(FPom);
         assign(F,FM); erase(F);
         assign(FPom,'POMOCNY.$$$'); rename(FPom,FM)
         end
  end;

BEGIN
repeat
  clrscr;
  gotoxy(1,6);
  writeln('Vytvorit prazdny subor ....... S':50);
  writeln;
  writeln('Vytvorit neprazdny subor ..... V':50);
  writeln;
  writeln('Zobrazit subor ............... Z':50);
  writeln;
  writeln('Pocet zloziek suboru ......... P':50);
  writeln;
  writeln('Pridat zlozku na koniec ...... C':50);
  writeln;
  writeln('Odstranit zlozku zo suboru ... O':50);
  writeln;
  writeln('Koniec ....................... K':50);
  repeat ODP:=upcase(readkey); until ODP in ['S','V','Z','P','C','O','K'];
  clrscr;
  case ODP of
    'S' : PRAZDNY;
    'V' : VSTUP;
    'Z' : VYSTUP;
    'P' : POCET;
    'C' : PRIDAT_NA_KONIEC;
```

```
    'O' : ODSTRANIT;
  end
until ODP='K'
END.
```

Príklad SUB-3: Kartotéku v programe ZAZ_2 pracujúcu s poľom prerobte na „skutočnú“ kartotéku pracujúcu so súborom.

Analýzy: Program využíva len už vyššie použité procedúry a funkcie a preto ho nebudeme podrobnejšie opisovať. Odporúčame premenovať si súbor ZAZ-2 na SUB-3 a potom urobiť príslušné zmeny.

```
program SUB_3;
uses  crt;
const dmena=15;
      dadr=25;
type  karta=record
           ec:integer;
           meno:string[dmena];
           adr:record
               miesto,
               ulica:string[dadr];
               end;
            end;
      kartoteka=file of karta;
var   f:kartoteka;
      k:karta;
      q:char;

procedure vstup_karta;
  var pocet:integer;
  begin
  reset(f);
  pocet:=filesize(f);
  seek(f,pocet);
  with k,adr do
    begin
    inc(pocet);
    ec:=pocet;
    gotoxy(1,8);
    writeln('Evidencne cislo : ':40,ec);
    writeln;
    write('Meno a priezvisko : ':40);readln(meno);
    writeln;
    write('  Adresa - miesto : ':40);readln(miesto);
    writeln;
    write('          -  ulica : ':40);readln(ulica);
    end;
  write(f,k);
end;

procedure vystup_karta;
   begin
   clrscr;
   gotoxy(1,8);
   with k,adr do
     begin
     writeln('  Evidencne cislo : ':40,ec);
     writeln;
     writeln('Meno a priezvisko : ':40,meno);
     writeln;
     writeln('  Adresa - miesto : ':40,miesto);
     writeln;
     writeln('          - ulica : ':40,ulica);
     end;
   readln;
```

```
   end;

procedure vystup_kartoteka;
  begin
  reset(f);
  while not eof(f) do
    begin
    read(f,k);
    vystup_karta;
    end;
  end;

procedure najdi;
  var hlmeno:string[dmena];
  begin
  gotoxy(20,12);
  write('Zadaj meno hladaneho: ');
  readln(hlmeno);
  reset(f);
  while not eof(f) do
    begin
    read(f,k);
    if k.meno=hlmeno then vystup_karta
    end
  end;

procedure odstran;
  var hlmeno:string[dmena];
      fn:kartoteka;
  begin
  gotoxy(20,12);
  write('Zadaj meno na odstranenie: ');
  readln(hlmeno);
  assign(fn,'pracov.pom');
  reset(f);rewrite(fn);
  while not eof(f) do
    begin
    read(f,k);
    if k.meno<>hlmeno then write(fn,k);
    end;
  close(f); close(fn);
  assign(f,'pracov.dat');  erase(f);
  assign(fn,'pracov.pom'); rename(fn,'pracov.dat');
  assign(f,'pracov.dat');
end;

procedure pocet;
  begin
  reset(f);
  gotoxy(30,12);
  writeln('Pocet pracovnikov: ',filesize(f));
  readln;
  end;

procedure menu;
  begin
  repeat
    clrscr;
    gotoxy(1,6);
    writeln('           ZALOZIT KARTU..................... 1');
    writeln;
    writeln('           NAJST PODLA MENA.................. 2');
    writeln;
    writeln('           ODSTRANIT S MENOM................. 3');
    writeln;
    writeln('           POCET PRACOVNIKOV................. 4');
```

```
    writeln;
    writeln('            ZOZNAM PRACOVNIKOV.................. 5');
    writeln;
    writeln('            KONIEC.............................. 0');
    q:=readkey;
    clrscr;
    case q of
      '1': vstup_karta;
      '2': najdi;
      '3': odstran;
      '4': pocet;
      '5': vystup_kartoteka;
    end;
  until q='0';
  close(f);
  end;

BEGIN
clrscr;
assign(f,'pracov.dat');
gotoxy(10,12);
write('Vytvorit novy subor pracovnikov (ano - A) ? ');
{$I-} reset(f); {$I+}
if (IOResult<>0) or (upcase(readkey)='A') then rewrite(f);
menu;
END.
```

Príklad SUB-4: Vytvorte program, ktorý spojí dva súbory do jedného. Nech súbory obsahujú najviac dvadsaťznakové slová.

Analýza: Podprogram zabezpečujúci spojenie dvoch súborov je pomerne jednoduchý. Môžeme nastaviť prístupovú premennú na koniec v prvom súbore a pokračovať v zápise prvkov z druhého súboru. Ak má mať spojený súbor nový názov, stačí prvý súbor (v ňom sú všetky prvky) premenovať. Program sme doplnili aj o utriedenie prvkov súboru bubblesortom. Na triedenie súborov síce existujú špeciálne triediace algoritmy, pre zaujímavosť sme sa však „pohrali“ s bubblesortom. Všimnite si, že dve rôzne logické premenné súboru majú priradené to isté fyzické meno. Je to možné len za určitých podmienok (v bubblesorte sa nemení počet zložiek súboru, len ich poradie) a preto sa takýmto „perličkám“ radšej vyhýbajte.

```
program SUB_4;
uses  crt;
const DLSLOVA=20;
type  SLOVO=string[DLSLOVA];
      SUBOR=file of SLOVO;
      NAZOV=string[12];
var   FM1,FM2,FMS:NAZOV;

procedure VSTUP(FM:NAZOV);
  var F:SUBOR;
      S:SLOVO;
  begin
  assign(F,FM);
  rewrite(F);
  repeat
    write('Vlozit slovo: ');
    readln(S);
    write(F,S);
    writeln('Koniec - stlac 0':45)
  until upcase(readkey)='0';
  close(F)
  end;

procedure VYSTUP(FM:NAZOV);
```

```
  var F:SUBOR;
      S:SLOVO;
  begin
  assign(F,FM);
  reset(F);
  while not eof(F) do
    begin
    read(F,S);
    write(S,' ')
    end;
  writeln;
  close(F)
  end;

procedure SPOJ(FM1,FM2,FMS:NAZOV);
  var F1,F2:SUBOR;
      S:SLOVO;
  begin
  assign(F1,FM1); reset(F1);
  assign(F2,FM2); reset(F2);
  seek(F1,filesize(F1));
  while not eof(F2) do
    begin
    read(F2,S);
    write(F1,S)
    end;
  close(F1); close(F2);
  assign(F1,FMS); erase(F1);
  assign(F1,FM1); rename(F1,FMS);
  end;

procedure BUBBLESORT(FM:NAZOV);
  var F,FPom:SUBOR;
      A,B:SLOVO;
      DL,PP,I:integer;
  begin
  assign(F,FM); assign(FPom,FM);
  reset(F);
  DL:=filesize(F);
  for PP:=1 to DL-1 do
    begin
    reset(F); reset(FPom);
    read(F,A);
    for I:=1 to DL-PP do
      begin
      read(F,B);
      if A<B
        then begin write(FPom,A); A:=B end
        else write(FPom,B)
      end;
      write(FPom,A)
    end;
  close(F);close(FPom)
  end;

BEGIN
clrscr;
write('Meno prveho suboru na disku: '); readln(FM1);
VSTUP(FM1);
write('Meno druheho suboru na disku: '); readln(FM2);
VSTUP(FM2);
write('Meno spojeneho suboru na disku: '); readln(FMS);
clrscr;
writeln; writeln('Prvy subor:');
VYSTUP(FM1);
writeln; writeln('Druhy subor:');
```

```
VYSTUP(FM2);
writeln; writeln('Spojeny subor:');
SPOJ(FM1,FM2,FMS);
VYSTUP(FMS);
writeln; writeln('Utriedeny subor:');
BUBBLESORT(FMS);
VYSTUP(FMS);
readln
END.
```

Textové súbory

Existujú súbory dát, ktoré sú členené na riadky. Takéto súbory nazývame textové a ich typ sa označuje štandardným identifikátorom typu *text*. V textovom súbore F okrem funkcie eof(F) je deklarovaná aj funkcia eoln(F) (end of line), slúžiaca na rozpoznanie konca riadku. Príkazy read(F,P) a write(F,V) sú rozšírené o príkazy readln(F,P) – z textového súboru F sa prečíta lexikálna jednotka (hodnota) zodpovedajúca typu premennej P a prejde sa na nový riadok, a writeln(F,V) – do textového súboru F sa zapíše hodnota výrazu V a oddeľovač riadkov eoln.

Najznámejšie súbory typu text sú štandardný vstupný súbor **input**, ktorému je priradené vstupné zariadenia klávesnica, a štandardný výstupný súbor **output**, ktorému je priradené výstupné zariadenie monitor. Definovanie ich typu ako súborov typu text je implicitné. K otvoreniu oboch súborov dochádza automaticky po spustení programu. Príkazy read(input,P), readln(input,P), write(output,V) a writeln(output,V) sa používajú bez identifikátorov input a output.

Príklad SUB-5: Vytvorte program, ktorý bude čítať znaky z klávesnice (vstupný súbor input) a pre každý riadok zobrazí, koľkokrát sa v ňom vyskytuje jeho prvý znak.

Analýza: Program musí obsahovať dva vnorené cykly. Vonkajší cyklus zabezpečuje čítanie znakov, až kým nie je načítaný koniec súboru. Vnútorný cyklus číta znaky, až kým nie je koniec riadku, potom musí odriadkovať. Vložiť znak koniec riadku nie je problém, stačí stlačiť Enter (chr(13)). Ak si spomenieme, že „v starom dobrom DOSe“ sa koniec súboru vkladal kombináciou kláves Ctrl+Z, sme „zachránení“.

```
program SUB_5;
var pocet:integer;
    z,s:char;
begin
writeln('Vkladaj text, ukoncenie - Ctrl+Z !!!');
while not eof do
  begin
  read(z);
  pocet:=1;
  while not eoln do
    begin
    read(s);
    if s=z then inc(pocet);
    end;
  writeln('Pocet vyskytov znaku ',z,' je ',pocet,'-krat');
  readln;
  end;
end.
```

Príklad SUB-6: Pokúste sa vysvetliť, prečo oba ďalej uvedené programy SUB-6a aj SUB-6b pracujú rovnako. Ich úlohou je uložiť do súboru na disk text z klávesnice.

```
Program SUB_6a;
var z:char;
    T:text;
    FM:string[12];

BEGIN
write('Meno suboru na disku: ');
readln(FM);
assign(T,FM);
rewrite(T);
writeln('Vkladaj text, koniec - vloz Ctrl+Z !!!');
while not eof do
  begin
  while not eoln do
    begin
    read(z);
    write(T,z)
    end;
  readln;
  writeln(T)
  end;
writeln;
writeln('Ulozeny subor:');
reset(T);
while not eof(T) do
  begin
  while not eoln(T) do
    begin
    read(T,z);
    write(z);
    end;
  readln(T);
  writeln
  end;
close(T);
readln
END.

program SUB_6b;
uses crt;
var  z:char;
     T:text;
     FM:string[12];

BEGIN
clrscr;
write('Meno suboru na disku: ');
readln(FM);
assign(T,FM);
rewrite(T);
writeln('Vkladaj text, koniec - stlac `');
repeat
  read(z);
  if z<>'`' then write(T,z)
until z='`';
writeln;
writeln('Ulozeny subor:');
reset(T);
while not eof(T) do
  begin
  read(T,z);
  write(z);
  end;
close(T);
readln;readln
END.
```

Neriešené úlohy na súbor:

7. Kartotéku z príkladu SUB-3 doplňte o položku rodné číslo a podprogram, ktorý zistí počet osôb mužského a ženského pohlavia v kartotéke (u mužov je v rodnom čísle na 3. mieste 0 alebo1, u žien 5 alebo 6).
8. Kartotéku z príkladu SUB-3 doplňte o položku plat a podprogram, ktorý vypíše osoby, ktorých plat spĺňa zadané kritériá (menší, väčší, rovný).
9. Vytvorte program, ktorý zo súboru reálnych čísel prepíše do nového súboru len celé čísla.
10. Úloha z jednorozmerného poľa o vyhľadávaní, počte výskytov, miesta výskytu a pod. zrealizujte pre súbor celých čísel (znakov, reťazcov).
11. Vytvorte program, ktorý z daného textového súboru odstráni všetky viacnásobné medzery a prázdne riadky.
12. Vytvorte program slúžiaci ako elektronický dvojjazyčný slovník.
13. Vytvorte program, ktorý bude čítať znaky z klávesnice a pre každý riadok určí súčet ordinálnych čísel znakov vyskytujúcich sa v riadku.

Dynamické premenné

Údajové typy, s ktorými sme sa zaoberali doteraz, patria medzi tzv. statické údajové typy. Premenné prislúchajúce statickým typom sú zavedené v úseku definícií a deklarácií a je im trvalo vyhradené miesto v pamäti počas behu programu. Počet a rozsah premenných sa počas práce v bloku, t.j. aj v programe, nemôže meniť.

Jazyk pascal umožňuje vytvárať premenné nie len v úseku deklarácií premenných, ale aj v príkazovej časti. Dokonca možno tieto premenné v príkazovej časti aj rušiť. Premenné s týmito vlastnosťami nazývame **dynamické premenné** a príslušné údajové typy nazývame dynamické údajové štruktúry.

Z toho, čo sme uviedli doteraz, vyplýva, že dynamické premenné nemožno zaviesť v úseku deklarácií a teda zabezpečiť prístup k ich hodnotám pomocou ich identifikátorov. Dynamické premenné vznikajú a zanikajú počas realizácie programu a sprístupnenie ich hodnôt sa robí pomocou nového údajového typu **ukazovateľ**.

Ak t je identifikátor nejakého typu, tak typ ukazovateľ definujeme zápisom:

```
type u = ^ t
```

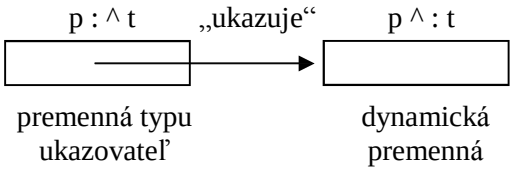
kde u je meno (identifikátor) typu ukazovateľ. Hovoríme aj, že typ t je zviazaný s typom u a množina hodnôt premennej typu ukazovateľ „ukazuje“ na prvky typu t.

Napríklad:

```
type   UKAZ1 = ^ integer;
       UKAZ2 = ^ PRVOK;
```

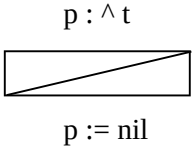
Ak p je premenná typu ukazovateľ, tak premenná typu zviazaného s typom ukazovateľ (premenná typu t) sa nazýva dynamická premenná a označuje sa p^.

Grafická interpretácia:



Pomocou hodnoty premennej typu ukazovateľ sprístupňujeme dynamickú premennú. V úseku definícií a deklarácií je zavedená len premenná typu ukazovateľ! Do množiny hodnôt premennej typu ukazovateľ patrí vždy aj hodnota *nil*, jediná konštanta typu ukazovateľ, ktorá neukazuje na nijaký prvok (na žiadnu dynamickú premennú).

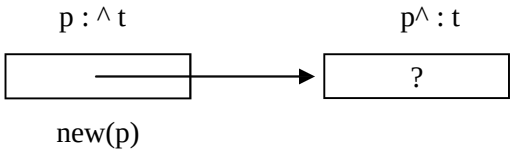
Graficky:



Premenná p má síce priradenú hodnotu, ale „neukazuje“ na žiadnu dynamickú premennú.

Príkazom **new(p)**, kde p je premenná typu ukazovateľ, sa vytvorí dynamická premenná typu t s nedefinovanou hodnotou a ukazovateľ na túto premennú sa uloží do premennej p. Príkazom new(p) priraďujeme premennej p hodnotu, ktorou je zodpovedajúca dynamická premenná a ktorú označujeme p^.

Graficky:



Príkazom **dispose(p)**, kde p je premenná typu ukazovateľ, sa zruší dynamická premenná p^, na ktorú ukazovala premenná p. Hodnota premennej p nie je po ukončení príkazu dispose(p) definovaná. Pamäťový priestor, ktorý zaberala dynamická premenná, sa dal k dispozícii na ďalšie použitie procesorom.

Zhrnutie:

var p : ^t ; p

(v pamäti počítača sa vyhradí pamäťové miesto pre statickú premennú typu ukazovateľ, t.j. 32 bitov pre „akúsi“ adresu)

p := nil ; p

(do pamäťového miesta vyhradeného pre premennú typu ukazovateľ sa uloží číslo neexistujúcej adresy, napríklad 0)

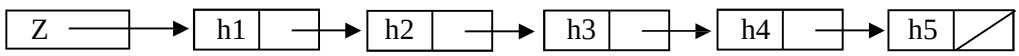
new (p) ; p ?

p^ : t

(do pamäťového miesta vyhradeného pre premennú typu ukazovateľ sa uloží adresa vytvorenej dynamickej premennej p^)

Lineárny jednosmernezreťazený zoznam

Z dynamických štruktúr sa najčastejšie využíva lineárny jednosmernezreťazený zoznam. Graficky ho možno znázorniť:



kde Z je premenná typu ukazovateľ (pointer), ktorá ukazuje na prvú dynamickú premennú v zozname, ktorého prvky sú typu záznam (record) s dvoma položkami: s hodnotou (h1 až h5) a s ukazovateľom na ďalší prvok zoznamu. Ukazovateľ posledného prvku zoznamu už nemá kde ukazovať a preto má hodnotu nil.
Postup v zozname je sekvenčný, od ukazovateľa na prvý prvok cez ukazovatele na nasledujúci prvok až na koniec zoznamu. V úseku definícií a deklarácií je zavedený len ukazovateľ na zoznam (premenná Z typu ^t) a typ dynamickej premennej (t). Vlastné prvky zoznamu, dynamické premenné, vznikajú podľa potreby počas práce programu príkazom new. Na pohyb v zozname používame pomocnú premennú typu ukazovateľ a premennú Z nechávame väčšinou ukazovať vždy na začiatok zoznamu (inak sa nemáme možnosť dostať na začiatok zoznamu).

Program pracujúci so zoznamom by začínal:

```
type    UKAZ=^PRVOK;
        PRVOK=record
                    hodnota: typ_hodnoty;
                    dalsi: UKAZ
                end;
var Z : UKAZ;
```

Prácu s lineárnym jednosmernezreťazeným zoznamom demonštruje nasledujúci príklad.

Program DYN-1: Vytvorte program umožňujúci vo forme menu vytvoriť prázdny zoznam, naplniť ho hodnotami, vypísať hodnoty zo zoznamu, odstrániť zvolenú hodnotu, uložiť hodnoty zoznamu do súboru a naopak, načítať ich zo súboru do zoznamu.

Analýza: Program je ponechaný na samoštúdium, odporúčame však kresliť si jednotlivé situácie pomocou zavedenej symboliky. Všimnite si, v akom poradí prvky ukladané do zoznamu!

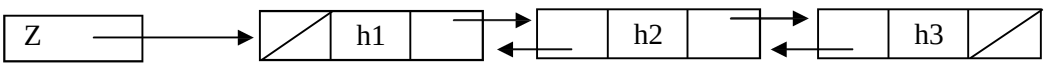
```
program DYN_1;
uses crt;
type ukazovatel=^prvok;
     prvok=record
             h:integer;
             u:ukazovatel
           end;
     subor=file of integer;
var  z:ukazovatel;
     sub:subor;
     q:char;

procedure vytvor(var z:ukazovatel);
  begin
  z:=nil
  end;

function prazdny(z:ukazovatel):boolean;
  begin
  prazdny:=z=nil
  end;

procedure nahodne(var z:ukazovatel);
  var p:ukazovatel;
      i:integer;
  begin
  z:=nil; randomize;
  for i:=1 to random(10)+2 do
    begin
    new(p);
    with p^ do
      begin
      h:=random(10);
      u:=z;z:=p;
      end;
    end;
  end;

procedure napln(var z:ukazovatel);
  var p:ukazovatel;
      q:char;
  begin
  z:=nil;
```

```
  repeat
    writeln('Vlozit prvok ( ano -> A ) ?');q:=upcase(readkey);
    if q='A'
      then begin
           new(p);
           with p^ do
             begin
             write('Hodnota: ');readln(h);
             u:=z;z:=p;
             end;
           end;
  until q<>'A';
  end;

procedure dopln(var z:ukazovatel);
  var p:ukazovatel;
      q:char;
  begin
  repeat
    writeln('Vlozit prvok ( ano -> A ) ?');q:=upcase(readkey);
    if q='A'
      then begin
           new(p);
           with p^ do
             begin
             write('Hodnota: ');readln(h);
             u:=z;z:=p;
             end;
           end;
  until q<>'A';
  end;

procedure vymaz(var z:ukazovatel);
  var p,p1:ukazovatel;
  begin
  if not prazdny(z)
    then begin
         repeat
           p:=z;
           writeln(p^.h:5,' - vymazat ( ano -> A ) ');q:=upcase(readkey);
           if q='A'
             then begin
                  z:=p^.u;
                  dispose(p);
                  p:=z
                  end
             else begin
                  p1:=p;
                  p:=p^.u
                  end;
         until q<>'A';
         while p<>nil do
           begin
           writeln(p^.h:5,' - vymazat ( ano -> A ) ');q:=upcase(readkey);
             if q='A'
               then begin
                    p1^.u:=p^.u;
                    dispose(p);
                    p:=p1^.u
                    end
               else begin
                    p1:=p;
                    p:=p^.u
                    end;
           end
         end
```

```
  end;

procedure vypis(z:ukazovatel);
  var p:ukazovatel;
  begin
  if prazdny(z)
    then writeln('Zoznam je prazdny')
    else begin
         p:=z;
         write(p^.h:5);
         while p^.u<>nil do
           begin
           p:=p^.u;
           write(p^.h:5)
           end
         end
  end;

procedure utried(var z:ukazovatel);
  var p,q,k:ukazovatel; pom:integer;
      bola_vymena:boolean;
  begin
  if (z<>nil)and(z^.u<>nil)
    then repeat
         p:=z^.u; q:=z; bola_vymena:=false;
         while (p<>nil)and(p<>k) do
           begin
           if p^.h<q^.h
             then begin
                  pom:=p^.h; p^.h:=q^.h; q^.h:=pom;
                  bola_vymena:=true
                  end;
           q:=p; p:=p^.u
           end;
         k:=q
         until not bola_vymena;
  end;

procedure zapis_sub(z:ukazovatel);
  var p:ukazovatel;
  begin
  rewrite(sub);
  p:=z;
  repeat
    write(sub,p^.h);
    p:=p^.u
  until p^.u=nil;
  write(sub,p^.h);
end;

procedure citaj_sub(var z:ukazovatel);
  var p:ukazovatel;
  begin
  reset(sub);
  z:=nil;
  while not eof(sub) do
    begin
    new(p);
    with p^ do
      begin
      read(sub,h);
      u:=z;z:=p;
      end;
    end;
end;
```

```
BEGIN
assign(sub,'dynamic.sub');
rewrite(sub);
vytvor(z);
repeat
  clrscr;gotoxy(1,5);
  WRITELN('VYTVOR PRAZDNY ..... 1':53);
  WRITELN('VYTVOR NAHODNE ..... 2':53);
  WRITELN('NAPLN ..... 3':53);
  WRITELN('DOPLN ..... 4':53);
  WRITELN('VYMAZ ..... 5':53);
  WRITELN('UTRIED ..... 6':53);
  WRITELN;
  WRITELN('VYPIS ..... 7':53);
  WRITELN;
  WRITELN('NACITAJ ZO SUBORU ..... 8':53);
  WRITELN('ULOZ DO SUBORU ..... 9':53);
  WRITELN;
  WRITELN('KONIEC ..... 0':53);
  q:=readkey;clrscr;
  case q of
    '1':vytvor(z);
    '2':nahodne(z);
    '3':napln(z);
    '4':dopln(z);
    '5':vymaz(z);
    '6':utried(z);
    '7':vypis(z);
    '8':citaj_sub(z);
    '9':zapis_sub(z);
  end;
if (q>'2') and (q<'5') or (q='7') then readln
until q='0';
close(sub)
END.
```

Na záver uvedieme príklad, ktorý využíva lineárny obojsmernezreťazený zoznam, ktorý možno graficky znázorniť (3-prvkový):



Program DYN-2: Vytvorte program na výpočet cifier n! s využitím dynamických dátových štruktúr.

Analýza: Výpočet n! zlyháva u štandardných celočíselných typov už pri n=13. Nasledujúci program počíta cifry n-faktoriálu pre „neobmedzené“ n. Podstatou je „manuálne“ násobenie, cifru za cifrou.

```
program DYN_3;
uses  crt;
const Esc=chr(27);
type  ukaz=^prvok;
      prvok=record
              cifra:0..9;
              pred,za:ukaz
            end;
var   cislica,prva:ukaz;
      posl:pointer;
      i,n,pocet,prenos,stl,ria:integer;
                          sucin:integer; { longint }
procedure text_pocitam;
  begin
    stl:=wherex;ria:=wherey;
    gotoxy(34,1);textcolor(15+128);writeln('POCITAM  ');
```

```
    gotoxy(stl,ria);normvideo;
  end;

procedure text_faktorial;
  begin
    stl:=wherex;ria:=wherey;if ria=1 then ria:=2;
    gotoxy(34,1);writeln('FAKTORIAL');
    gotoxy(stl,ria)
  end;

BEGIN
repeat
  clrscr;
  text_faktorial;
  write('N = ');readln(n);write(n,' ! = ');
  text_pocitam;
  new(cislica);
  with cislica^ do begin cifra:=1;pred:=nil;za:=nil end;
  posl:=cislica;prva:=cislica;prenos:=0;sucin:=1;
  for i:=1 to n do
    begin
    cislica:=posl;
    repeat
      with cislica^ do
        begin
        sucin:=cifra*i+prenos;
        cifra:=sucin mod 10;
        prenos:=sucin div 10;
        cislica:=pred
        end
    until cislica=nil;
    while prenos<>0 do
      begin
      new(cislica);
      with cislica^ do
        begin
        pred:=nil;za:=prva;prva^.pred:=cislica;prva:=cislica;
        cifra:=prenos mod 10;prenos:=prenos div 10;
        end;
      end;
    end;
  text_faktorial;
  cislica:=prva;pocet:=0;
  repeat
    write(cislica^.cifra);pocet:=pocet+1;cislica:=cislica^.za;
    dispose(prva); prva:=cislica;
  until cislica=nil;
  writeln;writeln('Pocet cifier v cisle: ',pocet, '        Koniec - Esc')
until readkey=Esc
END.
```

Neriešené úlohy na dynamické dátové štruktúry:

3. Zásobník je dynamická dátová štruktúra, na ktorej sú definované operácie:
   a) pridať prvok na vrch zásobníka
   b) odobrať prvok z vrchu zásobníka
   c) zistiť, či je zásobník prázdny
   d) vytvoriť prázdny zásobník

   Zrealizujte zoznam pracujúci ako zásobník (zoznam typu LIFO, last-in first-out).
4. Vytvorte program kontrolujúci správnosť ozátvorkovania výrazu s využitím v príklade 3 opísaného zásobníka.
5. Rad je dynamická dátová štruktúra, na ktorej sú definované operácie:
   a) pridať prvok na koniec radu
   b) odobrať prvok zo začiatku radu
   c) zistiť, či je rad prázdny
   d) vytvoriť prázdny rad

   Zrealizujte zoznam pracujúci ako rad (zoznam typu FIFO, first-in first-out).
6. Vytvorte program na výpočet Fibonacciho čísel pomocou trojprvkového radu.

   Fibonacciho postupnosť: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,...

   vzorcami: $F_0 = 0 \quad F_1 = 1 \quad F_i = F_{i-1} + F_{i-2} \quad \text{pre } i \geq 2$
7. Vytvorte program na výpočet hodnôt Pascalovho trojuholníka pomocou radu.
8. Vytvorte podprogramy na vyhľadávanie, zistenie počtu výskytov, miesta výskytu a pod. prvkov so zvolenými vlastnosťami v lineárnom jednosmernezreťazenom zozname.
9. Vytvorte program, ktorý spojí dva lineárne jednosmernezreťazené zoznamy do jedného.